DIN EN 61508-7
(VDE 0803-7)

DIN

Diese Norm ist zugleich eine **VDE-Bestimmung** im Sinne von VDE 0022. Sie ist nach Durchführung des vom VDE-Präsidium beschlossenen Genehmigungsverfahrens unter der oben angeführten Nummer in das VDE-Vorschriftenwerk aufgenommen und in der „etz Elektrotechnik + Automation“ bekannt gegeben worden.

VDE

ICS 35.240.50

Ersatz für
DIN EN 61508-7
(VDE 0803-7):2003-06
Siehe jedoch Anwendungsbeginn

Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme – Teil 7: Überblick über Verfahren und Maßnahmen (IEC 61508-7:2010); Deutsche Fassung EN 61508-7:2010

Functional safety of electrical/electronic/programmable electronic safety-related systems –
Part 7: Overview of techniques and measures
(IEC 61508-7:2010);
German version EN 61508-7:2010

Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité –
Partie 7: Présentation de techniques et mesures
(CEI 61508-7:2010);
Version allemande EN 61508-7:2010

Gesamtumfang 162 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Preisgr. 66 K
VDE-Vertr.-Nr. 0803018

Einzelverkauf und Abonnements durch VDE VERLAG GMBH, 10625 Berlin
Einzelverkauf auch durch Beuth Verlag GmbH, 10772 Berlin

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2010-05-01 angenommene Europäische Norm als DIN-Norm ist 2011-02-01.

Daneben darf DIN EN 61508-7 (VDE 0803-7):2003-06 noch bis 2013-05-01 angewendet werden.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN EN 61508-7 (VDE 0803-7):2009-06.

Für diese Norm ist das nationale Arbeitsgremium GK 914 „Funktionale Sicherheit elektrischer, elektronischer und programmierbarer elektronischer Systeme (E, E, PES) zum Schutz von Personen und Umwelt“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (www.dke.de) zuständig.

Die enthaltene IEC-Publikation wurde vom SC 65A „System aspects“ erarbeitet.

Das IEC-Komitee hat entschieden, dass der Inhalt dieser Publikation bis zu dem Datum (maintenance result date) unverändert bleiben soll, das auf der IEC-Website unter „http://webstore.iec.ch“ zu dieser Publikation angegeben ist. Zu diesem Zeitpunkt wird entsprechend der Entscheidung des Komitees die Publikation
– bestätigt,
– zurückgezogen,
– durch eine Folgeausgabe ersetzt oder
– geändert.

Änderungen

Gegenüber DIN EN 61508-7 (VDE 0803-7):2003-06 wurden folgende Änderungen vorgenommen:

a) Beschreibung der Verfahren und Maßnahmen an den Stand der Technik angepasst;

b) Hinzufügung des Anhangs E „Überblick über Verfahren und Maßnahmen für den Entwurf von ASICs“;

c) Hinzufügung des Anhangs F „Definitionen der Eigenschaften der Software-Lebenszyklusphasen“;

d) Hinzufügung des Anhangs G „Anleitung zur Entwicklung von sicherheitsbezogener objektorientierter Software“.

Frühere Ausgaben

DIN EN 61508-7 (VDE 0803-7): 2003-06

Nationaler Anhang NA
(informativ)

Zusammenhang mit Europäischen und Internationalen Normen

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf eine Norm ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils neueste gültige Ausgabe der in Bezug genommenen Norm.

Für den Fall einer datierten Verweisung im normativen Text bezieht sich die Verweisung immer auf die in Bezug genommene Ausgabe der Norm.

Eine Information über den Zusammenhang der zitierten Normen mit den entsprechenden Deutschen Normen ist in Tabelle NA.1 wiedergegeben.

Tabelle NA.1

Europäische Norm	Internationale Norm	Deutsche Norm	Klassifikation im VDE-Vorschriftenwerk
EN 14882:2005	ISO/IEC 14882:2003	DIN EN 14882:2005-11	–
EN 60068-2-1	IEC 60068-2-1	DIN EN 60068-2-1 (VDE 0468-2-1)	VDE 0468-2-1
EN 60068-2-2	IEC 60068-2-2	DIN EN 60068-2-2 (VDE 0468-2-2)	VDE 0468-2-2
EN 60601 (alle Teile)	IEC 60601 (alle Teile)	DIN EN 60601 (VDE 0750) (alle Teile)	VDE 0750 (alle Teile)
EN 60812:2006	IEC 60812:2006	DIN EN 60812:2006-11	–
EN 60880:2009	IEC 60880:2006	DIN EN 60880 (VDE 0491-3-2):2010-03	VDE 0491-3-2
EN 61000-4-1:2007	IEC 61000-4-1:2006	DIN EN 61000-4-1 (VDE 0847-4-1):2007-10	VDE 0847-4-1
EN 61000-4-5:2006	IEC 61000-4-5:2005	DIN EN 61000-4-5 (VDE 0847-4-5):2007-06	VDE 0847-4-5
EN 61025:2007	IEC 61025:2006	DIN EN 61025:2007-08	–
EN 61069-5:1995	IEC 61069-5:1994	DIN EN 61069-5:1995-09	–
EN 61078:2006	IEC 61078:2006	DIN EN 61078:2006-10	–
EN 61160:2005	IEC 61160:2005	DIN EN 61160:2006-06	–
EN 61163-1:2006	IEC 61163-1:2006	DIN EN 61163-1:2007-06	–
EN 61164:2004	IEC 61164:2004	DIN EN 61164:2004-11	–
EN 61165:2006	IEC 61165:2006	DIN EN 61165:2007-02	–
EN 61326-3-1:2008	IEC 61326-3-1:2008	DIN EN 61326-3-1 Berichtigung 1 (VDE 0843-20-3-1 Berichtigung 1) :2009-04	VDE 0843-20-3-1 Berichtigung 1
EN 61326-3-2:2008	IEC 61326-3-2:2008	DIN EN 61326-3-2 (VDE 0843-20-3-2):2008-11	VDE 0843-20-3-2
EN 61508-1:2010	IEC 61508-1:2010	DIN EN 61508-1 (VDE 0803-1):2011-02	VDE 0803-1
EN 61508-2:2010	IEC 61508-2:2010	DIN EN 61508-2 (VDE 0803-2):2011-02	VDE 0803-2
EN 61508-3:2010	IEC 61508-3:2010	DIN EN 61508-3 (VDE 0803-3):2011-02	VDE 0803-3
EN 61508-4:2010	IEC 61508-4:2010	DIN EN 61508-4 (VDE 0803-4):2011-02	VDE 0803-4
EN 61508-6:2010	IEC 61508-6:2010	DIN EN 61508-6 (VDE 0803-6):2011-02	VDE 0803-6
EN 61511 (alle Teile)	IEC 61511 (alle Teile)	DIN EN 61511 (VDE 0810-1) (alle Teile)	VDE 0810-1 (alle Teile)
EN 61800-5-2	IEC 61800-5-2	DIN EN 61800-5-2 (VDE 0160-105-2)	VDE 0160-105-2
EN 62061:2005	IEC 62061:2005	DIN EN 62061 (VDE 0113-50):2005-10	VDE 0113-50
EN 62308:2006	IEC 62308:2006	DIN EN 62308:2007-07	–

Tabelle NA.1 *(fortgesetzt)*

EN 81346-1:2009	IEC 81346-1:2009	DIN EN 81346-1:2010-05	–
–	ISO/IEC 8652:1995	DIN ISO/IEC 8652:1996-09	–
EN ISO 9000	ISO 9000	DIN EN ISO 9000	–
–	IEC 104/445/CD:2007	E DIN IEC 60068-1 (VDE 0468-1):2008-03	VDE 0468-1
–	IEC 70/118/CD:2009	E DIN IEC 60529/A1 (VDE 0470-1/A1):2010-04	VDE 0470-1/A1
–	IEC 65B/725/CD:2009	E DIN IEC 61131-3:2009-12	–
–	IEC/TR 61000-5-2 :1997	–	–
–	IEC 61506:1997	–	–
–	IEC/TR 61508-0:2005	–	–
–	ISO/IEC 1539-1:2004	–	–
–	ISO 5807:1985	–	–
–	ISO/IEC 7185:1990	–	–
–	ISO/IEC 8631:1989	–	–
–	ISO 8807:1989	–	–
–	ISO/IEC 9899:1999	–	–
–	ISO/IEC 10206:1991	–	–
–	ISO/IEC 10514-1 :1996	–	–
–	ISO/IEC 10514-3 :1998	–	–
–	ISO/IEC 13817-1 :1996	–	–
–	ISO/IEC/TR 15942 :2000	–	–

Nationaler Anhang NB
(informativ)

Literaturhinweise

DIN EN 14882:2005-11, *Mit Kautschuk oder Kunststoff beschichtete Textilien – Bestimmung der Koeffizienten von Haftreibung und Bewegungsreibung; Deutsche Fassung EN 14882:2005*

DIN EN 60068-2-1 (VDE 0468-2-1), *Umgebungseinflüsse – Teil 2-1: Prüfverfahren – Prüfung A: Kälte*

DIN EN 60068-2-2 (VDE 0468-2-2), *Umgebungseinflüsse – Teil 2-2: Prüfverfahren – Prüfung B: Trockene Wärme*

DIN EN 60601 (VDE 0750) (alle Teile), *Medizinische elektrische Geräte*

DIN EN 60812:2006-11, *Analysetechniken für die Funktionsfähigkeit von Systemen – Verfahren für die Fehlzustandsart- und -auswirkungsanalyse (FMEA) (IEC 60812:2006); Deutsche Fassung EN 60812:2006*

DIN EN 60880 (VDE 0491-3-2):2010-03, *Kernkraftwerke – Leittechnik für Systeme mit sicherheitstechnischer Bedeutung – Softwareaspekte für rechnerbasierte Systeme zur Realisierung von Funktionen der Kategorie A (IEC 60880:2006); Deutsche Fassung EN 60880:2009*

DIN EN 61000-4-1 (VDE 0847-4-1):2007-10, *Elektromagnetische Verträglichkeit (EMV) – Teil 4-1: Prüf- und Messverfahren – Übersicht über die Reihe IEC 61000-4 (IEC 61000-4-1:2006); Deutsche Fassung EN 61000-4-1:2007*

DIN EN 61000-4-5 (VDE 0847-4-5):2007-06, *Elektromagnetische Verträglichkeit (EMV) – Teil 4-5: Prüf- und Messverfahren – Prüfung der Störfestigkeit gegen Stoßspannungen (IEC 61000-4-5:2005); Deutsche Fassung EN 61000-4-5:2006*

DIN EN 61025:2007-08, *Fehlzustandsbaumanalyse (IEC 61025:2006); Deutsche Fassung EN 61025:2007*

DIN EN 61069-5:1995-09, *Leittechnik für industrielle Prozesse – Ermittlung der Systemeigenschaften zum Zweck der Eignungsbeurteilung eines Systems – Teil 5: Eignungsbeurteilung der Systemverläßlichkeit (IEC 61069-5:1994); Deutsche Fassung EN 61069-5:1995*

DIN EN 61078:2006-10, *Techniken für die Analyse der Zuverlässigkeit – Zuverlässigkeitsblockdiagramm und Boole'sche Verfahren (IEC 61078:2006); Deutsche Fassung EN 61078:2006*

DIN EN 61160:2006-06, *Entwicklungsbewertung (IEC 61160:2005); Deutsche Fassung EN 61160:2005*

DIN EN 61163-1:2007-06, *Zuverlässigkeitsvorbehandlung durch Beanspruchung – Teil 1: Instandsetzbare Baugruppen, losweise gefertigt (IEC 61163-1:2006); Deutsche Fassung EN 61163-1:2006*

DIN EN 61164:2004-11, *Zuverlässigkeitswachstum – Statistische Prüf- und Schätzverfahren (IEC 61164:2004); Deutsche Fassung EN 61164:2004*

DIN EN 61165:2007-02, *Anwendung des Markoff-Verfahrens (IEC 61165:2006); Deutsche Fassung EN 61165:2006*

DIN EN 61326-3-1 Berichtigung 1 (VDE 0843-20-3-1 Berichtigung 1):2009-04, *Elektrische Mess-, Steuer-, Regel- und Laborgeräte – EMV-Anforderungen – Teil 3-1: Störfestigkeitsanforderungen für sicherheitsbezogene Systeme und für Geräte, die für sicherheitsbezogene Funktionen vorgesehen sind (Funktionale Sicherheit) – Allgemeine industrielle Anwendungen (IEC 61326-3-1:2008); Deutsche Fassung EN 61326-3-1:2008, Berichtigung zu DIN EN 61326-3-1 (VDE 0843-20-3-1):2008-11*

DIN EN 61326-3-2 (VDE 0843-20-3-2):2008-11, *Elektrische Mess-, Steuer-, Regel- und Laborgeräte – EMV-Anforderungen –Teil 3-2: Störfestigkeitsanforderungen für sicherheitsbezogene Systeme und für Geräte, die für sicherheitsbezogene Funktionen vorgesehen sind (Funktionale Sicherheit) – Industrielle Anwendungen in spezifizierter elektromagnetischer Umgebung (IEC 61326-3-2:2008); Deutsche Fassung EN 61326-3-2:2008*

DIN EN 61508-1 (VDE 0803-1):2011-02, *Funktionale Sicherheit sicherheitsbezogener elektrischer/ elektronischer/programmierbarer elektronischer Systeme – Allgemeine Anforderungen (IEC 61508-1:2010); Deutsche Fassung EN 61508-1:2010*

DIN EN 61508-2 (VDE 0803-2):2011-02, *Funktionale Sicherheit sicherheitsbezogener elektrischer/ elektronischer/programmierbarer elektronischer Systeme – Teil 2: Anforderungen an sicherheitsbezogene elektrische/elektronische/programmierbare elektronische Systeme (IEC 61508-2:2010); Deutsche Fassung EN 61508-2:2010*

DIN EN 61508-3 (VDE 0803-3):2011-02, *Funktionale Sicherheit sicherheitsbezogener elektrischer/ elektronischer/programmierbarer elektronischer Systeme – Teil 3: Anforderungen an Software (IEC 61508-3:2010); Deutsche Fassung EN 61508-3:2010*

DIN EN 61508-4 (VDE 0803-4):2011-02, *Funktionale Sicherheit sicherheitsbezogener elektrischer/ elektronischer/programmierbarer elektronischer Systeme – Teil 4: Begriffe und Abkürzungen (IEC 61508-4:2010); Deutsche Fassung EN 61508-4:2010*

DIN EN 61508-6 (VDE 0803-6):2011-02, *Funktionale Sicherheit sicherheitsbezogener elektrischer/ elektronischer/programmierbarer elektronischer Systeme – Teil 6: Anwendungsrichtlinie für IEC 61508-2 und IEC 61508-3 (IEC 61508-6:2010); Deutsche Fassung EN 61508-6:2010*

DIN EN 61511 (VDE 0810-1) (alle Teile), *Funktionale Sicherheit – Sicherheitstechnische Systeme für die Prozessindustrie*

DIN EN 61800-5-2 (VDE 0160-105-2), *Elektrische Leistungsantriebssysteme mit einstellbarer Drehzahl – Teil 5-2: Anforderungen an die Sicherheit – Funktionale Sicherheit*

DIN EN 62061 (VDE 0113-50):2005-10, *Sicherheit von Maschinen – Funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und programmierbarer elektronischer Steuerungssysteme (IEC 62061:2005); Deutsche Fassung EN 62061:2005*

DIN EN 62308:2007-07, *Zuverlässigkeit von Geräten – Verfahren zur Zuverlässigkeitsbewertung (IEC 62308:2006); Deutsche Fassung EN 62308:2006*

DIN EN 81346-1:2010-05, *Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte – Strukturierungsprinzipien und Referenzkennzeichnung – Teil 1: Allgemeine Regeln (IEC 81346-1:2009); Deutsche Fassung EN 81346-1:2009*

DIN ISO/IEC 8652:1996-09, *Informationstechnik, Programmiersprachen – Ada (ISO/IEC 8652:1995)*

DIN EN ISO 9000, *Qualitätsmanagementsysteme – Grundlagen und Begriffe*

E DIN IEC 60068-1 (VDE 0468-1):2008-03, *Umgebungseinflüsse – Teil 1: Allgemeines und Leitfaden (IEC 104/445/CD:2007)*

E DIN IEC 60529/A1 (VDE 0470-1/A1):2010-04, *Schutzarten durch Gehäuse (IP-Code) – Vorschlag für IPX9 "Hochdruckwasserprüfung" (IEC 70/118/CD:2009)*

E DIN IEC 61131-3:2009-12, *Speicherprogrammierbare Steuerungen – Teil 3: Programmiersprachen; Englische Fassung (IEC 65B/725/CD:2009)*

EUROPÄISCHE NORM

EUROPEAN STANDARD

NORME EUROPÉENNE

EN 61508-7

Mai 2010

ICS 35.240.50

Ersatz für EN 61508-7:2001

Deutsche Fassung

Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme – Teil 7: Überblick über Verfahren und Maßnahmen

(IEC 61508-7:2010)

Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 7: Overview of techniques and measures (IEC 61508-7:2010)

Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 7: Présentation de techniques et mesures (CEI 61508-7:2010)

Diese Europäische Norm wurde von CENELEC am 2010-05-01 angenommen. Die CENELEC-Mitglieder sind gehalten, die CEN/CENELEC-Geschäftsordnung zu erfüllen, in der die Bedingungen festgelegt sind, unter denen dieser Europäischen Norm ohne jede Änderung der Status einer nationalen Norm zu geben ist.

Auf dem letzten Stand befindliche Listen dieser nationalen Normen mit ihren bibliographischen Angaben sind beim Zentralsekretariat oder bei jedem CENELEC-Mitglied auf Anfrage erhältlich.

Diese Europäische Norm besteht in drei offiziellen Fassungen (Deutsch, Englisch, Französisch). Eine Fassung in einer anderen Sprache, die von einem CENELEC-Mitglied in eigener Verantwortung durch Übersetzung in seine Landessprache gemacht und dem Zentralsekretariat mitgeteilt worden ist, hat den gleichen Status wie die offiziellen Fassungen.

CENELEC-Mitglieder sind die nationalen elektrotechnischen Komitees von Belgien, Bulgarien, Dänemark, Deutschland, Estland, Finnland, Frankreich, Griechenland, Irland, Island, Italien, Kroatien, Lettland, Litauen, Luxemburg, Malta, den Niederlanden, Norwegen, Österreich, Polen, Portugal, Rumänien, Schweden, der Schweiz, der Slowakei, Slowenien, Spanien, der Tschechischen Republik, Ungarn, dem Vereinigten Königreich und Zypern.

CENELEC

Europäisches Komitee für Elektrotechnische Normung
European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique

Zentralsekretariat: Avenue Marnix 17, B-1000 Brüssel

Ref. Nr. EN 61508-7:2010 D

Vorwort

Der Text des Schriftstücks 65A/554/FDIS, zukünftige 2. Ausgabe von IEC 61508-7, ausgearbeitet von dem SC 65A „System aspects“ des IEC/TC 65 „Industrial-process measurement, control and automation“, wurde der IEC-CENELEC Parallelen Abstimmung unterworfen und von CENELEC am 2010-05-01 als EN 61508-7 angenommen.

Diese Europäische Norm ersetzt EN 61508-7:2001.

Es wird auf die Möglichkeit hingewiesen, dass einige Elemente dieses Dokuments Patentrechte berühren können. CEN und CENELEC sind nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

Nachstehende Daten wurden festgelegt:

– spätestes Datum, zu dem die EN auf nationaler Ebene durch Veröffentlichung einer identischen nationalen Norm oder durch Anerkennung übernommen werden muss (dop): 2011-02-01

– spätestes Datum, zu dem nationale Normen, die der EN entgegenstehen, zurückgezogen werden müssen (dow): 2013-05-01

Der Anhang ZA wurde von CENELEC hinzugefügt.

_____

Anerkennungsnotiz

Der Text der Internationalen Norm IEC 61508-7:2010 wurde von CENELEC ohne irgendeine Abänderung als Europäische Norm angenommen.

In der offiziellen Fassung sind unter „Literaturhinweise“ zu den aufgelisteten Normen die nachstehenden Anmerkungen einzutragen:

[1] IEC 60068-1:1988	ANMERKUNG	Harmonisiert als EN 60068-1:1994 (nicht modifiziert).
[2] IEC 60529:1989	ANMERKUNG	Harmonisiert als EN 60529:1991 (nicht modifiziert).
[3] IEC 60812:2006	ANMERKUNG	Harmonisiert als EN 60812:2006 (nicht modifiziert).
[4] IEC 60880:2006	ANMERKUNG	Harmonisiert als EN 60880:2009 (nicht modifiziert).
[5] IEC 61000-4-1:2006	ANMERKUNG	Harmonisiert als EN 61000-4-1:2007 (nicht modifiziert).
[6] IEC 61000-4-5:2005	ANMERKUNG	Harmonisiert als EN 61000-4-5:2006 (nicht modifiziert).
[8] IEC 61025:2006	ANMERKUNG	Harmonisiert als EN 61025:2007 (nicht modifiziert).
[9] IEC 61069-5:1994	ANMERKUNG	Harmonisiert als EN 61069-5:1995 (nicht modifiziert).
[10] IEC 61078:2006	ANMERKUNG	Harmonisiert als EN 61078:2006 (nicht modifiziert).
[11] IEC 61131-3:2003	ANMERKUNG	Harmonisiert als EN 61131-3:2003 (nicht modifiziert).
[12] IEC 61160:2005	ANMERKUNG	Harmonisiert als EN 61160:2005 (nicht modifiziert).
[13] IEC 61163-1:2006	ANMERKUNG	Harmonisiert als EN 61163-1:2006 (nicht modifiziert).
[14] IEC 61164:2004	ANMERKUNG	Harmonisiert als EN 61164:2004 (nicht modifiziert).
[15] IEC 61165:2006	ANMERKUNG	Harmonisiert als EN 61165:2006 (nicht modifiziert).
[16] IEC 61326-3-1:2008	ANMERKUNG	Harmonisiert als EN 61326-3-1:2008 (nicht modifiziert).

[17] IEC 61326-3-2:2008	ANMERKUNG	Harmonisiert als EN 61326-3-2:2008 (nicht modifiziert).
[18] IEC 81346-1:2009	ANMERKUNG	Harmonisiert als EN 81346-1:2009 (nicht modifiziert).
[21] IEC 61511 Reihe	ANMERKUNG	Harmonized in der Reihe EN 61511 (nicht modifiziert).
[22] IEC 62061:2005	ANMERKUNG	Harmonisiert als EN 62061:2005 (nicht modifiziert).
[23] IEC 62308:2006	ANMERKUNG	Harmonisiert als EN 62308:2006 (nicht modifiziert).
[37] IEC 61800-5-2	ANMERKUNG	Harmonisiert als EN 61800-5-2.
[38] IEC 60601 Reihe	ANMERKUNG	Harmonized in der Reihe EN 60601 (teilweise modifiziert).
[39] IEC 60068-2-1	ANMERKUNG	Harmonisiert als EN 60068-2-1.
[40] IEC 60068-2-2	ANMERKUNG	Harmonisiert als EN 60068-2-2.
[41] ISO 9000	ANMERKUNG	Harmonisiert als EN ISO 9000.
[42] IEC 61508-1:2010	ANMERKUNG	Harmonisiert als EN 61508-1:2010 (nicht modifiziert).
[43] IEC 61508-2:2010	ANMERKUNG	Harmonisiert als EN 61508-2:2010 (nicht modifiziert).
[44] IEC 61508-3:2010	ANMERKUNG	Harmonisiert als EN 61508-3:2010 (nicht modifiziert).
[45] IEC 61508-6:2010	ANMERKUNG	Harmonisiert als EN 61508-6:2010 (nicht modifiziert).

Inhalt

Seite

Vorwort ... 2
Einleitung ... 12
1 Anwendungsbereich ... 14
2 Normative Verweisungen ... 16
3 Begriffe und Abkürzungen ... 16
Anhang A (informativ) Überblick über Verfahren und Maßnahmen für sicherheitsbezogene E/E/PE-Systeme: Beherrschung von zufälligen Hardwareausfällen (siehe IEC 61508-2) ... 17
A.1 Elektrik ... 17
A.1.1 Erkennung von Ausfällen durch Überwachung während des Betriebs ... 17
A.1.2 Überwachung von Relaiskontakten ... 17
A.1.3 Vergleicher ... 17
A.1.4 Mehrheitsentscheider ... 18
A.1.5 Ruhestromprinzip ... 18
A.2 Elektronik ... 18
A.2.1 Tests durch redundante Hardware ... 18
A.2.2 Dynamische Prinzipien ... 18
A.2.3 Standardtestschnittstelle (Access Port) und Boundary-Scan-Architektur ... 19
A.2.4 (Nicht verwendet) ... 19
A.2.5 Überwachte Redundanz ... 19
A.2.6 Hardware mit automatischen Tests ... 19
A.2.7 Analogsignal-Überwachung ... 20
A.2.8 Unterlastung ... 20
A.3 Verarbeitungseinheiten (CPUs) ... 20
A.3.1 Selbsttest durch Software: begrenzte Anzahl von Mustern (ein Kanal) ... 20
A.3.2 Selbsttest durch Software: Walking Bit (ein Kanal) ... 20
A.3.3 Selbsttest unterstützt durch Hardware (ein Kanal) ... 21
A.3.4 Codierte Verarbeitung (ein Kanal) ... 21
A.3.5 Gegenseitiger Vergleich durch Software ... 21
A.4 Unveränderliche Speicherbereiche ... 21
A.4.1 Wortsicherungsverfahren mit Mehr-Bit-Redundanz (zum Beispiel ROM-Überwachung mit einem modifizierten Hammingcode) ... 21
A.4.2 Modifizierte Prüfsumme ... 22
A.4.3 Signatur mit einfacher Wortbreite (8 Bit) ... 22
A.4.4 Signatur mit doppelter Wortbreite (16 Bit) ... 22
A.4.5 Blockwiederholung (zum Beispiel doppeltes ROM mit Hardware- oder Softwarevergleich) ... 23
A.5 Veränderliche Speicherbereiche ... 23
A.5.1 RAM-Test „Checkerboard“ oder „March“ ... 24

Seite

A.5.2 RAM-Test „Walkpath“ ... 24
A.5.3 RAM-Test „Galpat“ oder „transparenter Galpat“ ... 24
A.5.4 RAM-Test „Abraham“ ... 25
A.5.5 Ein-Bit-Redundanz (zum Beispiel RAM-Überwachung mit einem Parity-Bit) ... 25
A.5.6 RAM-Überwachung mit einem modifizierten Hammingcode oder Erkennung von Datenfehlern mit fehlererkennenden und -korrigierenden Codes (en: error-detection-correction codes, EDC) ... 25
A.5.7 Doppeltes RAM mit Hardware- oder Softwarevergleich und Schreib-/Lesetest ... 26
A.6 E/A-Einheiten und Schnittstellen (externe Kommunikation) ... 26
A.6.1 Testmuster ... 26
A.6.2 Codesicherung ... 26
A.6.3 Mehrkanalige parallele Ausgabe ... 27
A.6.4 Überwachte Ausgaben ... 27
A.6.5 Eingabevergleich/-entscheidung ... 27
A.7 Datenwege (interne Kommunikation) ... 27
A.7.1 Ein-Bit-Hardwareredundanz ... 27
A.7.2 Mehr-Bit-Hardwareredundanz ... 27
A.7.3 Vollständige Hardwareredundanz ... 28
A.7.4 Inspektion durch Verwendung von Testmustern ... 28
A.7.5 Übertragungsredundanz ... 28
A.7.6 Informationsredundanz ... 28
A.8 Spannungsversorgung ... 28
A.8.1 Überspannungsschutz mit Sicherheitsabschaltung ... 28
A.8.2 Spannungsüberwachung (sekundärseitig) ... 29
A.8.3 Energieabschaltung mit Sicherheitsabschaltung ... 29
A.9 Zeitliche und logische Programmlaufüberwachung ... 29
A.9.1 Watchdog mit separater Zeitbasis ohne Zeitfenster ... 29
A.9.2 Watchdog mit separater Zeitbasis und Zeitfenster ... 29
A.9.3 Logische Überwachung des Programmablaufs ... 30
A.9.4 Kombination von zeitlicher und logischer Überwachung des Programmablaufs ... 30
A.9.5 Zeitliche Überwachung mit Test während des Betriebs ... 30
A.10 Lüftung und Beheizung ... 30
A.10.1 Temperatursensor ... 30
A.10.2 Lüfterkontrolle ... 30
A.10.3 Auslösung der Sicherheitsabschaltung über eine thermische Sicherung ... 31
A.10.4 Gestaffelte Meldung von Thermosensoren und bedingter Alarm ... 31
A.10.5 Zuschaltung von Umluftkühlung und Statusanzeige ... 31
A.11 Kommunikation und Massenspeicher ... 31
A.11.1 Trennung elektrischer Energieleitungen von Informationsleitungen ... 31
A.11.2 Räumliche Trennung mehrfacher Leitungen ... 31

Seite

A.11.3 Erhöhung der Störfestigkeit ... 31
A.11.4 Antivalente Signalübertragung ... 32
A.12 Sensoren ... 32
A.12.1 Referenzsensor ... 32
A.12.2 Zwangsöffnender Schalter ... 33
A.13 Stellglieder (Aktoren) ... 33
A.13.1 Überwachung ... 33
A.13.2 Kreuzweise Überwachung mehrfacher Aktoren ... 33
A.14 Maßnahmen gegen die Einwirkung der physikalischen Umgebung ... 33
Anhang B (informativ) Überblick über Verfahren und Maßnahmen für sicherheitsbezogene E/E/PE-Systeme: Vermeidung von systematischen Ausfällen (siehe IEC 61508-2 und IEC 61508-3) ... 35
B.1 Allgemeine Maßnahmen und Verfahren ... 35
B.1.1 Projektmanagement ... 35
B.1.2 Dokumentation ... 36
B.1.3 Trennung der Sicherheitsfunktionen des E/E/PE-Systems von Nichtsicherheitsfunktionen ... 37
B.1.4 Diversitäre Hardware ... 37
B.2 Spezifikation der Anforderungen an den Entwurf des E/E/PE-Systems ... 37
B.2.1 Strukturierte Spezifikation ... 37
B.2.2 Formale Methoden ... 38
B.2.3 Semi-formale Methoden ... 39
B.2.4 Rechnerunterstützte Spezifikationswerkzeuge ... 41
B.2.5 Checklisten ... 42
B.2.6 Inspektion der Spezifikation ... 43
B.3 Entwurf und Entwicklung des E/E/PE-Systems ... 43
B.3.1 Beachtung von Richtlinien und Normen ... 44
B.3.2 Strukturierter Entwurf ... 44
B.3.3 Verwendung von bewährten Bauteilen ... 44
B.3.4 Modularisierung ... 45
B.3.5 Rechnerunterstützte Entwurfswerkzeuge ... 45
B.3.6 Simulation ... 46
B.3.7 Inspektion (Überprüfungen und Analysen) ... 46
B.3.8 Walkthrough ... 47
B.4 Betriebs- und Instandhaltungsverfahren für das E/E/PE-System ... 47
B.4.1 Betriebs- und Instandhaltungsanweisungen ... 47
B.4.2 Benutzerfreundlichkeit ... 47
B.4.3 Instandhaltungsfreundlichkeit ... 48
B.4.4 Eingeschränkte Betriebsmöglichkeiten ... 48
B.4.5 Betrieb nur durch erfahrene Bediener ... 48
B.4.6 Schutz gegen Irrtümer des Bedieners ... 49

Seite

B.4.7 (Nicht verwendet) ... 49
B.4.8 Schutz vor Modifikation ... 49
B.4.9 Eingabebestätigung ... 49
B.5 Integration des E/E/PE-Systems ... 50
B.5.1 Funktionstest ... 50
B.5.2 Black-Box-Test ... 50
B.5.3 Statistisches Testen ... 51
B.5.4 Felderfahrung ... 51
B.6 Validierung der Sicherheit des E/E/PE-Systems ... 52
B.6.1 Funktionstest unter Umgebungsbedingungen ... 52
B.6.2 Test der Störfestigkeit gegen Stoßspannungen ... 53
B.6.3 (Nicht verwendet) ... 53
B.6.4 Statische Analyse ... 53
B.6.5 Dynamische Analyse und Test ... 54
B.6.6 Ausfallanalyse ... 54
B.6.7 Worst-Case-Analyse ... 60
B.6.8 Erweiterte Funktionstests ... 61
B.6.9 Test unter Grenzbedingungen ... 61
B.6.10 Test durch Fehlereinbau ... 61
Anhang C (informativ) Überblick über Verfahren und Maßnahmen zum Erreichen der Sicherheitsintegrität der Software (siehe IEC 61508-3) ... 63
C.1 Allgemeines ... 63
C.2 Anforderungen und detaillierter Entwurf ... 63
C.2.1 Strukturierte schematische Methoden ... 63
C.2.2 Datenflussdiagramme ... 65
C.2.3 Strukturdiagramme ... 66
C.2.4 Formale Methoden ... 67
C.2.5 Defensive Programmierung ... 71
C.2.6 Entwurfs- und Programmierrichtlinien ... 72
C.2.7 Strukturierte Programmierung ... 77
C.2.8 Geheimnisprinzip/Kapselung ... 77
C.2.9 Modularer Ansatz ... 78
C.2.10 Verwendung bewährter/verifizierter Softwareelemente ... 79
C.2.11 Nachvollziehbarkeit ... 81
C.2.12 Zustandsloser Softwareentwurf (oder Entwurf eingeschränkter Zustände) ... 82
C.2.13 Numerische Offline-Analyse ... 83
C.2.14 Nachrichtenverlaufstabellen ... 83
C.3 Entwurf der Architektur ... 83
C.3.1 Fehlererkennung und Diagnose ... 83

Seite

C.3.2 Fehlererkennende und korrigierende Codes ... 84
C.3.3 Assertion-Programmierung (en: failure assertion programming) ... 85
C.3.4 Diversitäre Überwachungseinrichtung ... 85
C.3.5 Diversitäre Programmierung ... 86
C.3.6 Rückwärtsregeneration ... 86
C.3.7 Regeneration durch Wiederholung ... 87
C.3.8 Abgestufte Funktionseinschränkungen ... 87
C.3.9 Künstliche Intelligenz – Fehlerkorrektur ... 88
C.3.10 Dynamische Rekonfiguration ... 88
C.3.11 Sicherheit und Leistungsfähigkeit in Echtzeit: zeitgesteuerte Architektur ... 89
C.3.12 UML ... 90
C.4 Entwicklungswerkzeuge und Programmiersprachen ... 91
C.4.1 Streng typisierte Programmiersprache ... 91
C.4.2 Sprachenteilmenge ... 92
C.4.3 Zertifizierte Werkzeuge und Compiler ... 92
C.4.4 Betriebsbewährte Werkzeuge und Compiler ... 92
C.4.5 Geeignete Programmiersprache ... 93
C.4.7 Testmanagement und Automatisierungswerkzeuge ... 97
C.5 Verifikation und Modifikation ... 97
C.5.1 Statistisches Testen ... 97
C.5.2 Datenaufzeichnung und Analyse ... 98
C.5.3 Schnittstellenprüfung ... 98
C.5.4 Durchführung von Testfällen nach einer Grenzwertanalyse ... 99
C.5.5 Durchführung von Testfällen aus der Fehlererwartung („Fehler erraten“) ... 99
C.5.6 Durchführung von Testfällen nach Fehlereinpflanzung ... 100
C.5.7 Äquivalenzklassen und Test von Partitionen des Eingangsbereichs ... 100
C.5.8 Strukturabhängige Tests ... 101
C.5.9 Kontrollflussanalyse ... 102
C.5.10 Datenflussanalyse ... 102
C.5.11 Symbolische Ausführung ... 103
C.5.12 Formaler Beweis (Verifikation) ... 103
C.5.13 Softwarekomplexitätskontrolle ... 105
C.5.14 Formale Inspektion ... 105
C.5.15 Walk-through (Software) ... 106
C.5.16 Entwurfsüberprüfung ... 106
C.5.17 Prototypenerstellung/Animation ... 107
C.5.18 Simulation des Prozesses ... 107
C.5.19 Anforderungen an die Leistungsfähigkeit ... 108
C.5.20 Modellierung der Leistungsfähigkeit ... 108

Seite

C.5.21 Belastungstest (en: avalanche/stress testing) ... 109
C.5.22 Reaktionszeiten und Speicherbeschränkungen ... 110
C.5.23 Einflussanalyse ... 110
C.5.24 Software-Konfigurationsmanagement ... 110
C.5.25 Validierung durch Regressionstest ... 111
C.5.26 Animation der Spezifikation und des Entwurfs ... 111
C.5.27 Modellbasiertes Testen (Testfallgenerierung) ... 112
C.6 Beurteilung der funktionalen Sicherheit ... 113
C.6.1 Entscheidungs-/Wahrheitstabellen ... 114
C.6.2 Software-Gefährdungs- und Betreibbarkeitsuntersuchung (en: software hazard and operability study, CHAZOP, FMEA) ... 114
C.6.3 Analyse von Ausfällen infolge gemeinsamer Ursache ... 114
C.6.4 Zuverlässigkeitsblockdiagramm ... 115
Anhang D (informativ) Ein probabilistischer Ansatz zur Bestimmung der Sicherheitsintegrität von vorentwickelter Software ... 117
D.1 Allgemeines ... 117
D.2 Gleichungen für statistische Tests und Beispiele für ihre Anwendung ... 118
D.2.1 Einfacher statistischer Test für die Betriebsart mit niedriger Anforderungsrate ... 118
D.2.2 Test des Wertebereichs der Eingänge für die Betriebsart mit niedriger Anforderungsrate ... 118
D.2.3 Einfacher statistischer Test für die Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung ... 119
D.2.4 Vollständiger Test ... 120
D.3 Literaturhinweise ... 121
Anhang E (informativ) Überblick über Verfahren und Maßnahmen für den Entwurf von ASICs ... 122
E.1 Entwurfsbeschreibung in (V)HDL ... 122
E.2 Schaltplaneingabe ... 122
E.3 Strukturierte Beschreibungsmethodik ... 122
E.4 Betriebsbewährte Werkzeuge ... 123
E.5 (V)HDL-Simulation ... 123
E.6 Funktionstest auf Modulebene ... 123
E.7 Funktionstest auf oberster Ebene ... 124
E.8 Funktionstest eingebettet in Systemumgebung ... 124
E.9 Eingeschränkte Verwendung asynchroner Konstrukte ... 124
E.10 Synchronisation von primären Eingängen und Kontrolle von Metastabilitäten ... 124
E.11 Entwurf für Testbarkeit ... 125
E.12 Modularisierung ... 125
E.13 Testabdeckung der Verifikationsszenarien (Testbenches) ... 125
E.14 Beachtung von Programmierrichtlinien ... 126
E.15 Verwendung eines Codecheckers ... 127
E.16 Defensive Programmierung ... 127

Seite

E.17 Dokumentation von Simulationsergebnissen ... 127
E.18 Codeinspektion ... 128
E.19 Walkthrough ... 128
E.20 Anwendung validierter Soft-Cores ... 128
E.21 Validierung von Soft-Cores ... 129
E.22 Simulation der Gatter-Netzliste zur Überprüfung der Zeitvorgaben ... 129
E.23 Statische Laufzeitanalyse (STA) ... 129
E.24 Vergleich der Gatter-Netzliste gegen ein Referenzmodell durch Simulation ... 130
E.25 Vergleich der Gatter-Netzliste mit dem Referenzmodell (formale Äquivalenzprüfung) ... 130
E.26 Überprüfung der Anforderungen und Beschränkungen des Herstellers ... 130
E.27 Dokumentation der Synthesevorgaben, Ergebnisse und Werkzeuge ... 131
E.28 Verwendung von betriebsbewährten Synthesewerkzeugen ... 131
E.29 Verwendung von betriebsbewährten Zielbibliotheken ... 131
E.30 Skriptbasierende Verfahren ... 131
E.31 Implementierung von Teststrukturen ... 132
E.32 Abschätzung der Testabdeckung durch Simulation ... 132
E.33 Abschätzung der Testabdeckung durch Anwendung eines ATPG-Werkzeugs ... 133
E.34 Nachweis der Betriebsbewährung für verwendete Hard-Cores ... 133
E.35 Verwendung validierter Hard-Cores ... 133
E.36 Online-Tests der Hard-Cores ... 133
E.37 Design-Rule-Check (DRC) ... 133
E.38 Überprüfung des Layouts Versus Schematic (LVS) ... 134
E.39 Zusätzliche Reserven (> 20 %) für Prozesstechnologien mit weniger als 3 Jahren Anwendung ... 134
E.40 Burn-In-Test ... 134
E.41 Verwendung von betriebsbewährten Schaltkreisfamilien ... 134
E.42 Betriebsbewährter Fertigungsprozess ... 134
E.43 Qualitätskontrolle des Fertigungsprozesses ... 135
E.44 Produktions-Qualitätsprüfung des Schaltkreises ... 135
E.45 Funktionale Qualitätsprüfung des Schaltkreises ... 135
E.46 Qualitätsnormen ... 135
Anhang F (informativ) Definitionen der Eigenschaften der Software-Lebenszyklusphasen ... 136
Anhang G (informativ) Anleitung zur Entwicklung von sicherheitsbezogener objektorientierter Software ... 143
Stichwortverzeichnis ... 145
Literaturhinweise ... 153
Anhang ZA (normativ) Normative Verweisungen auf internationale Publikationen mit ihren entsprechenden europäischen Publikationen ... 156

Bilder

Bild 1 – Gesamtrahmen der IEC 61508 ... 15

Seite

Tabellen

Tabelle C.1 – Empfehlungen für bestimmte Programmiersprachen 95
Tabelle D.1 – Notwendige Vorgeschichte zur Zuordnung von Sicherheits-Integritätsleveln bei gegebenem Vertrauensniveau 117
Tabelle D.2 – Wahrscheinlichkeiten eines Versagens für die Betriebsart mit niedriger Anforderungsrate 118
Tabelle D.3 – Mittlerer Abstand von zwei Testpunkten 119
Tabelle D.4 – Wahrscheinlichkeit eines Versagens für die Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung 120
Tabelle D.5 – Wahrscheinlichkeit des Tests aller Programmeigenschaften 121
Tabelle F.1 – Spezifikation der Anforderungen an die Sicherheit der Software 136
Tabelle F.2 – Softwareentwurf und Softwareentwicklung: Entwurf der Softwarearchitektur 137
Tabelle F.3 – Softwareentwurf und Softwareentwicklung: unterstützende Werkzeuge und Programmiersprache 138
Tabelle F.4 – Softwareentwurf und Softwareentwicklung: detaillierter Entwurf 138
Tabelle F.5 – Softwareentwurf und Softwareentwicklung: Test der Softwaremodule und Integration 139
Tabelle F.6 – Integration der programmierbaren Elektronik (Hardware und Software) 139
Tabelle F.7 – Softwareaspekte bezüglich der Validierung der Systemsicherheit 140
Tabelle F.8 – Softwaremodifikation 140
Tabelle F.9 – Softwareverifikation 141
Tabelle F.10 – Beurteilung der funktionalen Sicherheit 141
Tabelle G.1 – Objektorientierte Softwarearchitektur 143
Tabelle G.2 – Detaillierter objektorientierter Entwurf 144
Tabelle G.3 – Einige detaillierte objektorientierte Begriffe 144

Einleitung

Systeme, die aus elektrischen und/oder elektronischen Elementen bestehen, werden seit vielen Jahren verwendet, um Sicherheitsfunktionen in den meisten Anwendungsbereichen auszuführen. Auf Rechnern basierende Systeme (allgemein ausgedrückt programmierbare elektronische Systeme) werden in allen Anwendungsbereichen benutzt, um Nichtsicherheitsfunktionen und zunehmend auch um Sicherheitsfunktionen auszuführen. Falls Rechnersystemtechnologie wirksam und sicherheitsgerichtet eingesetzt wird, ist es wichtig, dass die für die Entscheidungsfindung Verantwortlichen ausreichende Hilfestellung bezüglich der Sicherheitsaspekte erhalten, nach denen diese Entscheidungen getroffen werden.

Diese Internationale Norm beschreibt einen allgemeinen Lösungsweg für alle Tätigkeiten während des Sicherheitslebenszyklus für Systeme, die aus elektrischen und/oder elektronischen und/oder programmierbaren elektronischen (E/E/PE) Elementen bestehen und die eingesetzt werden, um Sicherheitsfunktionen auszuführen. Dieser allgemeine Lösungsweg wurde gewählt, um ein sinnvolles und konsistentes technisches Verfahren für alle elektrischen Sicherheitssysteme zu entwickeln. Ein Hauptziel ist es, die Entwicklung von produkt- und anwendungsspezifischen internationalen Normen, die auf der Normenreihe IEC 61508 basieren, zu erleichtern.

ANMERKUNG 1 In den Literaturhinweisen sind Beispiele von produkt- und anwendungsspezifischen internationalen Normen, die auf der Normenreihe IEC 61508 basieren, enthalten (siehe Hinweise [21], [22] und [37]).

In den meisten Situationen wird Sicherheit durch eine Anzahl von Systemen erreicht, die auf vielerlei Technologien (zum Beispiel Mechanik, Hydraulik, Pneumatik, Elektrik, Elektronik, programmierbare Elektronik) basieren. Jede Sicherheitsstrategie muss deshalb nicht nur alle Elemente innerhalb eines Einzelsystems (zum Beispiel Sensoren, Steuereinheiten und Aktoren) betrachten, sondern auch all die sicherheitsbezogenen Systeme, welche die Gesamtheit von sicherheitsbezogenen Systemen bilden. Daher kann diese Internationale Norm, obwohl sie sich mit sicherheitsbezogenen E/E/PE-Systemen beschäftigt, auch einen Rahmen bereitstellen, innerhalb dessen sicherheitsbezogene Systeme basierend auf anderen Technologien betrachtet werden können.

Es ist beachtet worden, dass eine große Vielfalt von Anwendungen in vielfältigen Anwendungsbereichen vorliegt, die sicherheitsbezogene E/E/PE-Systeme verwenden, und diese einen weiten Bereich in Bezug auf Komplexität, Gefährdungs- und Risikopotentiale abdecken. In jeder speziellen Anwendung sind die erforderlichen Sicherheitsmaßnahmen von vielen anwendungsspezifischen Faktoren abhängig. Dadurch, dass diese Internationale Norm allgemein gehalten ist, wird die Formulierung solcher Maßnahmen in zukünftigen produkt- und anwendungsspezifischen internationalen Normen und in Überarbeitungen der bereits bestehenden Normen ermöglicht.

Diese Internationale Norm:

– betrachtet alle relevanten Phasen des Gesamt-Sicherheitslebenszyklus, des Sicherheitslebenszyklus des E/E/PE-Systems und des Software-Sicherheitslebenszyklus (zum Beispiel vom anfänglichen Konzept über Entwurf, Implementierung, Betrieb und Instandhaltung bis zur Außerbetriebnahme), wenn E/E/PE-Systeme benutzt werden, um Sicherheitsfunktionen auszuführen;
– wurde unter Berücksichtigung einer sich schnell entwickelnden Technologie entworfen. Der Betrachtungsrahmen ist ausreichend robust und ausführlich genug, um auch für zukünftige Entwicklungen verwendbar zu sein;
– ermöglicht die Erstellung produkt- und anwendungsspezifischer Internationaler Normen, die sich mit sicherheitsbezogenen E/E/PE-Systemen befassen. Die Entwicklung produkt- und anwendungsspezifischer Internationaler Normen sollte innerhalb des Rahmens dieser Norm zu einem hohen Grad an Übereinstimmung (zum Beispiel von zugrunde liegenden Prinzipien, Terminologie usw.) führen sowohl innerhalb der Anwendungsbereiche als auch über die Anwendungsbereiche hinweg. Dies hat sowohl sicherheitstechnische als auch wirtschaftliche Vorteile;
– liefert eine Methode für die Entwicklung der Spezifikation der Anforderungen an die Sicherheit, die notwendig ist, um die erforderliche funktionale Sicherheit für sicherheitsbezogene E/E/PE-Systeme zu erreichen;
– verwendet einen auf dem Risiko basierenden Lösungsansatz, durch den die Anforderungen an die Sicherheitsintegrität bestimmt werden können;

– führt Sicherheits-Integritätslevel für die Festlegung der Zielvorgabe der Sicherheitsintegrität der Sicherheitsfunktionen ein, die von den sicherheitsbezogenen E/E/PE-Systemen zu implementieren sind;

ANMERKUNG 2 Diese Norm legt weder die Anforderungen an den Sicherheits-Integritätslevel für irgendeine Sicherheitsfunktion fest, noch bestimmt sie, wie der Sicherheits-Integritätslevel festgelegt wird. Stattdessen stellt sie einen risikobasierenden konzeptionellen Rahmen und Beispielverfahren bereit.

– legt Ausfallgrenzwerte für die von den sicherheitsbezogenen E/E/PE-Systemen auszuführenden Sicherheitsfunktionen fest, die mit den Sicherheits-Integritätsleveln verbunden sind;
– legt eine untere Grenze für die Ausfallgrenzwerte für eine Sicherheitsfunktion fest, die von einem einzelnen sicherheitsbezogenen E/E/PE-System ausgeführt wird. Für sicherheitsbezogene E/E/PE-Systeme, die:
  – in der Betriebsart mit einer niedrigen Anforderungsrate betrieben werden, ist die untere Grenze bei einer mittleren Wahrscheinlichkeit eines gefahrbringenden Ausfalls bei Anforderung von 10^{-5} festgelegt;
  – in der Betriebsart mit einer hohen Anforderungsrate oder mit kontinuierlicher Anforderung betrieben werden, ist die untere Grenze bei einer mittleren Häufigkeit eines gefahrbringenden Ausfalls von 10^{-9} [h^{-1}] festgelegt;

ANMERKUNG 3 Ein einzelnes sicherheitsbezogenes E/E/PE-System bedeutet nicht notwendigerweise eine einkanalige Architektur.

ANMERKUNG 4 Es kann für einfache Systeme möglich sein, Entwürfe von sicherheitsbezogenen Systemen mit niedrigeren Zielwerten für die Sicherheitsintegrität zu erreichen, aber diese Grenzen werden als das betrachtet, was für relativ komplexe Systeme (zum Beispiel sicherheitsbezogene programmierbare elektronische Systeme) gegenwärtig erreicht werden kann.

– legt Anforderungen für die Vermeidung und Beherrschung von systematischen Fehlern fest, die auf Erfahrungen und Urteilsvermögen beruhen, die durch praktische Erfahrung in der Industrie gewonnen wurden. Wenn auch die Wahrscheinlichkeit des Auftretens systematischer Ausfälle im Allgemeinen nicht quantifiziert werden kann, erlaubt die Norm jedoch für eine festgelegte Sicherheitsfunktion den Anspruch zu erheben, dass der mit der Sicherheitsfunktion verbundene Ausfallgrenzwert als erreicht betrachtet werden kann, wenn alle Anforderungen dieser Norm erfüllt worden sind;
– führt die systematische Eignung ein, die für ein Element im Hinblick auf das Vertrauen gilt, dass die systematische Sicherheitsintegrität die Anforderungen des festgelegten Sicherheits-Integritätslevels erfüllt;
– lässt einen weiten Bereich von Prinzipien, Verfahren und Maßnahmen zu, um funktionale Sicherheit für sicherheitsbezogene E/E/PE-Systeme zu erreichen, verwendet aber nicht ausdrücklich das Fail-Safe-Konzept. „Fail-Safe“-Konzepte und „inhärent sichere“ Prinzipien können jedoch anwendbar sein, und der Einsatz solcher Konzepte ist zulässig, vorausgesetzt, dass die Anforderungen der zutreffenden Abschnitte in der Norm erfüllt werden.

Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme – Teil 7: Überblick über Verfahren und Maßnahmen

1 Anwendungsbereich

1.1 Dieser Teil der IEC 61508 beinhaltet einen Überblick über verschiedene Sicherheitsverfahren und -maßnahmen zur Anwendung der IEC 61508-2 und IEC 61508-3.

Die Literaturhinweise sollten als grundlegende Verweise auf Methoden und Werkzeuge oder als Beispiele betrachtet werden und können möglicherweise nicht den Stand der Technik wiedergeben.

1.2 IEC 61508-1, IEC 61598-2, IEC 61508-3 und IEC 61508-4 sind Sicherheitsgrundnormen, dieser Status ist aber im Zusammenhang mit einfachen sicherheitsbezogenen E/E/PE-Systemen nicht anwendbar (siehe 3.4.3 der IEC 61508-4). Als Sicherheitsgrundnormen sind sie zur Verwendung durch technische Komitees bei der Erstellung von Normen nach IEC Guide 104 und ISO/IEC Guide 51 vorgesehen. IEC 61508-1, IEC 61508-2, IEC 61508-3 und IEC 61508-4 sind ebenfalls zur Verwendung als eigenständige Norm vorgesehen. Die horizontale Sicherheitsfunktion dieser internationalen Norm trifft nicht auf medizinische Einrichtungen gemäß der Normenreihe IEC 60601 zu.

1.3 Es steht in der Verantwortlichkeit eines Technischen Komitees, zur Vorbereitung und Erstellung eigener Festlegungen soweit möglich die Sicherheitsgrundnormen anzuwenden. In diesem Zusammenhang gilt, dass die Anforderungen, Prüfverfahren oder Prüfbedingungen dieser Sicherheitsgrundnorm nur dann anwendbar sind, wenn in den Festlegungen der Technischen Komitees darauf verwiesen wird oder diese eingebunden werden.

1.4 Bild 1 zeigt den gesamten Rahmen für die Teile 1 bis 7 der IEC 61508 und zeigt die Rolle, die IEC 61508-7 zum Erreichen der funktionalen Sicherheit der sicherheitsbezogenen E/E/PE-Systeme spielt.

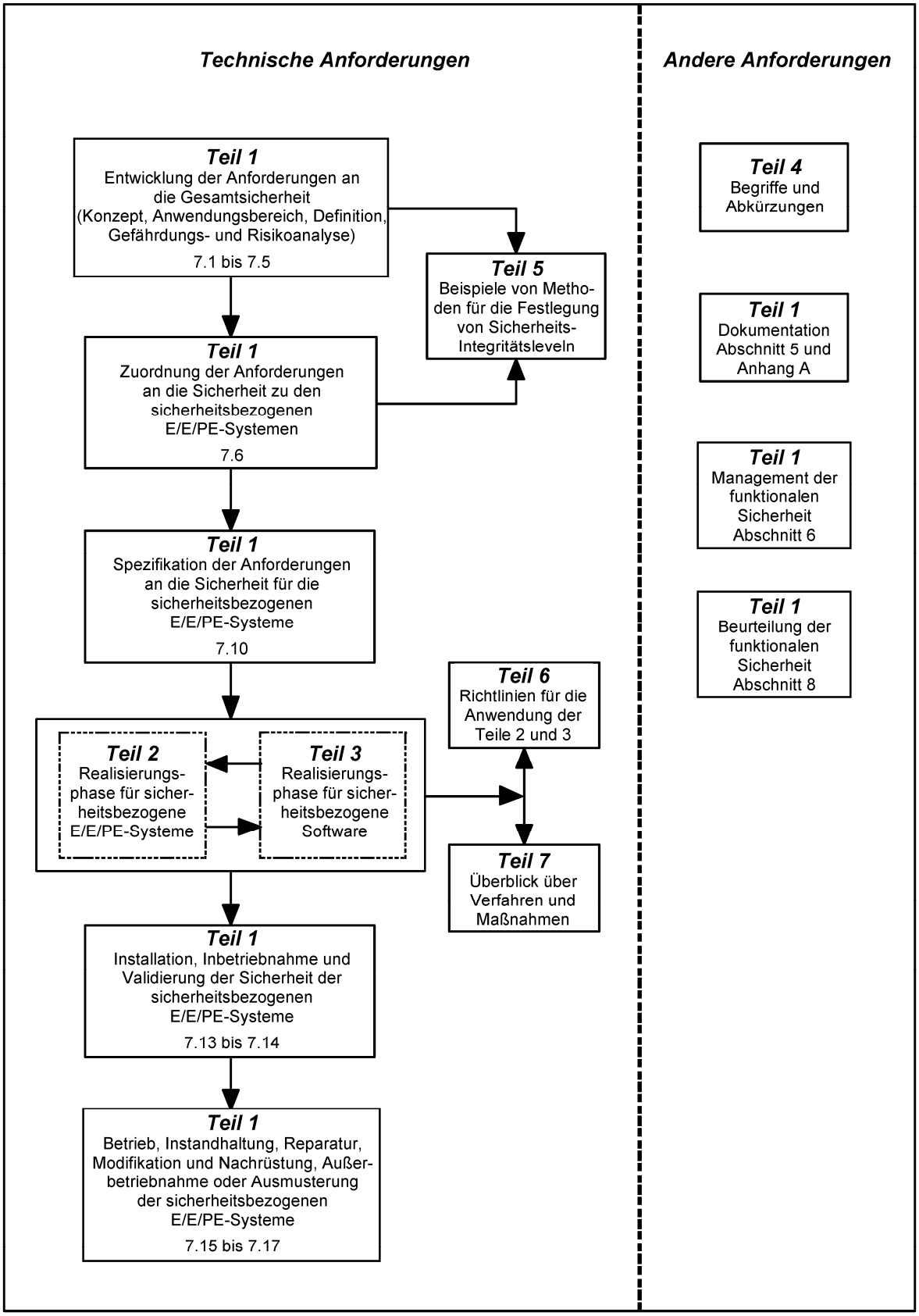


Bild 1 – Gesamtrahmen der IEC 61508

2 Normative Verweisungen

Die folgenden zitierten Dokumente sind für die Anwendung dieses Dokuments erforderlich. Bei datierten Verweisungen gilt nur die in Bezug genommene Ausgabe. Bei undatierten Verweisungen gilt die letzte Ausgabe des in Bezug genommenen Dokuments (einschließlich aller Änderungen).

IEC 61508-4:2010, *Functional safety of electrical/electronical/programmable electronic safety-related systems – Part 4: Definitions and abbreviations*

3 Begriffe und Abkürzungen

Für die Anwendung dieses Dokumentes gelten die in der IEC 61508-4 aufgeführten Begriffe und Abkürzungen.

Anhang A
(informativ)

Überblick über Verfahren und Maßnahmen für sicherheitsbezogene E/E/PE-Systeme: Beherrschung von zufälligen Hardwareausfällen (siehe IEC 61508-2)

A.1 Elektrik

Allgemeines Ziel: Beherrschung von Ausfällen in elektromechanischen Bauteilen.

A.1.1 Erkennung von Ausfällen durch Überwachung während des Betriebs

ANMERKUNG Diese(s) Verfahren/Maßnahme ist in den Tabellen A.2, A.3, A.7 und A.13 bis A.18 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen durch Überwachung des Verhaltens des sicherheitsbezogenen E/E/PE-Systems in Bezug auf den normalen (Online) Betrieb der EUC-Einrichtung.

Beschreibung: Unter bestimmten Bedingungen können Ausfälle durch (zum Beispiel) die Verwendung von Informationen über das Zeitverhalten der EUC-Einrichtung erkannt werden. Wenn zum Beispiel ein Schalter, der Teil eines sicherheitsbezogenen E/E/PE-Systems ist und normalerweise durch die EUC-Einrichtung betätigt wird, seinen Zustand nicht zur erwarteten Zeit ändert, wird ein Ausfall erkannt. Es ist gewöhnlich nicht möglich, den Ausfall zu lokalisieren.

A.1.2 Überwachung von Relaiskontakten

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.2 und A.14 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen von Relaiskontakten (zum Beispiel Verschweißen).

Beschreibung: Zwangsgeführte Relais sind derart entworfen, dass ihre Kontakte fest miteinander verbunden sind. Wenn, ausgehend von zwei Sätzen von Wechselkontakten *a* und *b*, der Schließerkontakt *a* verschweißt, kann der Öffnerkontakt *b* bei der nächsten Entregung der Relaisspule nicht schließen. Daher kann die Überwachung des Schließens des Öffnerkontakts *b* bei Entregung der Relaisspule verwendet werden, um zu prüfen, ob der Schließerkontakt *a* geöffnet hat. Das Nichtschließen des Öffnerkontakts *b* zeigt einen Ausfall von Kontakt *a* an, so dass der Überwachungskreis eine sichere Abschaltung und ggf. eine Fortsetzung des sicheren Abschaltens für jede Maschine, die durch Kontakt *a* angesteuert wird, gewährleistet.

Literaturhinweis:

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Verriegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld.
www.BGIA-HANDBUCHdigital.de/330212

A.1.3 Vergleicher

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.2, A.3 und A.4 der IEC 61508-2 angeführt.

Ziel: So früh wie möglich Erkennung von (nicht-gleichzeitigen) Ausfällen in einer unabhängigen Verarbeitungseinheit oder im Vergleicher.

Beschreibung: Die Signale unabhängiger Verarbeitungseinheiten werden zyklisch oder kontinuierlich von einem Hardwarevergleicher verglichen. Der Vergleicher selbst kann extern getestet werden oder selbstüber-

wachende Technologie verwenden. Erkannte Unterschiede im Verhalten der Prozessoren führen zu einer Ausfallmeldung.

A.1.4 Mehrheitsentscheider

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.2, A.3 und A.4 der IEC 61508-2 angeführt.

Ziel: Erkennung und Maskierung von Ausfällen in einem von mindestens drei Hardwarekanälen.

Beschreibung: Eine Entscheidungseinheit nach dem Mehrheitsprinzip (2-aus-3, 3-aus-3 oder *m*-aus-*n, moon*) wird zur Erkennung und Maskierung von Ausfällen verwendet. Die Einheit selbst kann extern getestet werden oder selbstüberwachende Technologie verwenden.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.1.5 Ruhestromprinzip

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.16 der IEC 61508-2 angeführt.

Ziel: Ausführung der Sicherheitsfunktion bei Energieunterbrechung oder -verlust.

Beschreibung: Die Sicherheitsfunktion wird ausgeführt, wenn die Kontakte offen sind und kein Strom fließt. Werden zum Beispiel Bremsen zum Anhalten einer gefahrbringenden Bewegung eines Motors verwendet, dann werden die Bremsen durch das Schließen von Kontakten in dem sicherheitsbezogenen System gelöst und durch das Öffnen der Kontakte im sicherheitsbezogenen System betätigt.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.2 Elektronik

Allgemeines Ziel: Beherrschung von Ausfällen in Halbleiter-Bauelementen.

A.2.1 Tests durch redundante Hardware

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.3, A.15, A.16 und A.18 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen durch Verwendung von Hardwareredundanz, d. h. Verwendung zusätzlicher Hardware, die nicht erforderlich ist, um die Prozessfunktionen auszuführen.

Beschreibung: Redundante Hardware kann verwendet werden, um mit angemessener Häufigkeit die festgelegten Sicherheitsfunktionen zu testen. Dieser Ansatz ist normalerweise zur Realisierung von A.1.1 oder A.2.2 notwendig.

A.2.2 Dynamische Prinzipien

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-2 angeführt.

Ziel: Erkennung statischer Ausfälle durch dynamische Signalverarbeitung.

Beschreibung: Ein erzwungener Wechsel von normalerweise statischen Signalen (intern oder extern erzeugt) hilft, statische Ausfälle in Bauteilen zu erkennen. Dieses Verfahren wird oft mit elektromechanischen Bauteilen in Verbindung gebracht.

Literaturhinweis:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld 1993.
http://www.bgia-handbuchdigital.de/330220

A.2.3 Standardtestschnittstelle (Access Port) und Boundary-Scan-Architektur

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.3, A.15 und A.18 der IEC 61508-2 angeführt.

Ziel: Steuerung und Beobachtung der Ereignisse an jedem Anschluss eines ICs.

Beschreibung: Der Boundary-Scan-Test ist ein IC-Entwurfsverfahren, das die Testbarkeit des ICs verbessert, indem das Problem gelöst wurde, einen Zugriff auf die Testpunkte des Schaltkreises im Inneren des ICs zu erhalten. In einem typischen Boundary-Scan-IC, das aus einer Kernlogik und Eingabe- und Ausgabepuffern besteht, wird eine Schieberegisterstufe zwischen der Kernlogik und den Eingabe- und Ausgabepuffern, die mit jedem IC-Anschluss verbunden sind, platziert. Jede Schieberegisterstufe ist in einer Boundary-Scan-Zelle enthalten. Die Boundary-Scan-Zelle kann die Ereignisse an jedem Eingabe- und Ausgabeanschluss eines ICs über den standardisierten Testanschluss steuern und beobachten. Die interne Testung der IC-Kernlogik wird durch Isolation der auf dem Chip befindlichen Kernlogik von den Signalen der umgebenden Bauteile und anschließende interne Selbsttests erreicht. Diese Tests können zur Ausfallerkennung im IC verwendet werden.

Literaturhinweis:

IEEE 1149-1:2001, *IEEE standard test access port and boundary-scan architecture, IEEE Computer Society*, 2001, ISBN: 0-7381-2944-5

A.2.4 (Nicht verwendet)

A.2.5 Überwachte Redundanz

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen durch Bereitstellung mehrerer funktionaler Einheiten, durch Überwachung des Verhaltens jeder dieser Einheiten, um Ausfälle zu erkennen, und durch Einleiten eines Übergangs in einen sicheren Zustand, wenn irgendeine Unstimmigkeit im Verhalten erkannt wird.

Beschreibung: Die Sicherheitsfunktion wird durch mindestens zwei Hardwarekanäle ausgeführt. Die Ausgaben dieser Kanäle werden überwacht, und ein sicherer Zustand wird eingeleitet, wenn ein Fehler erkannt wird (d. h. wenn die Ausgabesignale von allen Kanälen nicht identisch sind).

Literaturhinweise:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld 1993.
http://www.bgia-handbuchdigital.de/330220

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

A.2.6 Hardware mit automatischen Tests

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-2 angeführt.

Ziel: Erkennung von Fehlern durch periodische Tests der Sicherheitsfunktionen.

Beschreibung: Die Hardware wird vor Anlauf des Prozesses und zyklisch in geeigneten Intervallen getestet. Die EUC-Einrichtung setzt den Betrieb nur dann weiter fort, wenn jeder Test erfolgreich gewesen ist.

Literaturhinweise:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld 1993.
http://www.bgia-handbuchdigital.de/330220

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

A.2.7 Analogsignal-Überwachung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.3 und A.13 der IEC 61508-2 angeführt.

Ziel: Verbesserung des Vertrauens in gemessene Signale.

Beschreibung: Wenn eine Wahlmöglichkeit besteht, werden Analogsignale binären Ein-/Aus-Zuständen vorgezogen. Zum Beispiel werden energielose oder sichere Zustände durch analoge Signalpegel dargestellt, normalerweise mit Toleranzüberwachung der Signalpegel. Dieses Verfahren liefert eine kontinuierliche Überwachung und einen höheren Vertrauensgrad in die Übertragungseinrichtung, indem die notwendige Häufigkeit von Selbsttests einer Messkette verringert wird. Externe Schnittstellen, zum Beispiel Impulsleitungen, erfordern ebenfalls eine Testung.

A.2.8 Unterlastung

ANMERKUNG Dieses Verfahren/Maßnahme ist in 7.4.2.13 der IEC 61508-2 angeführt.

Ziel: Erhöhung der Zuverlässigkeit von Hardwarebauteilen.

Beschreibung: Hardwarebauteile werden in Bereichen betrieben, die aufgrund des Entwurfs des Systems weit unterhalb der festgelegten Höchstgrenzen liegen. Unterlastung ist die Methode, um zu gewährleisten, dass Bauteile unter allen normalen Betriebsumständen weit unterhalb ihrer Belastungsgrenzen betrieben werden.

A.3 Verarbeitungseinheiten (CPUs)

Allgemeines Ziel: Erkennung von Ausfällen, die zu falschen Ergebnissen in Verarbeitungseinheiten führen.

A.3.1 Selbsttest durch Software: begrenzte Anzahl von Mustern (ein Kanal)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen in der Verarbeitungseinheit so früh wie möglich.

Beschreibung: Die Hardware ist unter Verwendung von Standardverfahren aufgebaut, die keine besonderen Anforderungen an die Sicherheit in Betracht ziehen. Die Ausfallerkennung ist vollständig durch zusätzliche Softwarefunktionen realisiert, die Selbsttests durchführen. Diese verwenden mindestens zwei komplementäre Datenmuster (zum Beispiel 55hex und AAhex).

A.3.2 Selbsttest durch Software: Walking Bit (ein Kanal)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen im physikalischen Speicher (zum Beispiel Register) und Befehlsdecoder der Verarbeitungseinheit so früh wie möglich.

Beschreibung: Die Ausfallerkennung wird vollständig durch zusätzliche Softwarefunktionen realisiert, die Selbsttests durchführen. Diese verwenden ein Datenmuster (zum Beispiel ein wanderndes Bitmuster), welches den physikalischen Speicher (Daten- und Adressregister) und den Befehlsdecoder testet. Der Diagnosedeckungsgrad beträgt jedoch nur 90 %.

A.3.3 Selbsttest unterstützt durch Hardware (ein Kanal)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen in der Verarbeitungseinheit durch Verwendung besonderer Hardware, die die Geschwindigkeit und den Umfang der Fehlererkennung erhöht, so früh wie möglich.

Beschreibung: Zusätzliche besondere Hardwaremittel unterstützen Selbsttestfunktionen, um Ausfälle zu erkennen. Dies könnte zum Beispiel eine Hardwareeinheit sein, die zyklisch die Ausgabe eines bestimmten Bitmusters gemäß dem Watchdog-Prinzip überwacht.

A.3.4 Codierte Verarbeitung (ein Kanal)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen in der Verarbeitungseinheit so früh wie möglich.

Beschreibung: Verarbeitungseinheiten können mit besonderen fehlererkennenden oder fehlerkorrigierenden Schaltkreisverfahren entworfen werden. Bisher wurden diese Verfahren nur in relativ einfachen Schaltkreisen angewendet und sind nicht weit verbreitet, jedoch sollten zukünftige Entwicklungen nicht ausgeschlossen werden.

Literaturhinweise:

Le processeur codé: un nouveau concept appliqué à la sécurité des systèmes de transports. Gabriel, Martin, Wartski, Revue Générale des chemins de fer, No. 6, June 1990

Vital Coded Microprocessor Principles and Application for Various Transit Systems. P. Forin, IFAC Control Computers Communications in Transportation, 79-84, 1989

A.3.5 Gegenseitiger Vergleich durch Software

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen in der Verarbeitungseinheit durch dynamischen Softwarevergleich so früh wie möglich.

Beschreibung: Zwei Verarbeitungseinheiten tauschen gegenseitig Daten (einschließlich Ergebnissen, Zwischenergebnissen und Testdaten) aus. Ein Vergleich der Daten wird durch Verwendung von Software in jeder Einheit durchgeführt und erkannte Unterschiede führen zu einer Ausfallmeldung.

A.4 Unveränderliche Speicherbereiche

Allgemeines Ziel: Die Erkennung von Veränderungen der Informationen im invarianten Speicher.

A.4.1 Wortsicherungsverfahren mit Mehr-Bit-Redundanz (zum Beispiel ROM-Überwachung mit einem modifizierten Hammingcode)

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-2 angeführt.

ANMERKUNG 2 Siehe auch A.5.6 „RAM-Überwachung mit einem modifizierten Hammingcode oder Erkennung von Datenfehlern mit fehlererkennenden und -korrigierenden Codes (EDC)“ und C.3.2 „Fehlererkennende und korrigierende Codes“.

Ziel: Erkennung von allen Einbitfehlern, allen Zweibitfehlern, einigen Dreibitfehlern und einigen Fehlern, bei denen alle Bits eines 16-Bit-Wortes verfälscht werden.

Beschreibung: Jedes Speicherwort wird durch mehrere redundante Bits erweitert, um einen modifizierten Hammingcode mit einer Hammingdistanz von mindestens 4 zu erzeugen. Jedes Mal, wenn ein Wort gelesen wird, kann eine Testung der redundanten Bits bestimmen, ob eine Verfälschung stattgefunden hat oder nicht. Wenn ein Unterschied festgestellt wird, wird eine Ausfallmeldung erzeugt. Das Verfahren kann auch zur Erkennung von Adressierungsfehlern verwendet werden, indem die redundanten Bits mit Hilfe der Verknüpfung des Datenwortes mit dessen Adresse berechnet werden.

Literaturhinweise:

Prüfbare und korrigierbare Codes. W. W. Peterson, München, Oldenburg, 1967

Error detecting and error correcting codes. R. W. Hamming, The Bell System Technical Journal 29 (2), 147-160, 1950

A.4.2 Modifizierte Prüfsumme

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-2 angeführt.

Ziel: Erkennung aller ungeradzahligen Bitfehler, d. h. ungefähr 50 % aller möglichen Bitfehler.

Beschreibung: Eine Prüfsumme wird durch einen geeigneten Algorithmus erzeugt, der alle Worte in einem Speicherblock verwendet. Die Prüfsumme kann als zusätzliches Wort im ROM gespeichert werden, oder ein zusätzliches Wort kann in einem Speicherblock hinzugefügt werden, um zu gewährleisten, dass der Prüfsummenalgorithmus einen vorbestimmten Wert erzeugt. In einem späteren Speichertest wird unter Verwendung des gleichen Algorithmus wieder eine Prüfsumme erzeugt. Das Ergebnis wird mit dem gespeicherten oder definierten Wert verglichen. Wenn ein Unterschied festgestellt wird, wird eine Ausfallmeldung erzeugt.

A.4.3 Signatur mit einfacher Wortbreite (8 Bit)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-2 angeführt.

Ziel: Erkennung aller Einbitfehler und aller Mehrbitfehler innerhalb eines Wortes sowie ungefähr 99,6 % aller möglichen Bitfehler.

Beschreibung: Der Inhalt eines Speicherblocks wird unter Verwendung eines CRC-Algorithmus (en: cyclic redundancy check) zu einem Speicherwort komprimiert (entweder durch Verwendung von Hardware oder Software). Ein typischer CRC-Algorithmus behandelt den gesamten Inhalt eines Blocks als byte- oder bitseriellen Datenstrom, auf den unter Verwendung eines Generatorpolynoms eine laufende Polynomdivision durchgeführt wird. Der Rest der Division stellt den komprimierten Speicherinhalt dar – er ist die „Signatur“ des Speichers – und wird gespeichert. Die Signatur wird in späteren Tests nochmalig berechnet und mit der bereits gespeicherten verglichen. Im Falle eines Unterschiedes wird eine Ausfallmeldung erzeugt.

A.4.4 Signatur mit doppelter Wortbreite (16 Bit)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-2 angeführt.

Ziel: Erkennung aller Einbitfehler und aller Mehrbitfehler innerhalb eines Wortes sowie ungefähr 99,998 % aller möglichen Bitfehler.

Beschreibung: Dieses Verfahren berechnet eine Signatur unter Verwendung eines CRC-Algorithmus (en: cyclic redundancy check), jedoch umfasst das Ergebnis mindestens zwei Worte. Die erweiterte Signatur wird gespeichert, neu berechnet und verglichen, wie bei der Signatur mit einfacher Wortbreite. Im Falle eines Unterschiedes zwischen dem gespeicherten und dem neu berechneten Wert wird eine Ausfallmeldung erzeugt.

A.4.5 Blockwiederholung (zum Beispiel doppeltes ROM mit Hardware- oder Softwarevergleich)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-2 angeführt.

Ziel: Erkennung aller Bitfehler.

Beschreibung: Der Adressraum wird in zwei Speichern hinterlegt. Der erste Speicher wird wie üblich betrieben. Der zweite Speicher beinhaltet die gleiche Information und wird parallel zum ersten angesprochen. Die Ausgaben werden verglichen, und wenn ein Unterschied erkannt wird, wird eine Ausfallmeldung erzeugt. Um bestimmte Arten der Abweichungen der Bits zu erkennen, müssen die Daten in einem der beiden Speicher invers abgelegt und beim Lesen erneut invertiert werden.

A.5 Veränderliche Speicherbereiche

Allgemeines Ziel: Erkennung von Ausfällen während der Adressierung, des Schreibens, des Speicherns und des Lesens.

ANMERKUNG Soft-Errors werden in Tabelle A.1 der IEC 61508-2 als Fehler verzeichnet, die während des Betriebs aufzudecken sind oder bei der Herleitung des Anteils sicherer Ausfälle zu analysieren sind. Ursachen von Soft-Errors sind: (1) Alphateilchen vom radioaktiven Zerfall, (2) Neutronen, (3) externe elektromagnetische Störungen, (4) internes Übersprechen. Externe elektromagnetische Störungen werden durch andere Anforderungen dieser Internationalen Norm abgedeckt.

Die Auswirkungen von Alphateilchen und Neutronen sollten durch Maßnahmen zur Sicherheitsintegrität zur Laufzeit beherrscht werden. Für Hard-Errors wirksame Maßnahmen zur Sicherheitsintegrität müssen nicht für Soft-Errors wirksam sein, z. B. RAM-Tests wie Walkpath, Galpat usw. sind nicht wirksam, wohingegen überwachende Verfahren wie Parity und ECC mit dem wiederholenden Lesen der Speicherzellen wirksam sind.

Ein Soft-Error kommt vor, wenn ein Strahlungsereignis eine ausreichende Störung der Ladung verursacht, um den Datenzustand einer niedrig erregten Halbleiter-Speicherzelle, eines Registers, Latches oder Flip-Flops zu invertieren oder zu kippen. Der Fehler wird „Soft“ genannt, weil der Schaltkreis selbst durch die Strahlung nicht dauerhaft beschädigt wird. Soft-Errors werden in Single-Bit-Upsets (SBU), Single-Event-Upsets (SEU) und Multi-Bit-Upsets (MBU) klassifiziert.

Wenn der gestörte Schaltkreis ein speicherndes Element wie eine Speicherzelle oder ein Flip-Flop ist, wird der Zustand bis zum nächsten (beabsichtigten) Schreibzugriff gespeichert. Die neuen Daten werden korrekt gespeichert. In einem kombinatorischen Schaltkreis entspricht die Wirkung eher eine Störung, weil es einen kontinuierlichen Energiefluss von dem Bauteil gibt, der diesen Knoten treibt. Die Wirkung auf Verbindungs- und Kommunikationsleitungen kann auch eine Störung sein. Jedoch wird wegen der größeren Kapazität die Wirkung durch Alphateilchen und Neutronen als vernachlässigbar betrachtet.

Soft-Errors können für veränderliche Speicher jeder Art, d. h. für DRAM, SRAM, Register-Bänke in Mikroprozessoren, Caches, Pipelines, Konfigurationsregister von Einheiten wie ADC, DMA, MMU, Interrupt-Controller, komplexe Timer, relevant sein. Die Empfindlichkeit gegen Alphateilchen und Neutronen ist abhängig von der Kernspannung und der geometrischen Struktur. Kleinere geometrische Strukturen bei einer Kernspannung von 2,5 V und besonders unterhalb 1,8 V würden mehr Aufwand bei der Beurteilung und wirksamere Schutzmaßnahmen erfordern.

Die Soft-Error-Rate für (eingebettete) Speicher wird in einem Bereich von 700 Fit/MBit bis 1200 Fit/MBit ausgewiesen (siehe a) und i) unten). Dies ist ein Bezugswert zum Vergleich mit Daten, die aus dem Halbleiterproduktionsprozess kommen, mit dem das Bauteil realisiert wird. Bis vor kurzem wurden SBU als vorherrschend angesehen, aber die neueste Prognose (siehe a) unten) berichtet von einem wachsenden Prozentsatz der MBU an der Gesamtrate der Soft-Errors (en: soft-error rate, SER) für Technologien von geringer als 65 nm.

Die folgende(n) Literatur und Quellen liefern Einzelheiten über Soft-Errors:

a) Altitude SEE Test European Platform (ASTEP) and First Results in CMOS 130 nm SRAM. J-L. Autran, P. Roche, C. Sudre et al. Nuclear Science, IEEE Transactions on Volume 54, Issue 4, Aug. 2007 Page(s):1002 – 1009;

b) Radiation-Induced Soft Errors in Advanced Semiconductor Technologies, Robert C. Baumann, Fellow, IEEE, IEEE TRANSACTIONS ON DEVICE AND MATERIALS RELIABILITY, VOL. 5, NO. 3, SEPTEMBER 2005;

c) Soft errors' impact on system reliability, Ritesh Mastipuram and Edwin C Wee, Cypress Semiconductor, 2004;

d) Trends And Challenges In VLSI Circuit Reliability, C. Costantinescu, Intel, 2003, IEEE Computer Society;

e) Basic mechanisms and modeling of single-event upset in digital microelectronics, P. E. Dodd and L. W. Massengill, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 583–602, Jun. 2003;

f) Destructive single-event effects in semiconductor devices and ICs, F. W. Sexton, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 603–621, Jun. 2003;

g) Coming Challenges in Microarchitecture and Architecture, Ronen, Mendelson, Proceedings of the IEEE, Volume 89, Issue 3, Mar 2001 Page(s):325 – 340;

h) Scaling and Technology Issues for Soft Error Rates, A Johnston, 4th Annual Research Conference on Reliability Stanford University, October 2000;

i) International Technology Roadmap for Semiconductors (ITRS), several papers.

A.5.1 RAM-Test „Checkerboard“ oder „March“

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

Ziel: Vorwiegend zur Erkennung statischer Bitfehler.

Beschreibung: Ein schachbrettartiges Muster von Nullen und Einsen wird in die Zellen eines bitorientierten Speichers geschrieben. Die Zellen werden dann paarweise überprüft, um zu gewährleisten, dass die Inhalte gleich und korrekt sind. Die Adresse der ersten Zelle eines solchen Paares ist veränderlich und die Adresse der zweiten Zelle des Paares wird durch bitweise Invertierung der ersten Adresse gebildet. Im ersten Durchlauf wird der Adressbereich des Speichers von der veränderlichen Adresse zu höheren Adressen durchlaufen und in einem zweiten Durchlauf zu niedrigeren Adressen. Beide Durchläufe werden dann mit invertierter Vorbelegung wiederholt. Im Falle eines Unterschiedes wird eine Ausfallmeldung erzeugt.

Bei einem RAM-Test „March“ werden die Zellen eines bitorientierten Speichers durch eine einheitliche Bitfolge vorbelegt. Im ersten Durchlauf werden die Zellen in aufsteigender Folge überprüft. Jede Zelle wird auf korrekten Inhalt getestet und ihr Inhalt invertiert. Der Hintergrund, der im ersten Durchlauf erzeugt wird, wird in einem zweiten Durchlauf in absteigender Folge und in der gleichen Art und Weise behandelt. Die ersten beiden Durchläufe werden mit einer invertierten Vorbelegung in einem dritten und vierten Durchlauf wiederholt. Im Falle eines Unterschiedes wird eine Ausfallmeldung erzeugt.

A.5.2 RAM-Test „Walkpath“

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

Ziel: Erkennung statischer und dynamischer Bitfehler und Übersprechen zwischen Speicherzellen.

Beschreibung: Der zu testende Speicherbereich wird mit einer einheitlichen Bitfolge vorbelegt. Die erste Zelle wird dann invertiert und der verbleibende Speicherbereich wird überprüft, um zu gewährleisten, dass der Hintergrund korrekt ist. Danach wird die erste Zelle erneut in ihren ursprünglichen Wert invertiert, und das gesamte Verfahren wird für die nächste Zelle wiederholt. Ein zweiter Durchlauf des „wandernden Bit-Modells“ wird mit einer inversen Hintergrundbelegung durchgeführt. Im Falle eines Unterschiedes wird eine Ausfallmeldung erzeugt.

A.5.3 RAM-Test „Galpat“ oder „transparenter Galpat“

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

Ziel: Erkennung von statischen Bitfehlern und eines Großteils dynamischer Kopplungen.

Beschreibung: Im RAM-Test „Galpat“ wird der ausgewählte Speicherbereich zunächst einheitlich (d. h. alles Nullen oder alles Einsen) vorbelegt. Die erste zu testende Speicherzelle wird dann invertiert und alle verbleibenden Zellen werden überprüft, um zu gewährleisten, dass ihre Inhalte korrekt sind. Nach jedem Lesezugriff auf eine der verbleibenden Zellen wird auch die invertierte Zelle überprüft. Dieses Verfahren wird

für jede Zelle in dem gewählten Speicherbereich wiederholt. Ein zweiter Durchlauf wird mit der gegensätzlichen Vorbelegung durchgeführt. Jeder Unterschied erzeugt eine Ausfallmeldung.

Der „transparente Galpat"-Test ist eine Variation des obigen Verfahrens. Statt einer Vorbelegung aller Zellen im ausgewählten Speicherbereich wird der bestehende Inhalt unverändert beibehalten, und es werden Signaturen verwendet, um den Inhalt der Zellenauswahl zu vergleichen. Die erste zu testende Zelle im ausgewählten Bereich wird ausgewählt und die Signatur S1 aller verbleibenden Zellen im Bereich wird berechnet und gespeichert. Die zu testende Zelle wird daraufhin invertiert und die Signatur S2 aller verbleibenden Zellen erneut berechnet. (Nach jedem Lesezugriff auf eine der verbleibenden Zellen wird die invertierte Zelle ebenfalls getestet.) S2 wird mit S1 verglichen und jeder Unterschied erzeugt eine Ausfallmeldung. Die dem Test unterliegende Zelle wird erneut invertiert, um den ursprünglichen Inhalt wiederherzustellen, und die Signatur S3 aller verbleibenden Zellen wird neu berechnet und mit S1 verglichen. Jeder Unterschied erzeugt eine Ausfallmeldung. Alle Speicherzellen in dem ausgewählten Bereich werden in der gleichen Art und Weise getestet.

A.5.4 RAM-Test „Abraham"

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

Ziel: Erkennung aller Stuck-at-Fehler und Kopplungen zwischen Speicherzellen.

Beschreibung: Der Anteil erkannter Fehler übersteigt den beim RAM-Test „Galpat". Die Anzahl der notwendigen Operationen, um den gesamten Speichertest durchzuführen, beträgt 30 *n*, wobei *n* die Anzahl der Speicherzellen ist. Der Test kann zur Anwendung während des Betriebszyklus transparent durchgeführt werden, indem der Speicher unterteilt und jeder Teil in verschiedenen Zeitsegmenten getestet wird.

Literaturhinweis:

Efficient Algorithms for Testing Semiconductor Random-Access Memories. R. Nair, S. M. Thatte, J. A. Abraham, IEEE Trans. Comput. C-27 (6), 572-576, 1978

A.5.5 Ein-Bit-Redundanz (zum Beispiel RAM-Überwachung mit einem Parity-Bit)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

Ziel: Erkennung von 50 % aller möglichen Bitfehler im getesteten Speicherbereich.

Beschreibung: Jedes Speicherwort wird um ein Bit (das Paritätsbit) erweitert, welches jedes Wort zu einer geraden oder ungeraden Anzahl logischer Einsen vervollständigt. Die Parität des Datenworts wird bei jedem Lesezugriff getestet. Wenn die falsche Anzahl von Einsen gefunden wird, wird eine Ausfallmeldung erzeugt. Die Wahl von gerader oder ungerader Parität sollte im Hinblick darauf getroffen werden, ob das Nullwort (alles Nullen) oder das Einswort (alles Einsen) im Falle eines Ausfalles das ungünstigste ist. Dann ist dieses Wort ein ungültiger Code. Die Parität kann auch verwendet werden, um Adressierungsfehler zu erkennen, wenn die Parität durch die Verknüpfung des Datenwortes mit seiner Adresse berechnet wird.

A.5.6 RAM-Überwachung mit einem modifizierten Hammingcode oder Erkennung von Datenfehlern mit fehlererkennenden und -korrigierenden Codes (en: error-detection-correction codes, EDC)

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

ANMERKUNG 2 Siehe auch A.4.1 „Wortsicherungsverfahren mit Mehr-Bit-Redundanz (zum Beispiel ROM-Überwachung mit einem modifizierten Hammingcode)" und C.3.2 „Fehlererkennende und korrigierende Codes".

Ziel: Erkennung aller ungeradzahligen Bitfehler, aller Zweibitfehler, einiger Dreibit- und einiger Mehrbitfehler.

Beschreibung: Jeder Speicherzugriff wird mit mehreren redundanten Bits erweitert, um einen modifizierten Hammingcode mit einer Hammingdistanz von mindestens 4 zu erzeugen. Bei jedem Lesezugriff kann man

durch Testung der redundanten Bits bestimmen, ob eine Verfälschung stattgefunden hat. Wenn ein Unterschied gefunden wird, wird eine Ausfallmeldung erzeugt. Das Verfahren kann auch zur Erkennung von Adressierungsfehlern verwendet werden, indem die redundanten Bits durch die Verknüpfung des Datums mit seiner Adresse berechnet werden.

Literaturhinweise:

Prüfbare und korrigierbare Codes. W. W. Peterson, München, Oldenburg, 1967

Error detecting and error correcting codes. R. W. Hamming, The Bell System Technical Journal, 29 (2), 147-160, 1950

A.5.7 Doppeltes RAM mit Hardware- oder Softwarevergleich und Schreib-/Lesetest

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.6 der IEC 61508-2 angeführt.

Ziel: Erkennung aller Bitfehler.

Beschreibung: Der Adressraum wird in zwei Speichern hinterlegt. Der erste Speicher wird wie üblich betrieben. Der zweite Speicher beinhaltet die gleiche Information und wird parallel zum ersten angesprochen. Die Ausgaben werden verglichen, und wenn ein Unterschied erkannt wird, wird eine Ausfallmeldung erzeugt. Um bestimmte Arten der Abweichungen der Bits zu erkennen, müssen die Daten in einem der beiden Speicher invers abgelegt und beim Lesen erneut invertiert werden.

A.6 E/A-Einheiten und Schnittstellen (externe Kommunikation)

Allgemeines Ziel: Fehlererkennung in Eingabe- und Ausgabeeinheiten (digital, analog, seriell oder parallel) und Verhinderung des Sendens unzulässiger Ausgaben an den Prozess.

A.6.1 Testmuster

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.7, A.13 und A.14 der IEC 61508-2 angeführt.

Ziel: Erkennung statischer Ausfälle (Stuck-at-Fehler) und Übersprechen.

Beschreibung: Dies ist ein Datenfluss-unabhängiger, zyklischer Test von Ein- und Ausgabeeinheiten. Er verwendet ein definiertes Testmuster, um die beobachteten mit den zugehörigen erwarteten Werten zu vergleichen. Die Testmusterinformation, der Testmusterempfang und die Testmusterauswertung müssen alle voneinander unabhängig sein. Die EUC-Einrichtung sollte durch das Testmuster nicht unzulässig beeinflusst werden.

A.6.2 Codesicherung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.7, A.15, A.15 und A.18 der IEC 61508-2 angeführt.

Ziel: Erkennung von zufälligen Hardwareausfällen und systematischen Ausfällen im Eingabe- und Ausgabedatenfluss.

Beschreibung: Dieses Verfahren schützt die Eingabe- und Ausgabeinformation sowohl vor systematischen Ausfällen als auch vor zufälligen Hardwareausfällen. Die Codesicherung stellt eine Datenfluss-abhängige Ausfallerkennung der Eingabe- und Ausgabeeinheiten, basierend auf Informationsredundanz und/oder Zeitredundanz, bereit. Normalerweise werden redundante Informationen den Eingabe- und/oder Ausgabedaten überlagert. Dies stellt ein Mittel bereit, um den korrekten Betrieb der Eingabe- und Ausgabeschaltkreise zu überwachen. Viele Verfahren sind möglich, es kann zum Beispiel dem Ausgangssignal eines Sensors ein Trägerfrequenzsignal überlagert werden. Die Logikeinheit kann dann die Anwesenheit der Trägerfrequenz testen. Es können redundante Codebits einem Ausgabekanal hinzugefügt werden, um die Überwachung der Gültigkeit eines Signals zu ermöglichen, das zwischen Logikeinheit und Stellglied übertragen wird.

A.6.3 Mehrkanalige parallele Ausgabe

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.7 der IEC 61508-2 angeführt.

Ziel: Erkennung von zufälligen Hardwareausfällen (Stuck-at-Fehlern), Ausfällen verursacht durch äußere Einflüsse, Timing-Fehlern, Adressierungsfehlern, Driftfehlern und transienten Fehlern.

Beschreibung: Dies ist eine Datenfluss-abhängige, mehrkanalige parallele Ausgabe mit unabhängigen Ausgaben für die Erkennung von zufälligen Hardwareausfällen. Die Ausfallerkennung wird durch externe Vergleicher durchgeführt. Im Falle eines Ausfalls wird die EUC-Einrichtung direkt abgeschaltet. Diese Maßnahme ist nur wirksam, wenn sich der Datenfluss während des Diagnose-Testintervalls verändert.

A.6.4 Überwachte Ausgaben

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.7 der IEC 61508-2 angeführt.

Ziel: Erkennung individueller Ausfälle, von Ausfällen verursacht durch äußere Einflüsse, Timing-Fehlern, Adressierungsfehlern, Driftfehlern (von analogen Signalen) und transienten Fehlern.

Beschreibung: Dies ist ein Datenfluss-abhängiger Ausgabevergleich mit unabhängigen Eingaben, um Übereinstimmung mit einem definierten Toleranzbereich (Zeit, Wert) zu gewährleisten. Ein erkannter Ausfall kann nicht immer auf die defekte Ausgabe bezogen werden. Diese Maßnahme ist nur wirksam, wenn sich der Datenfluss während des Diagnose-Testintervalls verändert.

A.6.5 Eingabevergleich/-entscheidung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.7 und A.13 der IEC 61508-2 angeführt.

Ziel: Erkennung individueller Ausfälle, von Ausfällen verursacht durch äußere Einflüsse, Timing-Fehlern, Adressierungsfehlern, Driftfehlern (von analogen Signalen) und transienten Fehlern.

Beschreibung: Dies ist ein Datenfluss-abhängiger Vergleich unabhängiger Eingänge, um Übereinstimmung mit einem definierten Toleranzbereich (Zeit, Wert) zu gewährleisten. Es wird 1-aus-2, 2-aus-3 oder bessere Redundanz verwendet. Diese Maßnahme ist nur wirksam, wenn sich der Datenfluss während des Diagnose-Testintervalls verändert.

A.7 Datenwege (interne Kommunikation)

Allgemeines Ziel: Erkennung von Ausfällen, die durch einen Defekt bei der Informationsübertragung verursacht werden.

A.7.1 Ein-Bit-Hardwareredundanz

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-2 angeführt.

Ziel: Erkennung aller ungeraden Bitfehler, d. h. 50 % aller möglichen Bitfehler in der Datenfolge.

Beschreibung: Der Bus wird um eine Leitung (Bit) erweitert, und diese zusätzliche Leitung (Bit) wird verwendet, um durch Prüfung der Parität Ausfälle zu erkennen.

A.7.2 Mehr-Bit-Hardwareredundanz

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen während der Kommunikation über den Bus und in Verbindungen mit serieller Übertragung.

Beschreibung: Der Bus wird um zwei oder mehrere Leitungen (Bits) erweitert, und diese zusätzlichen Leitungen (Bits) werden verwendet, um unter Verwendung des Hammingcode-Verfahrens Ausfälle zu erkennen.

A.7.3 Vollständige Hardwareredundanz

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen während der Kommunikation durch Vergleich der Signale auf zwei Bussen.

Beschreibung: Der Bus wird verdoppelt und die zusätzlichen Leitungen (Bits) werden zur Erkennung von Ausfällen verwendet.

A.7.4 Inspektion durch Verwendung von Testmustern

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-2 angeführt.

Ziel: Erkennung statischer Ausfälle (Stuck-at-Fehler) und Übersprechen.

Beschreibung: Dies ist ein Datenfluss-unabhängiger, zyklischer Test der Datenwege. Er verwendet ein definiertes Testmuster, um die beobachteten mit den zugehörigen erwarteten Werten zu vergleichen.

Die Testmusterinformation, der Testmusterempfang und die Testmusterauswertung müssen alle voneinander unabhängig sein. Die EUC-Einrichtung sollte nicht in unzulässiger Weise durch das Testmuster beeinflusst werden.

A.7.5 Übertragungsredundanz

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-2 angeführt.

Ziel: Erkennung transienter Ausfälle in der Buskommunikation.

Beschreibung: Die Information wird mehrmalig in Folge übertragen. Die Wiederholung wirkt nur gegen transiente Ausfälle.

A.7.6 Informationsredundanz

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-2 angeführt.

Ziel: Erkennung von Ausfällen in der Buskommunikation.

Beschreibung: Daten werden in Blöcken zusammen mit einer für jeden Block berechneten Prüfsumme übertragen. Der Empfänger berechnet erneut die Prüfsumme der erhaltenen Daten und vergleicht das Ergebnis mit der empfangenen Prüfsumme.

A.8 Spannungsversorgung

Allgemeines Ziel: Erkennung oder Tolerierung von Ausfällen, verursacht durch Defekte in der Spannungsversorgung.

A.8.1 Überspannungsschutz mit Sicherheitsabschaltung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.9 der IEC 61508-2 angeführt.

Ziel: Schutz des sicherheitsbezogenen Systems vor Überspannungen.

Beschreibung: Eine Überspannung wird früh genug erkannt, damit alle Ausgänge durch eine Abschaltroutine in einen sicheren Zustand geschaltet werden können oder zu einer zweiten Versorgungseinheit umgeschaltet werden kann.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.8.2 Spannungsüberwachung (sekundärseitig)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.9 der IEC 61508-2 angeführt.

Ziel: Überwachung der sekundärseitigen Spannungen und Einleitung eines sicheren Zustands, wenn die Spannung sich nicht im festgelegten Bereich befindet.

Beschreibung: Die sekundärseitige Spannung wird überwacht, und wenn sie sich nicht in ihrem festgelegten Bereich befindet, wird eine Abschaltung eingeleitet oder zu einer zweiten Versorgungseinheit umgeschaltet.

A.8.3 Energieabschaltung mit Sicherheitsabschaltung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.9 der IEC 61508-2 angeführt.

Ziel: Abschaltung der Energie mit Speicherung aller sicherheitskritischen Informationen.

Beschreibung: Eine Über- oder Unterspannung wird früh genug erkannt, damit (wenn erforderlich) der interne Status in einem nichtflüchtigen Speicher gesichert werden kann und damit alle Ausgänge durch die Abschaltroutine in einen sicheren Zustand gesetzt werden können oder damit zu einer zweiten Versorgungseinheit umgeschaltet werden kann.

A.9 Zeitliche und logische Programmlaufüberwachung

ANMERKUNG Diese Gruppe von Verfahren und Maßnahmen ist in den Tabellen A.15, A.16 und A.18 der IEC 61508-2 angeführt.

Allgemeines Ziel: Erkennung eines defekten Programmablaufes. Ein defekter Programmablauf liegt vor, wenn einzelne Elemente eines Programms (zum Beispiel Softwaremodule, Unterprogramme oder Befehle) in falscher Reihenfolge oder falschen Zeitabständen verarbeitet werden oder wenn der Prozessortakt fehlerhaft ist.

A.9.1 Watchdog mit separater Zeitbasis ohne Zeitfenster

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.10 und A.11 der IEC 61508-2 angeführt.

Ziel: Überwachung des Verhaltens und der Plausibilität des Programmablaufes.

Beschreibung: Externe Zeitglieder mit einer separaten Zeitbasis (zum Beispiel Watchdog-Timer) werden periodisch getriggert, um das Verhalten des Rechners und die Plausibilität des Programmablaufes zu überwachen. Es ist wichtig, dass die Triggerpunkte korrekt im Programm platziert werden. Der Watchdog wird nicht in einem festen Zeitraster getriggert, aber ein maximales Intervall wird festgelegt.

A.9.2 Watchdog mit separater Zeitbasis und Zeitfenster

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.10 und A.11 der IEC 61508-2 angeführt.

Ziel: Überwachung des Verhaltens und der Plausibilität des Programmablaufes.

Beschreibung: Externe Zeitglieder mit einer separaten Zeitbasis (zum Beispiel Watchdog-Timer) werden periodisch getriggert, um das Verhalten des Rechners und die Plausibilität des Programmablaufes zu überwachen. Es ist wichtig, dass die Triggerpunkte korrekt im Programm platziert werden. Für den Watchdog-Timer wird eine untere und obere Grenze festgelegt. Wenn der Programmablauf einer längere oder kürzere Zeit als erwartet dauert, wird eine Notaktion ausgeführt.

A.9.3 Logische Überwachung des Programmablaufs

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.10 und A.11 der IEC 61508-2 angeführt.

Ziel: Überwachung des korrekten Ablaufes von einzelnen Programmabschnitten.

Beschreibung: Der korrekte Ablauf der einzelnen Programmabschnitte wird durch Software (Zählverfahren, Schlüsselverfahren) oder durch Anwendung externer Überwachungseinrichtungen überwacht. Es ist wichtig, dass die Kontrollpunkte korrekt im Programm platziert werden.

A.9.4 Kombination von zeitlicher und logischer Überwachung des Programmablaufs

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.10 und A.11 der IEC 61508-2 angeführt.

Ziel: Überwachung des Verhaltens und des korrekten Ablaufes von einzelnen Programmabschnitten.

Beschreibung: Eine zeitliche Einrichtung (zum Beispiel Watchdog-Timer), die den Programmablauf überwacht, wird nur getriggert, wenn der Ablauf der einzelnen Programmabschnitte korrekt ausgeführt wurde.

A.9.5 Zeitliche Überwachung mit Test während des Betriebs

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.10 und A.11 der IEC 61508-2 angeführt.

Ziel: Erkennung von Fehlern in der zeitlichen Überwachung.

Beschreibung: Die zeitliche Überwachung wird nach dem Einschalten getestet. Ein Anlaufen ist nur möglich, wenn die zeitliche Überwachung korrekt funktioniert. Zum Beispiel könnte ein Wärmesensor durch einen erwärmten Widerstand beim Einschalten getestet werden.

A.10 Lüftung und Beheizung

ANMERKUNG Diese Gruppe von Verfahren und Maßnahmen ist in den Tabellen A.16 und A.18 der IEC 61508-2 angeführt.

Allgemeines Ziel: Beherrschung von Ausfällen in der Lüftung oder Heizung und/oder deren Überwachung, sofern dies sicherheitsrelevant ist.

A.10.1 Temperatursensor

Ziel: Erkennung von Über- oder Untertemperatur, bevor das System außerhalb der Spezifikation betrieben wird.

Beschreibung: Ein Temperatursensor überwacht die Temperatur an den kritischsten Stellen des sicherheitsbezogenen E/E/PE-Systems. Bevor die Temperatur den spezifizierten Bereich verlässt, wird eine Notaktion ausgeführt.

A.10.2 Lüfterkontrolle

Ziel: Erkennung des inkorrekten Betriebs eines Lüfters.

Beschreibung: Die Lüfter werden auf korrekten Betrieb überwacht. Wenn ein Lüfter nicht richtig arbeitet, wird eine Tätigkeit zur Instandhaltung (oder letztlich eine Notaktion) ausgeführt.

A.10.3 Auslösung der Sicherheitsabschaltung über eine thermische Sicherung

Ziel: Abschaltung des sicherheitsbezogenen Systems, bevor das System außerhalb seiner Temperatur-Spezifikation arbeitet.

Beschreibung: Eine Temperatursicherung wird verwendet, um das sicherheitsbezogene System abzuschalten. Für ein PES wird die Abschaltung durch eine Abschaltroutine eingeleitet, welche alle für die Notaktion erforderlichen Informationen speichert.

A.10.4 Gestaffelte Meldung von Thermosensoren und bedingter Alarm

Ziel: Anzeige, dass das sicherheitsbezogene System außerhalb seiner Temperatur-Spezifikation arbeitet.

Beschreibung: Die Temperatur wird überwacht und ein Alarm wird ausgelöst, wenn die Temperatur außerhalb des festgelegten Bereichs ist.

A.10.5 Zuschaltung von Umluftkühlung und Statusanzeige

Ziel: Verhinderung von Überhitzung durch Umluftkühlung.

Beschreibung: Die Temperatur wird überwacht und Zwangsluftkühlung eingeleitet, wenn die Temperatur höher als eine festgelegte Grenze wird. Der Anwender wird über den Status informiert.

A.11 Kommunikation und Massenspeicher

Allgemeines Ziel: Beherrschung von Ausfällen während der Kommunikation mit externen Quellen und Massenspeichern.

A.11.1 Trennung elektrischer Energieleitungen von Informationsleitungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.16 der IEC 61508-2 angeführt.

Ziel: Verringerung der Beeinflussung von Informationsleitungen durch hohe Ströme.

Beschreibung: Elektrische Versorgungsleitungen werden von den Leitungen getrennt, die die Information führen. Das elektrische Feld, das Spannungsspitzen in die Informationsleitungen induzieren könnte, nimmt mit dem Abstand ab.

A.11.2 Räumliche Trennung mehrfacher Leitungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.16 der IEC 61508-2 angeführt.

Ziel: Verringerung von Übersprechen auf Mehrfachleitungen.

Beschreibung: Leitungen, die redundante Signale übertragen, werden voneinander getrennt. Das elektrische Feld, das Spannungsspitzen in die Mehrfachleitungen induzieren könnte, nimmt mit dem Abstand ab. Diese Maßnahme reduziert auch Ausfälle infolge gemeinsamer Ursache.

A.11.3 Erhöhung der Störfestigkeit

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.16 und A.18 der IEC 61508-2 angeführt.

Ziel: Verringerung der elektromagnetischen Störungen auf das sicherheitsbezogene System.

Beschreibung: Entwurfsverfahren wie Schirmung und Filterung werden verwendet, um die Störfestigkeit des sicherheitsbezogenen Systems gegen elektromagnetische Störungen zu erhöhen, die eingestrahlt, über Versorgungs- oder Signalleitungen geführt werden oder aus elektrostatischen Entladungen resultieren können.

ANMERKUNG Siehe [16] und [17] für Anforderungen an die Störfestigkeit von sicherheitsbezogenen Systemen und von Betriebsmitteln, die vorgesehen sind, sicherheitsbezogene Funktionen (funktionale Sicherheit) in industriellen Anwendungen auszuführen.

Literaturhinweise:

IEC/TR 61000-5-2:1997, *Electromagnetic compatibility (EMC) – Part 5: Installation and mitigation guidelines – Section 2: Earthing and cabling*

Principles and Techniques of Electromagnetic Compatibility, Second Edition, C. Christopoulos, CRC Press, 2007, ISBN-10: 0849370353, ISBN-13: 978-0849370359

Noise Reduction Techniques in Electronic Systems. H. W. Ott, John Wiley Interscience, 2nd Edition, 1988

EMC for Product Designers. T. Williams, Newnes, 2007, ISBN 0750681705

Grounding and Shielding Techniques in Instrumentation, 3rd edition, R. Morrison. Wiley-Interscience, New York, 1986, ISBN-10: 0471838055, ISBN-13: 978-0471838050

A.11.4 Antivalente Signalübertragung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.7 und A.16 der IEC 61508-2 angeführt.

Ziel: Erkennung gleicher induzierter Spannungen in mehrfachen Signalübertragungsleitungen.

Beschreibung: Alle Informationen werden redundant mit antivalenten Signalen (zum Beispiel logisch 1 und 0) übertragen. Ausfälle infolge gemeinsamer Ursache (z. B. durch elektromagnetische Störstrahlung) können durch einen Antivalenzvergleicher erkannt werden.

Literaturhinweis:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch. 20. Lfg. V/93, Erich-Schmidt Verlag, Bielefeld.
http://www.bgia-handbuchdigital.de/330220

A.12 Sensoren

Allgemeines Ziel: Ausfallerkennung in den Sensoren des sicherheitsbezogenen Systems.

A.12.1 Referenzsensor

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.13 der IEC 61508-2 angeführt.

Ziel: Erkennung des inkorrekten Betriebs eines Sensors.

Beschreibung: Ein unabhängiger Referenzsensor wird verwendet, um den Betrieb eines Prozesssensors zu überwachen. Alle Eingabesignale werden in geeigneten Zeitabständen durch den Referenzsensor getestet, um Ausfälle des Prozesssensors zu erkennen.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.12.2 Zwangsöffnender Schalter

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.13 der IEC 61508-2 angeführt.

Ziel: Öffnen eines Kontaktes durch eine direkte mechanische Verbindung zwischen Stößel und Kontakt.

Beschreibung: Ein zwangsgeführter Schalter öffnet seinen Öffnerkontakt durch eine direkte mechanische Verbindung zwischen Stößel und Kontakt. Dies gewährleistet, dass jedes Mal, wenn der Stößel in der betätigten Stellung ist, der Schalterkontakt offen sein muss.

Literaturhinweis:

Verriegelung beweglicher Schutzeinrichtungen. F. Kreutzkampf, K. Becker, Sicherheitstechnisches Informations- und Arbeitsblatt 330210, BIA-Handbuch, 1. Lfg. IX/85, Erich Schmidt Verlag, Bielefeld

A.13 Stellglieder (Aktoren)

Allgemeines Ziel: Beherrschung von Ausfällen in den Stellgliedern eines sicherheitsbezogenen Systems.

A.13.1 Überwachung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.14 der IEC 61508-2 angeführt.

Ziel: Erkennung des inkorrekten Betriebs eines Aktors.

Beschreibung: Der Betrieb des Aktors wird überwacht (zum Beispiel durch die zwangsöffnenden Kontakte eines Relais, siehe Überwachung von Relaiskontakten in A.1.2). Die durch diese Überwachung eingeführte Redundanz kann zum Auslösen einer Notaktion verwendet werden.

Literaturhinweise:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Verriegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld

A.13.2 Kreuzweise Überwachung mehrfacher Aktoren

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.14 der IEC 61508-2 angeführt.

Ziel: Erkennung von Fehlern in Aktoren durch Vergleich der Ergebnisse.

Beschreibung: Jeder der mehrfach vorhandenen Aktoren wird durch einen unterschiedlichen Hardwarekanal überwacht. Im Falle eines Widerspruchs wird eine Notaktion eingeleitet.

A.14 Maßnahmen gegen die Einwirkung der physikalischen Umgebung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.16 und A.18 der IEC 61508-2 angeführt.

Ziel: Verhindern, dass die Einflüsse der physikalischen Umgebung (Wasser, Staub, korrodierende Substanzen) Ausfälle verursachen können.

Beschreibung: Das Gehäuse des Gerätes wird so entworfen, dass es der erwarteten Umgebung standhält.

Literaturhinweis:

IEC 60529:1989, *Degrees of protection provided by enclosures (IP code)*

Anhang B
(informativ)

Überblick über Verfahren und Maßnahmen für sicherheitsbezogene E/E/PE-Systeme: Vermeidung von systematischen Ausfällen (siehe IEC 61508-2 und IEC 61508-3)

ANMERKUNG Viele Verfahren in diesem Anhang sind auf Software anwendbar, werden jedoch im Anhang C nicht noch einmal angeführt.

B.1 Allgemeine Maßnahmen und Verfahren

B.1.1 Projektmanagement

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.1 bis B.6 der IEC 61508-2 angeführt.

Ziel: Vermeidung von Ausfällen durch Annahme eines organisatorischen Modells sowie von Regeln und Maßnahmen für die Entwicklung und den Test sicherheitsbezogener Systeme.

Beschreibung: Die wichtigsten und besten Maßnahmen sind

– die Erstellung eines Organisationsmodells speziell für die Qualitätssicherung, welches in einem Qualitätssicherungshandbuch niedergelegt ist; und
– die Festlegung von Regeln und Maßnahmen für die Erstellung und Validierung von sicherheitsbezogenen Systemen in Projektplanung und projektspezifischen Richtlinien.

Mehrere wichtige Basisprinzipien werden im Folgenden genannt:

– Definition einer Entwurfsorganisation:
  – Aufgaben und Verantwortlichkeiten der organisatorischen Einheiten;
  – Befugnisse der Abteilung für die Qualitätssicherung;
  – Unabhängigkeit der Qualitätssicherung (interne Inspektion) von der Entwicklung;
– Definition des Ablaufplans (Phasenmodelle):
  – Bestimmung aller Tätigkeiten, die während der Durchführung des Projekts sachdienlich sind, einschließlich der internen Inspektionen und deren Zeitplanung;
  – Projektaktualisierung;
– Definition eines feststehenden Ablaufs einer internen Inspektion:
  – Planung, Ausführung und Überprüfung der Inspektion (Inspektionstheorie);
  – Freigabemechanismus für Teilprodukte;
  – Sicherung von Wiederholungsinspektionen;
– Konfigurationsmanagement:
  – Verwaltung und Kontrolle der Versionen;
  – Erkennung der Auswirkungen von Modifikationen;
  – Inspektion der Beibehaltung der Eigenschaften nach Modifikationen;
– Einführung einer quantitativen Beurteilung von Qualitätssicherungsmaßnahmen:
  – Festlegung der Anforderungen;
  – Ausfallstatistiken;
– Einführung von rechnerunterstützten universellen Methoden, Werkzeugen und Personalschulung.

Literaturhinweise:

ISO 9001:2008, *Quality management systems – Requirements*

ISO/IEC 15504 (alle Teile), *Information technology – Process assessment*

CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition. M. B. Chrissis, M. Konrad, S. Shrum, Addison-Wesley Professional, 2006, ISBN-10: 0-3212-7967-0, ISBN-13: 978-0-3212-7967-5

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

B.1.2 Dokumentation

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in den Tabellen B.1 bis B.6 der IEC 61508-2 angeführt.

ANMERKUNG 2 Siehe auch Abschnitt 5 und Anhang A der IEC 61508-1.

Ziel: Vermeidung von Ausfällen und Erleichterung der Beurteilung der Systemsicherheit, indem jeder Schritt während der Entwicklung dokumentiert wird.

Beschreibung: Die betriebliche Fähigkeit und Sicherheit sowie die von allen in die Entwicklung einbezogenen Parteien aufgewandte Sorgfalt muss während der Beurteilung nachgewiesen werden. Besondere Aufmerksamkeit wird auf die Dokumentation gelegt, um jederzeit die während der Entwicklung aufgewandte Sicherheit und die Sicherheitsnachweise darlegen zu können.

Wichtige allgemeine Maßnahmen sind die Einführung von Richtlinien und Rechnerhilfsmitteln, d. h.:

- Richtlinien, die
  - einen Zusammenstellungsplan festlegen;
  - nach Checklisten für den Inhalt fragen; und
  - die Form des Dokuments bestimmen;
- Verwaltung der Dokumentation mit Hilfe von rechnerunterstützten und -organisierten Projektbibliotheken.

Individuelle Maßnahmen sind:

- Unterteilung der Dokumentation
  - in die Anforderungen;
  - des Systems (Anwenderdokumentation); und
  - der Entwicklung (einschließlich interner Inspektion);
- Zusammenstellung der Entwicklungsdokumentation gemäß dem Sicherheitslebenszyklus;
- Definition standardisierter Dokumentationsmodule, aus denen die Dokumente zusammengestellt werden können;
- eindeutige Identifikation der Bestandteile der Dokumentation;
- formalisierte Aktualisierung der Revisionen;
- Auswahl eindeutiger und verständlicher Beschreibungsmittel:
  - formale Notation der Festlegungen;
  - natürliche Sprache für Einführungen, Rechtfertigungen und Darstellungen der Absichten;
  - graphische Darstellungen der Beispiele;
  - semantische Definition graphischer Elemente; und
  - Fachwortverzeichnisse.

Literaturhinweise:

IEC 61506:1997, *Industrial-process measurement and control – Documentation of application software*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.1.3 Trennung der Sicherheitsfunktionen des E/E/PE-Systems von Nichtsicherheitsfunktionen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.1 und B.6 der IEC 61508-2 angeführt.

Ziel: Verhinderung, dass nicht sicherheitsbezogene Teile des Systems die sicherheitsbezogenen Teile in unerwünschter Art beeinflussen.

Beschreibung: In der Spezifikation sollte entschieden werden, ob eine Trennung der sicherheitsbezogenen Systeme und der nicht sicherheitsbezogenen Systeme möglich ist. Für Schnittstellen zwischen beiden Teilen sollten eindeutige Spezifikationen geschrieben werden. Eine eindeutige Trennung reduziert den Testaufwand für die sicherheitsbezogenen Systeme.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.1.4 Diversitäre Hardware

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.15, A.16 und A.18 der IEC 61508-2 angeführt.

Ziel: Erkennung systematischer Ausfälle während des Betriebs der EUC-Einrichtung, indem diversitäre Bauteile mit unterschiedlichen Ausfallraten und -arten verwendet werden.

Beschreibung: Unterschiedliche Bauteilarten werden für die diversitären Kanäle eines sicherheitsbezogenen Systems verwendet. Dies reduziert die Wahrscheinlichkeit von Ausfällen infolge gemeinsamer Ursache (zum Beispiel Überspannung, elektromagnetische Störungen) und erhöht die Wahrscheinlichkeit, solche Ausfälle zu erkennen.

Mit Hilfe von unterschiedlichen Mitteln, etwa unterschiedlichen physikalischen Prinzipien zur Ausführung einer geforderten Funktion, werden unterschiedliche Wege zur Problemlösung beschritten.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.2 Spezifikation der Anforderungen an den Entwurf des E/E/PE-Systems

Allgemeines Ziel: Erstellung einer Spezifikation, die so weit wie möglich vollständig, frei von Irrtümern, frei von Widersprüchen und einfach zu verifizieren ist.

B.2.1 Strukturierte Spezifikation

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.1 und B.6 der IEC 61508-2 angeführt.

Ziel: Reduzierung der Komplexität durch Erstellung einer hierarchischen Struktur von Teilanforderungen. Vermeidung von Ausfällen an den Schnittstellen zwischen den Anforderungen.

Beschreibung: Dieses Verfahren strukturiert die funktionale Spezifikation so in Teilanforderungen, dass zwischen ihnen nur möglichst einfache Beziehungen vorkommen. Diese Zerlegung wird so lange bereinigt, bis kleine eindeutige Teilanforderungen unterschieden werden können. Das Ergebnis der abschließenden Bereinigung ist eine hierarchische Struktur von Teilanforderungen, die einen Rahmen für die Spezifikation der Gesamtanforderungen bereitstellt. Diese Methode betont die Schnittstelle der Teilanforderungen und ist besonders wirksam zur Vermeidung von Schnittstellenausfällen.

Literaturhinweise:

ESA PSS 05-02, *Guide to the user requirements definition phase*, Issue 1, Revision 1, ESA Board for Software Standardisation and Control (BSSC), ESA, Paris, March 1995, ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS0502.pdf

Structured Analysis and System Specification. T. De Marco, Yourdon Press, Englewood Cliffs, 1979, ISBN-10: 0138543801, ISBN-13: 978-0138543808

B.2.2 Formale Methoden

ANMERKUNG 1 Siehe C.2.4 für Einzelheiten zu bestimmten formalen Methoden.

ANMERKUNG 2 Dieses Verfahren/Maßnahme ist in den Tabellen B.1, B.2 und B.6 der IEC 61508-2 angeführt.

Ziel: Formale Methoden übertragen die Grundsätze des mathematischen Denkens auf die Spezifikation und Implementierung von technischen Systemen. Sie vergrößern daher die Vollständigkeit, Widerspruchsfreiheit oder Fehlerfreiheit einer Spezifikation oder Implementierung.

Beschreibung: Formale Methoden stellen ein Mittel zur Verfügung, eine Systembeschreibung während der Spezifikations- und/oder Implementierungsphase zu entwickeln. Diese formalen Beschreibungen sind mathematische Modelle der Systemfunktion und/oder Struktur.

Somit könnte eine eindeutige Systembeschreibung verwirklicht werden (z. B. wird jeder Zustand eines Automaten durch seinen Anfangszustand, Eingaben und den Übergangsgleichungen des Automaten beschrieben), welche das Verständnis für das zugrunde liegende System vergrößert.

Die Auswahl einer geeigneten formalen Methode ist ein schwieriges Vorhaben, das das volle Verständnis für das System, seinen Entwicklungsprozess und den Umfang der mathematischen Modelle verlangt, die vielleicht verwendet werden könnten (siehe folgende Anmerkungen).

ANMERKUNG 3 Die Modelltheoreme von Interesse (Eigenschaften) stellen Garantien über das System dar, die viel mehr Vertrauen bereitstellen als die Simulation, d. h. das Beobachten von ausgewählten Handlungen des Systems.

ANMERKUNG 4 Die Nachteile von formalen Methoden können sein:

– festgelegter Abstraktionsgrad;
– Einschränkungen beim Erfassen der ganzen Funktionalität, die zum gegebenen Stadium relevant ist;
– Schwierigkeit, dass Implementierungsingenieure das Modell verstehen müssen;
– großer Arbeitsaufwand, um das Modell während des Lebenszyklus des Systems zu entwickeln, zu analysieren und zu warten;
– Verfügbarkeit von effizienten Werkzeugen, die das Erstellen und die Analyse des Modells unterstützen;
– Verfügbarkeit des Personals, das fähig ist, das Modell zu entwickeln und zu analysieren.

ANMERKUNG 5 Das gemeinsame Ziel der formalen Methoden ist klar das Modellieren der Zielfunktion des Systems, was häufig die Robustheit gegen Fehler eines Systems bagatellisiert. Deshalb müssen entsprechende formale Methoden einschließlich der Systemrobustheit ausgewählt werden.

Literaturhinweis:

Formal Specification: Techniques and Applications. N.Nissanke, Springer-Verlag Telos, 1999, ISBN-10: 1852330023

B.2.3 Semi-formale Methoden

ANMERKUNG 1 IEC 61508-3, Tabelle B.7, erweitert diese Liste des Anhangs B mit anderen semi-formalen softwarebezogenen Verfahren. Teil 3 führt auf:

– Logik-/Funktionsblockdiagramme: beschrieben in IEC 61131-3;

– Ablaufdiagramme: beschrieben in IEC 61131-3;

– Datenflussdiagramme: siehe C.2.2;

– endliche Zustandsmaschinen/Zustandsübergangsdiagramme: siehe B.2.3.2;

– Zeitgesteuerte Petri-Netze: siehe B.2.3.3;

– Entity-Relationship-Attribut-Datenmodelle: siehe B.2.4.4;

– Nachrichtenverlaufstabellen: siehe C.2.14;

– Entscheidungs-/Wahrheitstabellen: siehe C.6.1.

Ziel: Unzweideutige und lückenlose Beschreibung einer Spezifikation, so dass Irrtümer, Auslassungen und fehlerhaftes Verhalten erkannt werden können.

ANMERKUNG 2 Dieses Verfahren/Maßnahme ist in den Tabellen B.1, B.2 und B.6 der IEC 61508-2 und in den Tabellen A.1, A.2, A.4, B.7, C.1, C.2, C.4 und C.17 der IEC 61508-3 angeführt.

B.2.3.1 Allgemeines

Ziel: Beweis, dass der Entwurf seine Spezifikation erfüllt.

Beschreibung: Semi-formale Methoden stellen Mittel zur Entwicklung einer Systembeschreibung in einigen Stufen der Systementwicklung, d. h. Spezifikation, Entwurf oder Codierung, bereit. Die Beschreibung kann in einigen Fällen maschinell analysiert oder animiert werden, um verschiedenen Aspekte des Systemverhaltens anzuzeigen. Die Animation kann zusätzliches Vertrauen liefern, dass das System sowohl die wirklichen als auch die formal festgelegten Anforderungen erfüllt.

In den folgenden Unterabschnitten werden zwei semi-formale Methoden beschrieben.

B.2.3.2 Endliche Zustandsmaschinen, Zustandsübergangsdiagramme

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5, B.7, C.15 und C.17 der IEC 61508-3 angeführt.

Ziel: Gestaltung, Verifikation, Spezifikation oder Implementierung der Steuerungsstruktur eines Systems.

Beschreibung: Viele Systeme können hinsichtlich ihrer Zustände, ihrer Eingaben und ihrer Aktionen beschrieben werden. So könnte ein System im Zustand S1 nach Empfang der Eingabe E die Aktion A ausführen und in den Zustand S2 übergehen. Wir können ein System durch Beschreibung der Systemaktionen für jede Eingabe in jedem Zustand vollständig beschreiben. Das resultierende Systemmodell wird eine endliche Zustandsmaschine (oder endlicher Zustandsautomat) genannt. Es wird oft als ein sogenanntes Zustandsübergangsdiagramm dargestellt, das zeigt, wie sich ein System von einem Zustand in den anderen bewegt, oder als eine Matrix, in der die Dimensionen Zustand und Eingabe sind. Die Matrix-Zellen enthalten die Aktion und den neuen Zustand, der sich aus dem Empfang der Eingabe in dem gegebenen Zustand ergibt.

Wenn ein System komplex ist oder eine natürliche Struktur hat, kann sich dies in einer geschichteten endlichen Zustandsmaschine ausdrücken. Ein Zustandsdiagramm ist eine Art eines Zustandübergangsdiagramms, in dem verschachtelte Zustände erlaubt sind (der momentane Zustand spaltet sich in zwei oder

mehr Unterzustände auf, die sich parallel entwickeln und sich vielleicht an einer Stelle wieder zu einem einzelnen Zustand verbinden können). Dies trägt zur ausdrucksvollen Leistungsfähigkeit der Zustandübergangs-Notation bei, kann aber zusätzliche Komplexität hinzufügen, die in einem sicherheitsbezogenen System unerwünscht sein kann. Zustandsdiagramme haben eine formale (mathematische) Spezifikation. Zustandübergangsdiagramme können auf ein ganzes System oder auf ein Objekt oder Element innerhalb dessen angewendet werden.

Eine als eine endliche Zustandsmaschine beschriebene Spezifikation oder Entwurf kann geprüft werden auf

– Vollständigkeit (das System oder Objekt muss eine Aktion und einen neuen Zustand für jede Eingabe in jedem Zustand haben),
– Konsistenz (nur ein Zustandsübergang ist für jedes Zustands-/Eingabepaar möglich),
– Erreichbarkeit (ob es möglich ist, von einem Zustand zum anderen durch jede beliebige Folge von Eingaben zu gelangen) und
– Mangel an Endlosschleifen oder Dead-End-States usw.

Dies sind wichtige Eigenschaften für kritische Systeme. Werkzeuge zur Unterstützung dieser Prüfungen sind leicht zu entwickeln und verschiedene auf endliche Zustandsautomaten basierende Modelle (formale Sprachen, Petri-Netze, Markov-Graphen usw.) können verwendet werden. Es existieren auch Algorithmen, die die automatische Erzeugung von Testfällen zur Verifikation der Implementierung einer Zustandsmaschine oder zur Animation eines Modells einer Zustandsmaschine erlauben. Zustandübergangsdiagramme und Zustandsdiagramme werden weitgehend durch Werkzeuge unterstützt, die es ermöglichen, Diagramme zu zeichnen und zu überprüfen, und die Codes erzeugen, um die beschriebene Zustandmaschine zu realisieren.

Sie können auch zur Berechnung der Ausfallwahrscheinlichkeit verwendet werden, siehe B.6 und C.6.

Literaturhinweise:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN: 0321462254

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.2.3.3 Zeitbehaftete Petri-Netze

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5, B.7, C.15 und C.17 der IEC 61508-3 angeführt.

Ziel: Modellierung von wichtigen Gesichtspunkten des Systemverhaltens, Bewertung und möglicherweise Verbesserung der Sicherheit und betrieblicher Anforderungen durch Analyse und Neuentwurf.

Beschreibung: Petri-Netze sind ein besonderer Fall von endlichen Zustandsautomaten. Sie gehören zu einer Klasse von Modellen der Graphentheorie, die geeignet sind, Informations- und Kontrollflüsse in Systemen darzustellen, die nebeneinander laufen und asynchrones Verhalten haben.

Ein Petri-Netz ist ein Netzwerk von Stellen und Transitionen. Die Stellen können markiert oder nicht markiert sein. Eine Transition wird aktiviert, wenn alle Eingabestellen zu ihr markiert sind. Wenn sie aktiviert ist, darf sie (muss aber nicht) „feuern“. Ist dies geschehen, werden die Markierungen der Eingabestellen wieder entfernt und stattdessen alle Ausgabestellen der Transition markiert.

Mögliche Gefahren können als besondere Zustände (Markierungen) in dem Modell dargestellt werden. Das Petri-Netz-Modell kann unter Berücksichtigung von zeitlichen Eigenschaften des Systems erweitert werden. Obwohl sich „klassische“ Petri-Netze auf Aspekte des Kontrollflusses konzentrieren, sind mehrere Erweiterungen zur Einbeziehung von Datenflüssen in das Modell vorgeschlagen worden.

Sie stellen ebenfalls eine sehr effiziente Unterstützung zur Ausführung der Monte-Carlo-Simulation zur Verfügung, um die Berechnungen der Ausfallwahrscheinlichkeit auszuführen, siehe B.6.6.8.

Literaturhinweise:

Timed Petri Nets: Theory and Application. Jiacun Wang, Springer, 1998, ISBN 0792382706

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.2.4 Rechnerunterstützte Spezifikationswerkzeuge

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.1 und B.6 der IEC 61508-2 und in den Tabellen A.1, A.2, C.1 und C.2 der IEC 61508-3 angeführt.

B.2.4.1 Allgemeines

Ziel: Verwendung formaler Spezifikationsverfahren, um die automatische Erkennung von Zweideutigkeiten und der Vollständigkeit zu erleichtern.

Beschreibung: Dieses Verfahren erzeugt eine Spezifikation in Form einer Datenbank, die automatisiert überprüft werden kann, um die Übereinstimmung und Vollständigkeit zu bewerten. Das Spezifikationswerkzeug kann dem Anwender verschiedene Aspekte des festgelegten Systems durch eine Animation zeigen. Im Allgemeinen unterstützt dieses Verfahren nicht nur die Erstellung der Spezifikation, sondern auch den Entwurf und andere Phasen des Projektlebenszyklus. Spezifikationswerkzeuge können nach den folgenden Unterabschnitten klassifiziert werden.

B.2.4.2 Methodenunspezifische Werkzeuge

Ziel: Unterstützung des Anwenders zur Erstellung einer geeigneten Spezifikation, indem Hinweise und Verbindungen zu zugehörigen Teilen bereitgestellt werden.

Beschreibung: Das Spezifikationswerkzeug übernimmt einige Routinearbeiten des Anwenders und unterstützt das Projektmanagement. Es erzwingt keine besonderen Spezifikationsmethoden. Die relative Unabhängigkeit im Hinblick auf die Methode gestattet dem Anwender einen hohen Freiheitsgrad, gibt ihm aber nur wenig von der zur Erstellung von Spezifikationen notwendigen Unterstützung. Dies erschwert die Einarbeitung in das System.

B.2.4.3 Modellorientierte Verfahren mit hierarchischer Analyse

Ziel: Vermeidung von Unvollständigkeiten, Zweideutigkeiten und Widersprüchen in der Spezifikation, z. B. durch Unterstützung des Anwenders bei der Erstellung einer geeigneten Spezifikation, indem Übereinstimmung der Beschreibungen der Aktionen mit den Daten auf verschiedenen Abstraktionsgraden sichergestellt wird.

Beschreibung: Diese Methode gibt eine funktionale Darstellung des gewünschten Systems (strukturierte Analyse) bei verschiedenen Abstraktionsgraden (Genauigkeitsgrad) wieder. Es gibt eine große Sammlung solcher Modelle: endliche Automaten (en: finite automata) sind eine weitgehend angewendete Klasse solcher Modelle, um die Entwicklung von diskreten/digitalen Systemen zu beschreiben. Differentialgleichungen sind in diesem Sinne ähnlich und zielen auf kontinuierliche/analoge Systeme. Die Analyse auf verschiedenen Ebenen betrifft sowohl die Aktionen als auch die Daten. Die Bewertung der Zweideutigkeit und Vollständigkeit ist zwischen hierarchischen Ebenen wie auch zwischen zwei Funktionseinheiten (Modulen) auf der gleichen Ebene möglich (z. B. wird jeder Zustand eines Systemmodells durch seinen Anfangszustand, Eingaben und den Übergangsgleichungen des Automaten beschrieben).

ANMERKUNG Probleme von modellbasierenden Beschreibungen können sein: der Abstraktionsgrad; Einschränkungen beim Erfassen der ganzen Funktionalität, die zum gegebenen Stadium relevant ist; Schwierigkeit, dass Fachleute das Modell verstehen müssen (vom Lesen des Syntax bis zum Verstehen); großer Arbeitsaufwand, um ein Modell während des Lebenszyklus des Systems zu entwickeln, zu analysieren und zu warten; Verfügbarkeit von effizienten Werkzeugen, die das Erstellen und die Analyse des Modells unterstützen (die Entwicklung solcher Werkzeuge ist sicher ein sehr aufwendiges Vorhaben) und Verfügbarkeit des Personals, das fähig ist, das Modell zu entwickeln und zu analysieren.

Literaturhinweis:

System requirements analysis. Jeffrey O. Grady, Academic Press, 2006, ISBN 012088514X, 9780120885145

B.2.4.4 Entity-Relationship-Attribut-Datenmodelle

Ziel: Unterstützung des Anwenders zur Erstellung einer geeigneten Spezifikation durch Konzentration auf Einheiten innerhalb des Systems und den Beziehungen zwischen diesen.

Beschreibung: Das gewünschte System wird als eine Ansammlung von Objekten und ihren Beziehungen beschrieben. Das Werkzeug ermöglicht es, zu bestimmen, welche Beziehungen von dem System interpretiert werden können. Die Beziehungen lassen im Allgemeinen eine Beschreibung der hierarchischen Struktur der Objekte, des Datenflusses, der Beziehungen zwischen den Daten und welche Daten bestimmten Anwendungsprozessen unterliegen zu. Das klassische Verfahren wurde für Prozesssteuerungsanwendungen erweitert. Inspektionsmöglichkeiten und Unterstützung des Anwenders hängen von der Vielfalt der dargestellten Beziehungen ab. Auf der anderen Seite macht eine große Anzahl von Darstellungsmöglichkeiten die Anwendung dieses Verfahren komplex.

Literaturhinweis:

Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. Karl Eugene Wiegers, Microsoft Press, 2003, ISBN 0735618798, 9780735618794

B.2.4.5 Anregung und Antwort

Ziel: Unterstützung des Anwenders zur Erstellung einer geeigneten Spezifikation durch Identifizierung von Anregungs-Antwort-Beziehungen.

Beschreibung: Die Beziehungen zwischen den Objekten des Systems werden in einer Notation der „Anregung“ und „Antwort“ festgelegt. Es wird eine einfache und leicht erweiterte Sprache verwendet, die Sprachelemente zur Darstellung von Objekten, Beziehungen, Eigenschaften und Strukturen beinhaltet.

B.2.5 Checklisten

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.1, B.2 und B.6 der IEC 61508-2 und in den Tabellen A.10, B.8, C.10 und C.18 der IEC 61508-3 angeführt.

Ziel: Vollständig und kritisch anhand des Sicherheitslebenszyklus alle wichtigen Systemaspekte behandeln, ohne genaue Anforderungen niederzulegen.

Beschreibung: Eine Auswahl von Fragen sind von der Person, die die Checkliste anwendet, zu beantworten. Viele Fragen sind von allgemeiner Natur, und der Beurteilende muss diese so geeignet wie möglich für das zu beurteilende System interpretieren. Checklisten können für alle Phasen des Gesamt-Sicherheitslebenszyklus, des Sicherheitslebenszyklus des E/E/PE-Systems und des Software-Sicherheitslebenszyklus verwendet werden und sind insbesondere als ein Werkzeug zur Unterstützung der Beurteilung der funktionalen Sicherheit nützlich.

Um die vielen Arten von zu validierenden Systemen zu berücksichtigen, beinhalten die meisten Checklisten Fragen, die auf viele Systemarten anwendbar sind. Deshalb mag es Fragen in den Checklisten geben, die für das zu behandelnde System nicht zutreffen und die deshalb vernachlässigt werden sollten. Ebenso mag es bei einem besonderen System notwendig sein, die Standard-Checkliste mit speziellen Fragen zu ergänzen.

Es ist einleuchtend, dass die erfolgreiche Verwendung von Checklisten von der Fachkenntnis und dem Beurteilungsvermögen des Ingenieurs abhängig ist, der die Checkliste auswählt und anwendet. Daraus resultierend sollte jede von dem Ingenieur getroffene Entscheidung zu den ausgewählten Checklisten und zusätzlichen oder überflüssigen Fragen vollständig dokumentiert und gerechtfertigt werden. Dadurch soll sichergestellt werden, dass die Anwendung der Checklisten überprüfbar ist und dass die gleichen Ergebnisse erreicht werden, sofern nicht unterschiedliche Kriterien verwendet werden.

Genauigkeit ist das Ziel bei der Erstellung einer Checkliste. Wenn eine umfassende Rechtfertigung notwendig ist, dann sollte dies durch Verweis auf zusätzliche Dokumente erfolgen. Zur Dokumentation der Ergebnisse jeder Frage sollte eine begrenzte Auswahl von Antworten wie bestanden, durchgefallen und ergebnislos oder ähnlich verwendet werden. Diese Genauigkeit vereinfacht außerordentlich das Verfahren, um ein Gesamtergebnis aus Checklistenauswertungen zu erzielen.

Literaturhinweise:

IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Quality Assurance: From Theory to Implementation. Daniel Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452

IEC 61346 (alle Teile), *Industrial systems, installations and equipment and industrial products – Structuring principles and reference designation*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

B.2.6 Inspektion der Spezifikation

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.1 und B.6 der IEC 61508-2 angeführt.

Ziel: Vermeidung von Unvollständigkeiten und Widersprüchen in der Spezifikation.

Beschreibung: Die Inspektion ist ein allgemeines Verfahren, in dem verschiedene Eigenschaften einer Spezifikation von einer unabhängigen Arbeitsgruppe bewertet werden. Die Arbeitsgruppe stellt Fragen an den Ersteller, der diese zufriedenstellend beantworten muss. Die Prüfung sollte wenn möglich von einer Arbeitsgruppe ausgeführt werden, die nicht in die Erstellung der Spezifikation eingebunden war. Der erforderliche Grad der Unabhängigkeit wird durch den für das System erforderlichen Sicherheits-Integritätslevel bestimmt. Die unabhängigen Prüfer sollten in der Lage sein, die Systemfunktion in einer zweifelsfreien Art und Weise ohne Bezug zu weiteren Spezifikationen zu rekonstruieren. Sie müssen ebenfalls prüfen, ob alle zutreffenden Sicherheits- und technischen Aspekte durch die betrieblichen und organisatorischen Maßnahmen abgedeckt werden. Dieses Verfahren hat sich in der Praxis als sehr wirksam erwiesen.

Literaturhinweise:

IEC 61160:2005, *Design review*

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Quality Assurance: From Theory to Implementation. D. Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452

B.3 Entwurf und Entwicklung des E/E/PE-Systems

Allgemeines Ziel: Erstellung eines stabilen Entwurfs eines sicherheitsbezogenen Systems in Übereinstimmung mit der Spezifikation.

B.3.1 Beachtung von Richtlinien und Normen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.2 der IEC 61508-2 angeführt.

Ziel: Beachtung anwendungsspezifischer Normen (nicht in dieser Norm festgelegte).

Beschreibung: Während des Entwurfs des sicherheitsbezogenen Systems sollten Richtlinien befolgt werden. Diese Richtlinien sollten erstens zu sicherheitsbezogenen Systemen führen, die praktisch frei von Ausfällen sind, und zweitens die nachfolgende Validierung bezüglich der Sicherheit erleichtern. Sie können allgemeingültig, projektspezifisch oder spezifisch für eine einzelne Phase sein.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.3.2 Strukturierter Entwurf

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.6 der IEC 61508-2 angeführt.

Ziel: Reduzierung der Komplexität durch Erstellung einer hierarchischen Struktur von Teilanforderungen. Vermeidung von Schnittstellenausfällen zwischen den Anforderungen. Vereinfachung der Verifikation.

Beschreibung: Während des Entwurfs der Hardware sollten bestimmte Kriterien und Methoden verwendet werden. Zum Beispiel kann Folgendes erforderlich sein:

– ein hierarchisch strukturierter Schaltungsentwurf und
– Verwendung bereits erstellter und getesteter Schaltungsteile.

Auf ähnliche Weise ermöglicht die Verwendung strukturierter Diagramme während des Entwurfs der Software eine eindeutige Struktur der Softwaremodule. Die Struktur zeigt, wie die Module zueinander in Verbindung stehen, die genauen Daten, die zwischen ihnen ausgetauscht werden, und deren Abstimmung.

Literaturhinweise:

IEC 61346 (alle Teile), *Industrial systems, installations and equipment and industrial products – Structuring principles and reference designation*

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192

An Overview of JSD, J. R. Cameron, IEEE Trans SE-12 No. 2, Februar 1986

Structured Development for Real-Time Systems (3 Bände). P. T. Yourdon, P. T. Yourdon Press, 1985

Structured Development for Real-Time Systems (3 Bände). P. T. Ward, S. J. Mellor, Yourdon Press, 1985

Applications and Extensions of SADT. D. T. Ross, Computer, 25-34, April 1985

Essential Systems Analysis. St. M. McMenamin, F. Palmer, Yourdon Inc, 1984

Structured Analysis (SA): A language for communicating ideas. D. T. Ross, IEEE Trans. Software Eng, Vol. SE-3 (1), 16-34

B.3.3 Verwendung von bewährten Bauteilen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.6 der IEC 61508-2 angeführt.

Ziel: Reduzierung des Risikos vieler erstmaliger und unerkannter Fehler durch die Verwendung von Bauteilen mit speziellen Eigenschaften.

Beschreibung: Die Auswahl von betriebsbewährten Bauteilen erfolgt durch den Hersteller nach der Zuverlässigkeit der Elemente im Hinblick auf die Sicherheit (zum Beispiel die Verwendung von betrieblich erprobten Einheiten, um hohe Anforderungen an die Sicherheit zu erfüllen, oder die Speicherung sicherheitsrelevanter Programme nur in sicheren Speichern). Die Sicherheit von Speichern kann sich sowohl auf unberechtigten Zugriff als auch auf Umgebungseinflüsse (elektromagnetische Verträglichkeit, Strahlung usw.) beziehen und auf die Reaktion der Elemente im Falle eines auftretenden Ausfalls.

Literaturhinweis:

IEC 61163-1:2006, *Reliability stress screening – Part 1: Repairable assemblies manufactured in lots*

B.3.4 Modularisierung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.6 der IEC 61508-2 angeführt.

Ziel: Reduzierung der Komplexität und Vermeidung von Ausfällen bezogen auf Schnittstellen zwischen Teilsystemen.

Beschreibung: Auf allen Ebenen des Entwurfs wird jedes Teilsystem klar definiert und ist in der Größe beschränkt (nur wenige Funktionen). Die Schnittstellen zwischen Teilsystemen werden so einfach wie möglich gehalten und Querverbindungen (d. h. gemeinsame Daten, Informationsaustausch) werden gering gehalten. Die Komplexität der einzelnen Teilsysteme ist ebenfalls beschränkt.

Literaturhinweise:

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Reliability – Principles and Practices. G. J. Myers, Wiley-Interscience, New York, 1976, ISBN-10: 0471627658, ISBN-13: 978-0471627654

B.3.5 Rechnerunterstützte Entwurfswerkzeuge

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.6 der IEC 61508-2 und in den Tabellen A.4 und C.4 der IEC 61508-3 angeführt.

Ziel: Systematischere Ausführung des Entwurfsprozesses. Einbeziehung geeigneter automatischer Konstruktionselemente, die bereits verfügbar und getestet sind.

Beschreibung: Rechnerunterstützte Entwurfswerkzeuge (en: computer aided design, CAD) sollten, wenn verfügbar und durch die Komplexität des Systems gerechtfertigt, während des Entwurfs der Hardware und Software verwendet werden. Die Korrektheit solcher Werkzeuge sollte durch spezielle Tests, durch langfristig nachgewiesene zufriedenstellende Anwendung oder durch unabhängige Verifikation ihrer Ergebnisse im Hinblick auf das zu entwerfende sicherheitsbezogene System nachgewiesen werden.

Unterstützende Werkzeuge sollten nach ihrem Integrationsgrad ausgewählt werden. In diesem Zusammenhang werden Werkzeuge als integriert bezeichnet, wenn sie derart zusammenwirkend arbeiten, dass die Ausgaben des einen Werkzeugs geeigneten Inhalt und geeignetes Format zur automatischen Eingabe des nachfolgenden Werkzeugs haben. Demzufolge wird die Möglichkeit gemindert, dass menschliches Versagen beim Überarbeiten von Zwischenergebnissen eingebracht wird.

Literaturhinweise:

Overview of Technology Computer-Aided Design Tools and Applications in Technology Development, Manufacturing and Design. W. Fichtner, Journal of Computational and Theoretical Nanoscience, Volume 5, Number 6, June 2008, pp. 1089-1105(17)

The Electromagnetic Data Exchange: Much more than a Common Data Format. P.E. Frandsen et al. In *Proceeding of the 2nd European Conference on Antennas and Propagation*. The Institution of Engineering and Technology (IET), 2007, ISBN 978-0-86341-842-6

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

B.3.6 Simulation

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2, B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Ausführung einer systematischen und vollständigen Inspektion eines elektrischen/elektronischen Schaltkreises bezogen sowohl auf die funktionale Ausführung als auch auf die korrekte Dimensionierung der Bauteile.

Beschreibung: Die Funktion des Schaltkreises eines sicherheitsbezogenen Systems wird auf einem Rechner mittels eines softwaremäßigen Verhaltensmodells simuliert. Jedes einzelne Bauteil des Schaltkreises hat sein eigenes simuliertes Verhalten und die Reaktion des Schaltkreises, in dem dieses eingebunden ist, wird durch Beobachtung der Grenzdaten jedes Bauelements geprüft.

B.3.7 Inspektion (Überprüfungen und Analysen)

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.6 der IEC 61508-2 angeführt.

Ziel: Aufdeckung von Widersprüchen zwischen Spezifikation und Ausführung.

Beschreibung: Festgelegte Funktionen des sicherheitsbezogenen Systems werden geprüft und ausgewertet, um zu gewährleisten, dass das sicherheitsbezogene System den in der Spezifikation gegebenen Anforderungen entspricht. Jeder Punkt, der die Ausführung und die Anwendung des Produkts in Frage stellt, wird dokumentiert, so dass dieser ausgeräumt werden kann. Im Gegensatz zu einem Walkthrough ist der Autor während der Inspektion passiv und der Prüfer aktiv.

Literaturhinweise:

IEC 61160:2005, *Design Review*

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

ANSI/IEE 1028:1997, *IEEE Standard for software reviews*

Dependability of Critical Computer Systems 3. P. G. Bishop et al, Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.3.8 Walkthrough

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.6 der IEC 61508-2 angeführt.

Ziel: Aufdeckung von Widersprüchen zwischen Spezifikation und Ausführung.

Beschreibung: Festgelegte Funktionen des Entwurfs des sicherheitsbezogenen Systems werden geprüft und ausgewertet, um zu gewährleisten, dass das sicherheitsbezogene System den in der Spezifikation gegebenen Anforderungen entspricht. Mögliche Schwachpunkte der Realisierung und der Anwendung des Produkts werden dokumentiert, so dass diese ausgeräumt werden können. Im Gegensatz zu einer Inspektion ist der Autor während der Inspektion aktiv und der Prüfer passiv.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

ANSI/IEE 1028:1997, *IEEE Standard for software reviews*

Dependability of Critical Computer Systems 3. P. G. Bishop et al, Elsevier Applied Science, 1990, ISBN 1-85166-544-7

Methodisches Testen von Programmen. G. J. Myers, Oldenbourg Verlag, München, Wien, 1987

B.4 Betriebs- und Instandhaltungsverfahren für das E/E/PE-System

Allgemeines Ziel: Entwicklung von Verfahren, die die Ausfälle während des Betriebs und der Instandhaltung des sicherheitsbezogenen Systems zu vermeiden helfen.

B.4.1 Betriebs- und Instandhaltungsanweisungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.4 der IEC 61508-2 angeführt.

Ziel: Vermeidung von Irrtümern während des Betriebs und der Instandhaltung des sicherheitsbezogenen Systems.

Beschreibung: Betriebsanweisungen beinhalten wesentliche Informationen, wie das sicherheitsbezogene System anzuwenden und in Stand zu halten ist. In besonderen Fällen enthalten diese Anweisungen auch Beispiele, wie das sicherheitsbezogene System im Allgemeinen zu installieren ist. Alle Anweisungen müssen leicht verständlich sein. Bilder und Diagramme sollten verwendet werden, um komplexe Verfahren und Abhängigkeiten zu beschreiben.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.2 Benutzerfreundlichkeit

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.4 der IEC 61508-2 angeführt.

Ziel: Reduzierung der Komplexität während des Betriebs des sicherheitsbezogenen Systems.

Beschreibung: Der korrekte Betrieb des sicherheitsbezogenen Systems kann zu einem bestimmten Grad von der menschlichen Handhabung abhängen. Unter Berücksichtigung des relevanten Systementwurfs und

des Entwurfs des Arbeitsplatzes muss der Entwickler des sicherheitsbezogenen Systems gewährleisten, dass

– der Bedarf nach menschlichen Eingriffen auf ein absolutes Minimum beschränkt wird;
– der notwendige Eingriff so einfach wie möglich ist;
– das Potential eines Schadens durch Irrtum des Bedieners verringert wird;
– Betätigungseinrichtungen und Anzeigeeinrichtungen gemäß ergonomischen Anforderungen entworfen werden;
– die Betätigungseinrichtungen einfach, gut beschriftet und intuitiv anzuwenden sind;
– der Bediener nicht überfordert wird, gerade in extremen Situationen;
– die Schulung der Betätigungsverfahren und -einrichtungen dem Kenntnisstand und der Motivation der Schulungsteilnehmer angepasst ist.

B.4.3 Instandhaltungsfreundlichkeit

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.4 der IEC 61508-2 angeführt.

Ziel: Vereinfachung der Instandhaltungsverfahren für das sicherheitsbezogene System und Entwurf der notwendigen Mittel für wirkungsvolle Diagnose und Reparatur.

Beschreibung: Vorbeugende Instandhaltung und Reparatur wird oft unter schwierigen Umständen und unter Termindruck ausgeführt. Deswegen sollte der Entwickler des sicherheitsbezogenen Systems gewährleisten, dass

– sicherheitsbezogene Instandhaltungsmaßnahmen so selten wie möglich oder sogar idealerweise gar nicht notwendig werden;
– ausreichende, vernünftige und leicht handhabbare Diagnosewerkzeuge für die Reparaturen, die unvermeidbar sind, vorhanden sind – die Werkzeuge sollten alle notwendigen Schnittstellen beinhalten;
– wenn separate Diagnosewerkzeuge entwickelt oder beschafft werden müssen, diese rechtzeitig verfügbar sind.

B.4.4 Eingeschränkte Betriebsmöglichkeiten

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.4 und B.6 der IEC 61508-2 angeführt.

Ziel: Reduzierung der Bedienungsmöglichkeiten des Normalanwenders.

Beschreibung: Dieses Verfahren reduziert die Bedienungsmöglichkeiten durch

– Begrenzung des Betriebs innerhalb spezieller Betriebsarten, zum Beispiel durch Schlüsselschalter;
– Begrenzung der Anzahl der Bedienungselemente;
– Begrenzung der Anzahl der allgemein möglichen Betriebsarten.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.5 Betrieb nur durch erfahrene Bediener

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.4 und B.6 der IEC 61508-2 angeführt.

Ziel: Vermeidung von Betriebsausfällen verursacht durch Missbrauch.

Beschreibung: Der Bediener des sicherheitsbezogenen Systems wird entsprechend eines Wissenstandes geschult, der der Komplexität und dem Sicherheits-Integritätslevel des sicherheitsbezogenen Systems ange-

messen ist. Die Schulung schließt das Studium der Grundzüge des Produktionsprozesses und die Kenntnis der Beziehung zwischen dem sicherheitsbezogenen System und der EUC-Einrichtung ein.

Literaturhinweis:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.6 Schutz gegen Irrtümer des Bedieners

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabelle B.4 und B.6 der IEC 61508-2 angeführt.

Ziel: Schutz des Systems gegen alle Arten von Irrtümern des Bedieners.

Beschreibung: Falsche Eingaben (Wert, Zeit usw.) werden durch Plausibilitätskontrollen oder durch Überwachung der EUC-Einrichtung erkannt. Um diese Möglichkeit in den Entwurf zu integrieren, ist es notwendig, in einem sehr frühen Stadium festzulegen, welche Eingaben möglich und welche zulässig sind.

B.4.7 (Nicht verwendet)

B.4.8 Schutz vor Modifikation

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.17 und A.18 der IEC 61508-2 angeführt.

Ziel: Schutz des sicherheitsbezogenen Systems gegen Hardwaremodifikationen durch technische Mittel.

Beschreibung: Modifikationen oder Manipulationen werden automatisch erkannt, z. B. durch Plausibilitätskontrollen der Sensorsignale, Erkennung durch den technischen Prozess und durch automatische Einschalttests. Wenn eine Modifikation erkannt wird, wird eine Notaktion ausgeführt.

B.4.9 Eingabebestätigung

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.17 und A.18 der IEC 61508-2 angeführt.

Ziel: Erkennung eines Irrtums während des Betriebs durch den Bediener selbst, bevor die EUC-Einrichtung gestartet wird.

Beschreibung: Eine Eingabe über das sicherheitsbezogene System an die EUC-Einrichtung wird dem Bediener angezeigt, bevor sie an die EUC-Einrichtung gesendet wird, so dass der Bediener die Möglichkeit hat, Irrtümer zu erkennen und zu korrigieren. Ebenso wie auch unnormale, grundlose menschliche Aktionen sollte der Systementwurf obere und untere Geschwindigkeitsgrenzen und das Ziel menschlicher Reaktionen berücksichtigen. Dies würde zum Beispiel vermeiden, dass das System eine doppelte Tastenbetätigung anstelle einer einzigen erkennt, wenn der Bediener Tasten schneller als erwartet drückt, oder dass eine Taste zweimal gedrückt werden muss, weil das System (die Anzeige) zu langsam war, um auf das erste Mal zu reagieren. Die gleiche Tastenbetätigung sollte nicht mehr als einmal in Folge für kritische Dateneingaben gültig sein; die wiederholte Betätigung der „Eingabe“- oder „Ja“-Taste darf nicht zu einer unsicheren Aktion des Systems führen.

Es sollten Fragen mit mehreren Antwortmöglichkeiten und festgelegtem Verhalten bei Zeitüberschreitung vorgesehen werden. Damit wird der Fall abgedeckt, dass der Bediener das System auf eine Eingabe warten lässt.

Timeout-Verfahren sollten mit Multiple-Choice-Fragen (ja/nein usw.) eingebunden werden, um den Fall zu berücksichtigen, dass der Bediener das System warten lässt.

Die Fähigkeit zum Neustart einer sicherheitsbezogenen PES macht das System verwundbar, außer die Hardware und Software wurden unter Berücksichtigung solcher Möglichkeiten entworfen.

B.5 Integration des E/E/PE-Systems

Allgemeines Ziel: Vermeidung von Ausfällen während der Integrationsphase und Aufdeckung jedes beliebigen Ausfalls, der während dieser und vorhergehender Phasen gemacht wurde.

B.5.1 Funktionstest

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.3 und B.5 der IEC 61508-2 und in den Tabellen A.5, A.6, A.7, C.5, C.6 und C.7 der IEC 61508-3 angeführt.

Ziel: Aufdeckung von Ausfällen während der Spezifikations- und Entwurfsphase. Vermeidung von Ausfällen während der Realisierung und der Integration der Software und Hardware.

Beschreibung: Während der Funktionstests werden Überprüfungen ausgeführt, um festzustellen, ob die festgelegten Eigenschaften des Systems erreicht worden sind. Dem System werden Eingabedaten übergeben, die den erwarteten Normalbetrieb angemessen wiedergeben. Die Ausgaben werden beobachtet, und deren Reaktion wird mit der in der Spezifikation festgelegten verglichen. Abweichungen von der Spezifikation und Anzeichen einer unvollständigen Spezifikation werden dokumentiert.

Funktionstests von elektronischen Bauteilen, die für eine mehrkanalige Architektur entworfen wurden, betreffen gewöhnlich die hergestellten Bauteile, die mit vorvalidierten baugleichen Bauteilen getestet wurden. Zusätzlich wird empfohlen, dass die hergestellten Bauteile in Kombination mit anderen Bauteilen des gleichen Fertigungsloses geprüft werden, um Fehler gleicher Art aufzudecken, welche sonst verdeckt bleiben würden.

Außerdem muss die Arbeitskapazität des Systems ausreichend sein, siehe Anleitung in C.5.20.

Literaturhinweise:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Practical Software Testing: A Process-oriented Approach. I. Burnstein, Springer, 2003, ISBN 0387951318, 9780387951317

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.5.2 Black-Box-Test

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.3, B.5 und B.6 der IEC 61508-2 und in den Tabellen A.5, A.6, A.7, C.5, C.6 und C.7 der IEC 61508-3 angeführt.

Ziel: Test des dynamischen Verhaltens unter realen funktionalen Bedingungen. Aufdeckung von Nichteinhalten der Spezifikation und Beurteilung der Brauchbarkeit und Robustheit.

Beschreibung: Die Funktionen eines Systems oder Programms werden in einer festgelegten Umgebung mit festgelegten Prüfdaten ausgeführt, die systematisch von der Spezifikation nach festgelegten Kriterien abgeleitet wurden. Dieses zeigt das Systemverhalten auf und lässt einen Vergleich mit der Spezifikation zu. Um den Test durchzuführen, wird kein Wissen über die interne Struktur des Systems benötigt. Das Ziel ist es, zu bestimmen, ob die Funktionseinheit alle in der Spezifikation geforderten Funktionen ausführt. Das Verfahren der Bildung von Äquivalenzklassen ist ein Beispiel eines Kriteriums für die Festlegung von Black-Box-Testdaten. Der Eingabedatenraum wird anhand der Spezifikation in spezielle Eingabewertebereiche (Äquivalenzklassen) unterteilt. Testfälle werden dann aus Folgendem gebildet:

– Daten aus zulässigen Bereichen;
– Daten aus unzulässigen Bereichen;
– Daten aus den Bereichsgrenzen;
– Extremwerte;
– Kombinationen dieser Klassen.

Andere Kriterien können wirksam sein, um Testfälle während der verschiedenen Testtätigkeiten (Modultest, Integrationstest und Systemtest) auszuwählen. Das Kriterium „extreme Betriebsbedingungen“ wird zum Beispiel für den Systemtest innerhalb einer Validierung herangezogen.

Literaturhinweise:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

Essentials of Software Engineering. Frank F. Tsui, Orlando Karam. Jones & Bartlett, 2006. ISBN 076373537X, 9780763735371

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Systematic Software Testing. Rick D. Craig, Stefan P. Jaskiel. Artech House, 2002. ISBN 1580535089, 9781580535083

B.5.3 Statistisches Testen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.3, B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Test des dynamischen Verhaltens des sicherheitsbezogenen Systems und Beurteilung der Brauchbarkeit und der Robustheit.

Beschreibung: Dieser Ansatz testet ein System oder ein Programm mit Eingabedaten, die nach der erwarteten statistischen Verteilung der realen Eingabedaten – dem betrieblichen Profil – ausgewählt werden.

Literaturhinweise:

A discussion of statistical testing on a safety-related application. S Kuball, J H R May, Proc. IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007

Practical Reliability Engineering. P. O'Connor, D. Newton, R. Bromley, John Wiley and Sons, 2002, ISBN 0470844639, 9780470844632

Dependability of Critical Computer Systems 3. P. G. Bishop et al, Elsevier Applied Science, 1990, ISBN 1-85166-544-7

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

B.5.4 Felderfahrung

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in den Tabellen B.3, B.5 und B.6 der IEC 61508-2 angeführt.

ANMERKUNG 2 Siehe auch C.2.10 für eine ähnliche Maßnahme und Anhang D für einen statistischen Ansatz, beides im Zusammenhang mit Software.

Ziel: Verwendung von Felderfahrungen in verschiedenen Anwendungen als eine der Maßnahmen, um Fehler entweder während der Integration des E/E/PE-Systems und/oder während der Validierung der Sicherheit des E/E/PE-Systems zu vermeiden.

Beschreibung: Verwendung von Bauteilen oder Teilsystemen, die während längerer Betriebserfahrung, während der sie nicht geändert wurden, über ausreichend lange Zeit in zahlreichen verschiedenen Anwendungen keine oder nur unbedeutende Fehler gezeigt haben. Insbesondere bei komplexen Bauteilen mit einer Vielzahl von möglichen Funktionen (zum Beispiel Betriebssysteme, integrierte Schaltungen) muss der Entwickler darauf achten, welche Funktionen tatsächlich durch Felderfahrung getestet wurden. Ein Beispiel hierfür sind die Selbsttest-Routinen zur Fehlererkennung: im Falle keines Ausfalls der Hardware innerhalb der Betriebsdauer können die Routinen nicht als getestet betrachtet werden, da sie nie ihre Funktion zur Fehlererkennung ausgeführt haben.

Für die Angabe „Felderfahrung“ müssen die folgenden Bedingungen erfüllt sein:

– unveränderte Spezifikation;
– 10 Systeme in verschiedenen Anwendungen;
– 10^5 Betriebsstunden und mindestens ein Jahr Betriebsaufzeichnung.

ANMERKUNG 3 Anwendungsspezifische Normen können andere Werte festlegen.

Die Felderfahrung wird durch Dokumentation des Herstellers und/oder der betreibenden Firma nachgewiesen. Die Dokumentation muss mindestens enthalten:

– die genaue Bezeichnung des Systems und seiner Komponenten, einschließlich Versionsverfolgung der Hardware;
– die Anwender und Zeit der Anwendung;
– die Betriebsdauer;
– die Verfahren zur Auswahl der Systeme und Anwendungen, die den Beweis erbrachten;
– die Verfahren zur Fehlererkennung, Fehlerregistrierung wie auch Fehlerbeseitigung.

Literaturhinweise:

IEC 60300-3-2:2004, *Dependability management – Part 3-2: Application guide – Collection of dependability data from the field*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.6 Validierung der Sicherheit des E/E/PE-Systems

Allgemeines Ziel: Nachweis, dass das sicherheitsbezogene E/E/PE-System der Spezifikation der Anforderungen an die Sicherheit des E/E/PE-Systems und der Spezifikation der Anforderungen an den Entwurf des E/E/PE-Systems entspricht.

B.6.1 Funktionstest unter Umgebungsbedingungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.5 der IEC 61508-2 angeführt.

Ziel: Beurteilung, ob das sicherheitsbezogene System gegen typische Umwelteinflüsse geschützt ist.

Beschreibung: Das System wird verschiedenen Umweltbedingungen (zum Beispiel nach den Normen der Reihe IEC 60068 oder der Reihe IEC 61000) ausgesetzt, unter denen die Sicherheitsfunktionen nach ihrer Verfügbarkeit (und Kompatibilität mit den genannten Normen) beurteilt werden.

Literaturhinweise:

IEC 60068-1:1988, *Environmental testing – Part 1: General and guidance* and Amendment 1(1992)

IEC 61000-4-1:2006, *Electromagnetic compatibility (EMC) – Part 4-1: Testing and measurement techniques – Overview of IEC 61000-4 series*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.2 Test der Störfestigkeit gegen Stoßspannungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Test der Fähigkeit des sicherheitsbezogenen Systems, Stoßspannungen zu bewältigen.

Beschreibung: Das System wird mit einem typischen Anwendungsprogramm geladen und alle peripheren Leitungen (alle digitalen, analogen und seriellen Schnittstellen wie auch die Busverbindungen und Energieversorgung usw.) werden genormten Störsignalen ausgesetzt. Um eine quantitative Aussage zu erhalten, ist es vernünftig, sich der Grenz-Stoßspannung vorsichtig anzunähern. Die gewählte Störklasse wird nicht erreicht, wenn die Funktion ausfällt.

Literaturhinweise:

IEC 61000-4-5:2005, *Electromagnetic compatibility (EMC) – Part 4-5: Testing and measurement techniques – Surge immunity test*

C37.90.1-2002, *IEEE Standard for Surge Withstand Capability (SWC) Tests for Relays and Relay Systems Associated with Electric Power Apparatus*

B.6.3 (Nicht verwendet)

B.6.4 Statische Analyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 und in den Tabellen A.9, B.8, C.9 und C.18 der IEC 61508-3 angeführt.

Ziel: Vermeidung systematischer Fehler, die zum Zusammenbruch des zu testenden Systems entweder früh oder nach vielen Betriebsjahren führen können.

Beschreibung: Dieser systematische und möglicherweise rechnerunterstützte Ansatz prüft besondere statische Eigenschaften des Prototypensystems, um Vollständigkeit, Übereinstimmung und Eindeutigkeit im Hinblick auf die gestellten Anforderungen (zum Beispiel Konstruktionsrichtlinien, Systemspezifikationen und Gerätedatenblatt) zu gewährleisten. Eine statische Analyse ist reproduzierbar. Sie wird auf einem Prototyp angewendet, der ein gut definiertes Stadium der Vervollständigung erreicht hat. Einige Beispiele für eine statische Analyse der Hardware und Software sind:

– Analyse der Übereinstimmung des Datenflusses (wie der Test, ob ein Datenobjekt überall mit dem gleichen Wert gelesen wird);
– Kontrollflussanalyse (wie die Pfadbestimmung, Bestimmung von nicht zugänglichem Code);
– Schnittstellenanalyse (wie die Untersuchung der Variablenübergabe zwischen verschiedenen Softwaremodulen);
– Datenflussanalyse, um verdächtige Abläufe der Erstellung, Verweisung und Beseitigung von Variablen zu erkennen;
– Test der Einhaltung besonderer Richtlinien (zum Beispiel Luft- und Kriechstrecken, Montageabstände, Anordnung der physikalischen Einheiten, mechanische Empfindlichkeit der physikalischen Einheiten, ausschließliche Verwendung der eingeführten physikalischen Einheiten).

Literaturhinweise:

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0

An Industrial Perspective on Static Analysis. B A Wichmann, A A Canning, D L Clutterbuck, L A Winsborrow, N J Ward and D W R Marsh. Software Engineering Journal., 69 – 75, March 1995

Dependability of Critical Computer Systems 3. P. G. Bishop et al,. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.5 Dynamische Analyse und Test

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 und in den Tabellen A.5 A.9, B.2, C.5, C.9 und C.12 der IEC 61508-3 angeführt.

Ziel: Erkennung von Spezifikationsausfällen durch Inspektion des dynamischen Verhaltens eines Prototyps im fortgeschrittenen Stadium der Vervollständigung.

Beschreibung: Die dynamische Analyse eines sicherheitsbezogenen Systems wird an einem nahezu betriebsfertigen Prototypen des sicherheitsbezogenen Systems ausgeführt unter Eingabe von Daten, die typisch für die vorgesehene Betriebsumgebung sind. Die Analyse ist zufriedenstellend, wenn das beobachtete Verhalten des sicherheitsbezogenen Systems mit dem erforderlichen Verhalten übereinstimmt. Jeder Ausfall des sicherheitsbezogenen Systems muss korrigiert werden, und die neue Betriebsversion muss dann erneut analysiert werden.

Literaturhinweise:

The Concept of Dynamic Analysis. T. Ball, ESEC/FSE ’99, Lecture Notes in Computer Science, Springer, 1999, ISBN 978-3-540-66538-0

Dependability of Critical Computer Systems 3. P. G. Bishop et al,. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.6 Ausfallanalyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 angeführt.

B.6.6.1 Ausfallarten- und Auswirkungsanalyse (FMEA)

Ziel: Analyse eines Systementwurfs durch systematische Prüfung aller möglicher Ausfallquellen eines Bauteils des Systems und Bestimmung der Auswirkungen dieser Ausfälle auf das Verhalten und die Sicherheit des Systems.

Beschreibung: Gewöhnlich findet die Analyse im Rahmen einer Sitzung von Ingenieuren statt. Jedes Bauteil des Systems wird eines nach dem anderen analysiert, um die Menge seiner Ausfallarten zu erhalten, deren Ursachen und Auswirkungen (lokal und auf Ebene des Gesamtsystems), Verfahren zur Erkennung und Empfehlung. Wenn auf eine Empfehlung reagiert wird, wird dieses als getroffene, abhelfende Aktion dokumentiert.

Literaturhinweise:

IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

Reliability Technology. A. E. Green, A. J. Bourne, Wiley-Interscience, 1972, ISBN 0471324809

B.6.6.2 Ursache-Wirkungsdiagramme

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.3, B.4, C.13 und C.14 der IEC 61508-3 angeführt.

Ziel: Analyse und Darstellung der Abfolge von Ereignissen in einem Modell in kompakter graphischer Form, die sich in einem System als eine Wirkung der Kombinationen von Basisereignissen entwickeln können.

Beschreibung: Dieses Verfahren kann als eine Kombination aus Fehlerbaum- und Ereignisbaumanalyse betrachtet werden. Es beginnt bei einem kritischen (einleitenden) Ereignis. Das Wirkungsdiagramm wird unter Anwendung von JA/NEIN-Aussagen, die den Erfolg und Ausfall von einigen Operationen beschreiben, vorwärts durchlaufen. Dies erlaubt, Ereignis-Abfolgen zu gestalten, die entweder zu einem Unfall oder zu einer beherrschten Situation führen. Dann werden Ursachediagramme (d. h. Fehlerbäume) für jeden Ausfall erstellt. Beginnend von einer unfallartigen Situation und fortlaufend in rückwärts gerichteter Richtung erhält man einen umfassenden Fehlerbaum mit der unfallartigen Situation als Ausgangsereignis. In Vorwärts-Richtung werden die möglichen Auswirkungen, hervorgerufen durch ein Ereignis, bestimmt. Das Diagramm kann Knotensymbole beinhalten, die die Zustände einer Fortpflanzung entlang verschiedener Zweige von diesen Punkten aus beschreiben. Auch Zeitverzögerungen können eingeschlossen werden. Diese Zustände können ebenso mit Fehlerbäumen beschrieben werden. Die Fortpflanzungslinien können mit logischen Symbolen kombiniert werden, um das Diagramm im Umfang zu reduzieren. Für die Verwendung im Ursache-Wirkungsdiagramm ist eine Menge von Standardsymbolen definiert. Die Diagramme können zur Erstellung von Fehlerbäumen und zur Berechnung der Wahrscheinlichkeit des Auftretens bestimmter kritischer Auswirkungen verwendet werden. Sie können auch zur Erstellung von Ereignisbäumen verwendet werden.

Literaturhinweise:

IEC 62502, *Analysis techniques for dependability – Event tree analysis (ETA)* [1)]

The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis. B. S. Nielsen, Danish Atomic Energy Commission, Riso-M-1374, 1971

B.6.6.3 Ereignisbaumanalyse (en: event tree analysis, ETA)

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.4 und C.14 der IEC 61508-3 angeführt.

Ziel: Darstellung der Abfolge von Ereignissen in einem Modell in graphischer Form, die sich in einem System nach einem Startereignis entwickeln können, und dadurch ebenfalls Darstellung möglicher schwerwiegender Folgeereignisse. Ein Ereignisbaum ist schwierig aus Notizen zu erstellen. Die Anwendung des Wirkungsdiagramms ist hilfreich.

Beschreibung: Am Beginn des Diagramms werden die Ablaufbedingungen angeführt, die wichtig im Fortschritt von Ereignissen sind, das dem Startereignis folgt. Am Startereignis beginnend, das das Ziel der Analyse ist, wird eine Linie zu der ersten Bedingung des Ablaufs gezogen. Dort verzweigt das Diagramm in einen „ja“- oder „nein“-Zweig mit der Beschreibung, wie zukünftige Ereignisse von dieser Bedingung abhängen. Jedem dieser Zweige folgt in einer ähnlichen Weise eine weitere Bedingung. Nicht alle Bedingungen sind jedoch für alle Zweige wichtig. Dies wird bis zum Ende des Ablaufs fortgeführt und jeder Zweig des auf diesem Weg konstruierten Baumes stellt eine mögliche Auswirkung dar. Vorausgesetzt, dass die Bedingungen in den Abläufen unabhängig sind, kann der Ereignisbaum dazu verwendet werden, die Wahrscheinlichkeit der verschiedenen Auswirkungen, basierend auf der Wahrscheinlichkeit und Anzahl der Bedingungen im Ablauf, zu berechnen. Da die Bedingungen selten völlig unabhängig sind, müssen solche Berechnungen vorsichtig betrachtet und von erfahrenen Analytikern durchgeführt werden.

Literaturhinweise:

IEC 62502, *Analysis techniques for dependability – Event tree analysis (ETA)*[2)]

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

[1)] In Diskussion.

[2)] In Diskussion.

B.6.6.4 Softwarefunktionsfehleranalyse

ANMERKUNG Die Ausfallanalyse ist in den Tabellen A.10, B.4, C.10 und C.14 der IEC 61508-3 angeführt.

Ziel: Klassifizierung der Kritikalität von Bauteilen, die zu Verletzungen, Beschädigungen oder Verschlechterung des Systems durch Einzelausfälle führen können, um zu bestimmen, welche Bauteile besondere Aufmerksamkeit und Kontrollmaßnahmen während des Entwurfs oder des Betriebs erfordern.

Beschreibung: Diese Methode ist vergleichbar zur FMEA, aber es gibt eine oder mehrere Rubriken zur Darstellung der Kritikalität, die auf verschiedene Weise eingeteilt werden kann. Die arbeitsaufwendigste Methode wurde durch die „Society for Automotive Engineers“ (SAE) in ARP 926 beschrieben. In diesem Verfahren wird die Kritikalitätszahl für jedes Bauteil mit Hilfe der Anzahl der Ausfälle eines spezifischen Typs angegeben, die im Laufe von einer Million Betriebszyklen in einer kritischen Betriebsart erwartet werden. Die Kritikalitätszahl ist eine Funktion von neun Parametern, von denen die meisten gemessen werden müssen. Eine sehr einfache Methode zur Bestimmung der Kritikalität ist es, die Wahrscheinlichkeit von Bauteilausfällen mit dem Schadensausmaß, das erzeugt werden könnte, zu multiplizieren. Diese Methode ist einer einfachen Risikofaktorbeurteilung ähnlich.

Literaturhinweise:

IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

Software FMEA techniques. P.L.Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1

B.6.6.5 Fehlerbaumanalyse (en: fault tree analysis, FTA)

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.4 und C.14 der IEC 61508-3 angeführt.

Ziel: Unterstützung der Analyse von Ereignissen oder Kombinationen von Ereignissen, die zu einer Gefährdung oder zu schwerwiegenden Auswirkungen führen und Berechnung der Wahrscheinlichkeit des Ausgangsereignisses.

Beschreibung: Beginnend von einem Ereignis (dem Ausgangsereignis), das unmittelbar zu einer Gefährdung oder zu schwerwiegenden Auswirkungen führt, wird die Analyse ausgeführt, um die Ursache dieses Ereignisses zu identifizieren. Dies wird in mehreren Schritten unter Anwendung von logischen Operatoren (AND, OR usw.) ausgeführt. Zwischenursachen werden genauso analysiert und dann auf ein Basisereignis zurückgeführt, womit die Analyse beendet wird.

Es handelt sich um ein graphisches Verfahren und zum Zeichnen des Fehlerbaums steht eine Anzahl genannter Symbole zur Verfügung. Der Fehlerbaum stellt am Ende der Analyse die logischen Funktionen dar, indem die Basisereignisse (im Allgemeinen die Bauteilausfälle) mit dem Ausgangsereignis (der Ausfall des Gesamtsystems) verbunden werden. Das Verfahren ist hauptsächlich für die Analyse von Hardwaresystemen vorgesehen, aber es hat bereits Versuche gegeben, diesen Ansatz auch für die Analyse von Softwareausfällen anzuwenden. Dieses Verfahren kann qualitativ zur Ausfall-Analyse (Identifizierung von Ausfallszenarien: minimale Schnittmengen oder primäre Implikanten), semi-quantitativ (indem die Szenarien gemäß ihren Wahrscheinlichkeiten eingeordnet werden) und quantitativ für probabilistische Berechnungen des Ausgangsereignisses (siehe C.6) verwendet werden.

Literaturhinweise:

IEC 61025:2006, *Fault Tree Analysis (FTA)*

From safety analysis to software requirements. K.M. Hansen, A.P. Ravn, A.P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998

B.6.6.6 Markov-Modelle

ANMERKUNG Siehe B.1 der IEC 61508-6 zur Anwendung dieses Verfahrens im Gegensatz zu Zuverlässigkeitsblockdiagrammen, im Zusammenhang mit der Analyse der Sicherheitsintegrität der Hardware.

Ziel: Darstellung des Systemverhaltens mittels eines Zustandübergangsgraphen. Bewertung der probabilistischen Parameter (z. B. Unzuverlässigkeit, Nichtverfügbarkeit, *MTTF*, *MUT*, *MDT* usw.) eines Systems.

Beschreibung: Es ist ein endlicher Zustandsautomat (siehe B.2.3.2) dargestellt durch einen geleiteten Graphen. Die Knoten (Kreise) stellen die Zustände dar und die Kanten (Pfeile) zwischen den Knoten stellen die Übergänge (Ausfall, Reparaturen usw.) dar, die zwischen den Zuständen auftreten. Die Kanten werden mit den entsprechenden Ausfallraten oder Reparaturraten gewichtet. Die grundsätzliche Eigenschaft von homogene Markov-Prozessen ist es, dass die Zukunft nur von der Gegenwart abhängt: ein Wechsel vom Zustand *N* zu einem darauf folgenden Zustand *N*+1 ist vom vorherigen Zustand *N*-1 unabhängig. Dies deutet an, dass alle probabilistischen Gesetze der Modelle exponentiell sind.

Die Ausfallereignisse, -zustände und -raten können so detailliert werden, dass eine genaue Beschreibung des Systems erreicht wird, zum Beispiel über erkannte oder unerkannte Ausfälle, Offenbarung eines größeren Ausfalls usw. Indem die sogenannten mehrphasigen Markov-Prozesse verwendend werden, können auch die Intervalle der Wiederholungsprüfungen genau hergeleitet werden, wobei die Wahrscheinlichkeiten der Zustände am Ende einer Phase (z. B. kurz vor einer Wiederholungsprüfung) verwendet werden können, um die anfänglichen Bedingungen für die folgende Phase zu berechnen (z. B. die Wahrscheinlichkeiten der verschiedenen Zustände, nachdem eine Wiederholungsprüfung durchgeführt worden ist).

Das Verfahren nach Markov ist zur Modellierung redundanter Systeme geeignet, in denen sich der Grad der Redundanz in zeitlicher Abhängigkeit von Bauteilausfällen und Reparaturen ändert. Andere klassische Methoden, zum Beispiel FMEA und FTA, können nicht ohne Weiteres zum Zwecke der Modellierung der Ausfälle im Systemlebenszyklus angepasst werden, da keine einfachen kombinatorischen Gleichungen zur Berechnung der zugehörigen Wahrscheinlichkeiten existieren.

In einfachen Fällen sind die Gleichungen, die die Wahrscheinlichkeiten des Systems beschreiben, ohne Weiteres in der Literatur verfügbar oder können manuell aufgestellt werden. Es existieren ebenfalls einige Methoden der Vereinfachung (d. h. Reduzierung der Anzahl der Zustände), um komplexere Fälle zu behandeln.

Dennoch ist, mathematisch gesprochen, ein homogener Markov-Graph nur eine einfache und übliche Menge von linearen Differenzialgleichungen mit konstanten Koeffizienten. Dies ist seit Langem analysiert worden und wirkungsvolle Algorithmen sind entwickelt worden und sind verfügbar, um sie zu behandeln. Wenn die Größe des Modells zunimmt, ist es deshalb sehr effizient, die oben genannten Algorithmen zu verwenden, die in verschiedenen Softwareanwendungen realisiert sind.

Es muss angemerkt werden, dass die Größe des Graphen exponentiell mit der Anzahl der Komponenten zunimmt: dies ist die sogenannte kombinatorische Explosion. Dieses Verfahren ist deshalb ohne vereinfachende Näherungen nur für kleine Systeme anwendbar.

Wenn nicht exponentielle Gesetze (Semi-Markov-Modelle) behandelt werden müssen, dann sollte die Monte-Carlo-Simulation (siehe B.6.6.8) angewendet werden.

Literaturhinweise:

IEC 61165:2006, *Application of Markov techniques*

The Theory of Stochastic Processes. R. E. Cox and H. D. Miller, Methuen and Co. Ltd., London, UK, 1963

Finite MARKOV Chains. J. G. Kemeny and J. L. Snell, D. Van Nostrand Company Inc, Princeton, 1959

The Theory and Practice of Reliable System Design. D. P. Siewiorek and R. S. Swarz, Digital Press, 1982

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.7 Zuverlässigkeitsblockdiagramm (en: reliability block diagrams, RBD)

ANMERKUNG 1 Dieses Verfahren/Maßnahme wird im Anhang B der IEC 61508-6 verwendet.

ANMERKUNG 2 Siehe auch C.6.4 „Zuverlässigkeitsblockdiagramm“.

Ziel: Graphische Darstellung einer Menge von Ereignissen, die stattfinden müssen, und Bedingungen, die erfüllt werden müssen, damit das System oder eine Anwendung erfolgreich betrieben wird. Es ist eher eine darstellende als eine analytische Methode.

Beschreibung: Das Objekt der Analyse wird als ein Erfolgspfad, bestehend aus Blöcken, Linien und logischen Verknüpfungen, dargestellt. Ein Erfolgspfad beginnt an einer Seite des Diagramms und setzt sich über die Blöcke und Verzweigungen bis zur anderen Seite des Diagramms fort. Ein Block stellt eine Bedingung oder ein Ereignis dar, und der Pfad kann ihn durchlaufen, wenn die Bedingung wahr ist oder das Ereignis stattgefunden hat. Kommt der Pfad zu einer Verzweigung, wird er fortgesetzt, wenn die Logik an der Verzweigung erfüllt ist. Wenn er einen Scheitelpunkt erreicht, kann eine Fortsetzung entlang aller abgehenden Linien erfolgen. Wenn mindestens ein Erfolgspfad durch das Diagramm existiert, wird das Objekt der Analyse ordnungsgemäß betrieben.

Ein RBD ist eine strukturelle Darstellung des entwickelten Systems. Es ist wie eine Art eines elektrischen Schaltkreises. Wenn der Strom einen Pfad vom Eingang zum Ausgang findet, bedeutet dies, dass das entwickelte System richtig arbeitet. Wenn der Schaltkreis unterbrochen ist, bedeutet dies, dass das entwickelte System ausgefallen ist. Dies führt zum Konzept der minimalen Schnittmengen, das die Kombinationen der Ausfälle darstellt (d. h. Stellen, wo das RBD „unterbrochen“ ist), die zum Ausfall des entwickelten Systems führen.

Mathematisch ist ein RBD einem Fehlerbaum ähnlich. Es stellt die logische Funktion dar, die die Zustände der einzelnen Komponenten (ausgefallen oder arbeitend) mit dem Zustand des Gesamtsystems (ausgefallen oder arbeitend) verbindet. Deshalb sind die Berechnungen mit denen ähnlich, die für Fehlerbäume beschrieben sind.

Literaturhinweise:

IEC 61078:2006, *Analysis techniques for dependability – Reliability block diagram method and boolean methods*

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.8 Monte-Carlo-Simulation

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.4 der IEC 61508-3 angeführt und wird im Anhang B der IEC 61508-6 verwendet.

Ziel: Simulation realer Phänomene durch Generierung zufälliger Zahlen, wenn analytische Methoden nicht anwendbar sind.

Beschreibung: Monte-Carlo-Simulationen werden verwendet, um zwei Klassen von Problemen zu lösen:

– probabilistische Probleme, wobei zufällige Zahlen verwendet werden, um stochastische Phänomene zu erzeugen, und

– deterministische Probleme, die mathematisch in ein gleichwertiges probabilistisches Problem übersetzt werden (z. B. Integralrechnung).

Das Prinzip der Monte-Carlo-Simulation ist es, Zufallszahlen zu verwenden, um ein funktionales und ein dysfunktionales Verhaltensmodell des zu betrachtenden Systems zu animieren. Solche Verhaltensmodelle werden durch Zustandübergangsmodelle (Markov-Graphen, Petri-Netze, formale Sprachen usw.) erstellt. Die Monte-Carlo-Simulation wird durchgeführt, um eine große Stichprobe zu erhalten, aus der statistische Ergebnisse ermittelt werden können.

Bei Anwendung der Monte-Carlo-Simulation muss darauf geachtet werden, dass die hinzugefügten Eingangsdaten, die Toleranzen oder das Rauschen vernünftigen Werten entsprechen. Dies muss durch das Vertrauensintervall erreicht werden, das leicht aus den Simulationen ermittelt werden kann. Entgegen analytischen Methoden ist die Monte-Carlo-Simulation selbst abschätzend. Unwesentliche Ereignisse erscheinen nicht ohne den Bedarf, sie zu identifizieren, um das Modell zu vereinfachen.

Ein allgemeines Prinzip der Monte-Carlo-Simulation ist es, ein Problem neu darzulegen und neu zu formulieren, so dass die erhaltenen Ergebnisse so genau wie möglich werden, anstatt das Problem wie ursprünglich festgelegt anzugehen.

Im Zusammenhang dieser Norm darf die Monte-Carlo-Simulation für Berechnungen des SIL verwendet werden und um die Unklarheiten der Zuverlässigkeitsdaten zu berücksichtigen. Mit gegenwärtigen Rechnern werden SIL-4-Berechnungen leicht ausgeführt.

Literaturhinweise:

Monte Carlo Methods. J. M. Hammersley, D. C. Handscomb, Chapman & Hall, 1979

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.9 Fehlerbaummodelle

ANMERKUNG 1 Siehe IEC 61508-6 für die Anwendung dieses Verfahrens zur Analyse der Sicherheitsintegrität der Hardware.

ANMERKUNG 2 Fehlerbäume sind bereits oben als ein Mittel zur Validierung der Sicherheit beschrieben worden (siehe B.6.6.5). Sie sind auch für die Ausfallanalyse und probabilistische Berechnungen weit verbreitet.

Ziel: Erstellung der logischen Funktionen durch einen systematischen und grafischen Top-Down-Ansatz (Wirkung-Ursache), der die Basisereignisse (Ausfallarten) mit dem Ausgangsereignis (unerwünschtes Ereignis) verbindet.

Beschreibung: Dies ist sowohl eine Analyse-Methode, die dem Analytiker hilft, das Modell Schritt für Schritt zu entwickeln, als auch ein mathematisches Modell für probabilistische Berechnungen. Sie erlaubt die Durchführung der:

– qualitativen Analyse, indem Ausfallszenarien identifiziert und klassifiziert werden (minimale Schnittmengen oder primäre Implikanten),

– semi-quantitativen Analyse, indem Szenarien gemäß ihren Eintrittswahrscheinlichkeiten eingeordnet werden,

– quantitativen Analyse, indem die Wahrscheinlichkeit des Ausgangsereignisses berechnet wird.

Wie beim Zuverlässigkeitsblockdiagramm (en: reliability block diagram, RBD) stellt ein Fehlerbaum die logische (boolesche) Funktion dar, die die Zustände der einzelnen Komponenten (ausgefallen oder arbeitend) mit dem Zustand des Gesamtsystems (ausgefallen oder arbeitend) verbindet. Wenn die Komponenten unabhängig sind, dürfen deshalb probabilistische Berechnungen durchgeführt werden, indem die grundlegenden Eigenschaften der Wahrscheinlichkeiten einfach auf die logische Funktion angewendet werden. Dies ist nicht so einfach, weil dies ein statisches Modell ist, das grundsätzlich nur mit echten (d. h. konstanten) Wahrscheinlichkeiten arbeitet. Zeitabhängige Wahrscheinlichkeiten müssen vorsichtig behandelt werden. Zum Beispiel kann die PFD_{avg} von Sicherheitssystemen, die Bauteile enthalten, die regelmäßig einer Wiederholungsprüfung unterzogen werden, nicht einfach berechnet werden. Dies ist für die *PFH* von Sicherheitssystemen, die in der Betriebsart mit kontinuierlicher Anforderung arbeiten, noch schwieriger. Deshalb sollten nur Sicherheitsingenieure mit einem einwandfreien Verständnis der zugrunde liegenden Mathematik Berechnungen der Nichtverfügbarkeit/*PFD* und der Unzuverlässigkeit/*PFH* mit dieser Methode durchführen.

Die Berechnungen können für sehr einfache Fehlerbäume von Hand durchgeführt werden, aber viele Algorithmen sind im Laufe der letzten 50 letzten Jahre entwickelt und umgesetzt worden, um komplexe logische Gleichungen zu bewältigen. Der Stand der Technik ist gegenwärtig die Verwendung von binären Entschei-

dungsdiagrammen (en: binary decision diagram, BDD). Dies ist ein Verfahren der kompakten Codierung der logischen Gleichung in den Speicher des Rechners. Sie ist gegenwärtig die einzige Methode, die fähig ist, die probabilistischen Berechnungen ohne Abschätzungen für Systeme im industriellen Format durchzuführen. Sie ist ebenso ausreichend schnell, um die Behandlung von Unklarheiten mit der Monte-Carlo-Simulation zu ermöglichen.

Literaturhinweis:

IEC 61025:2006, *Fault Tree Analysis (FTA)*

B.6.6.10 Verallgemeinertes stochastisches Petri-Netz (en: generalised stochastic petri net, GSPN)

ANMERKUNG 1 Siehe IEC 61508-6 für die Anwendung dieses Verfahrens zur Analyse der Sicherheitsintegrität der Hardware.

ANMERKUNG 2 Petri-Netze sind bereits als semi-formale Methode genannt worden (siehe B.2.3.3). Sie können ebenfalls effizient im Zusammenhang mit der Sicherheitsintegrität der Hardware verwendet werden.

Ziel: Graphische Erstellung eines funktionalen und dysfunktionalen Modells, das sich so real wie möglich wie das tatsächlich entwickelte System verhält, um eine effiziente Unterstützung für die Monte-Carlo-Simulation bereitzustellen.

Beschreibung: Dies ist ein asynchroner endlicher Zustandsautomat, wie in B.2.3.3 beschrieben, außer dass die gültige Eigenschaft, die bei der Ausführung der semi-formalen Validierung ermittelt wurde, nicht mehr vorhanden ist, wenn das dysfunktionale Verhalten eines Sicherheitssystems modelliert wird. Die sogenannten Stellen (dargestellt durch Kreise) stellen die möglichen Zustände dar und die sogenannten Transitionen (dargestellt durch Rechtecke) stellen die Ereignisse dar, die sich wahrscheinlich ereignen werden. Zusätzlich zu der Markierung der Stellen (siehe B.2.3.3) können Botschaften oder Aussagen verwendet werden, um die Transitionen gültig zu machen (zu ermöglichen). Die Verzögerung, die von der Ermöglichung einer Transition bis zu ihrer Abfeuerung vergeht, kann deterministisch oder stochastisch sein. Das ist der Grund, warum Petri-Netze „verallgemeinerte stochastische“ Petri-Netze genannt werden.

Petri-Netze erzeugen flexible Verhaltensmodelle, die sich als sehr effizient zur Unterstützung der Monte-Carlo-Simulierung erweisen (siehe B.6.6.8). Außer der Genauigkeit der Monte-Carlo-Simulierung selbst, die ohnehin immer bekannt ist, werden alle Beschränkungen anderer Methoden (Abhängigkeiten, kombinatorische Explosion, nicht-exponentielle Gesetze usw.) überwunden. Mit gegenwärtigen Rechnern ist dies sogar für SIL-4-Beurteilungen kein Problem mehr.

Literaturhinweise:

IEC 62551, *Analysis techniques for dependability – Petri net modelling (CD1)* [3)]

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.7 Worst-Case-Analyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Vermeidung systematischer Ausfälle, die durch ungünstige Kombinationen der Umgebungsbedingungen und Bauteiltoleranzen entstehen.

Beschreibung: Die Betriebsmöglichkeiten und die Bauteildimensionierung werden auf theoretischer Basis geprüft oder berechnet. Die Umgebungsbedingungen werden zu ihren höchsten zulässigen Grenzwerten hin verändert. Die entscheidenden Reaktionen des Systems werden überprüft und mit der Spezifikation verglichen.

[3)] In Diskussion.

B.6.8 Erweiterte Funktionstests

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Aufdeckung von Ausfällen während der Spezifikations-, Entwurfs- und Entwicklungsphase. Test des Verhaltens des sicherheitsbezogenen Systems im Falle seltener oder nicht festgelegter Eingaben.

Beschreibung: Die erweiterte Funktionsprüfung überprüft das funktionale Verhalten des sicherheitsbezogenen Systems als Reaktion auf Eingabebedingungen, die nur selten erwartet werden (zum Beispiel mehrfacher Ausfall) oder die außerhalb der Spezifikation des sicherheitsbezogenen Systems liegen (zum Beispiel nicht ordnungsgemäßer Betrieb). Für seltene Bedingungen wird das beobachtete Verhalten des sicherheitsbezogenen Systems mit der Spezifikation verglichen. Wenn die Reaktion des sicherheitsbezogenen Systems nicht festgelegt ist, sollte getestet werden, ob die Sicherheit der Anlage bei der beobachteten Reaktion erhalten bleibt.

Literaturhinweise:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Dependability of Critical Computer Systems 3. P. G. Bishop et al,. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.9 Test unter Grenzbedingungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Durchführung der Testfälle, die während der Worst-Case-Analyse festgelegt wurden.

Beschreibung: Die Betriebsmöglichkeiten und die Bauteildimensionierung werden unter Grenzbedingungen („worst case“) getestet. Die Umgebungsbedingungen werden zu ihren höchsten zulässigen Grenzwerten hin verändert. Die entscheidenden Reaktionen des Systems werden überprüft und mit der Spezifikation verglichen.

B.6.10 Test durch Fehlereinbau

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.6 der IEC 61508-2 angeführt.

Ziel: Einbringung oder Simulation von Fehlern in die Systemhardware und Dokumentation der Reaktion.

Beschreibung: Dies ist eine qualitative Methode zur Beurteilung der Zuverlässigkeit. Vorzugsweise werden detaillierte Blockschaltbilder, Schaltpläne und Verdrahtungspläne verwendet, um den Ort, den Typ und die Art der Einbringung eines Fehlers zu beschreiben; zum Beispiel: die Energieversorgung kann von verschiedenen Modulen abgetrennt werden; Energie, Bus- oder Adressierungsleitungen können aufgetrennt oder kurzgeschlossen werden; Bauteile oder deren Anschlüsse können aufgetrennt oder kurzgeschlossen werden; Relais können ausfallen, so dass sie nicht anziehen oder abfallen oder dies zu einem falschen Zeitpunkt tun usw. Die sich ergebenden Systemausfälle werden klassifiziert, wie zum Beispiel in den Tabellen 1 und 2 der IEC 60812. Prinzipiell werden einzelne Fehler dauerhaft eingebracht. In dem Fall jedoch, dass ein Fehler nicht durch die integrierten Selbsttests aufgedeckt oder auf andere Weise offensichtlich wird, kann dieser im System belassen und die Auswirkung eines zweiten Fehlers betrachtet werden. Die Anzahl der Fehler kann sich leicht auf mehrere Hundert erhöhen.

Die Arbeit wird von einer interdisziplinären Arbeitsgruppe ausgeführt. Der Hersteller des Systems sollte anwesend sein und befragt werden. Die MTBF für Fehler, die ernsthafte Auswirkungen haben, sollten berechnet oder abgeschätzt werden. Ist die berechnete Zeit zu gering, sollten Modifikationen erfolgen.

Literaturhinweise:

IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

IEC 61069-5:1994, *Industrial-process measurement and control – Evaluation of system properties for the purpose of system assessment – Part 5: Assessment of system dependability*

Anhang C
(informativ)

Überblick über Verfahren und Maßnahmen zum Erreichen der Sicherheitsintegrität der Software (siehe IEC 61508-3)

C.1 Allgemeines

Der in diesem Anhang enthaltene Überblick über Verfahren sollte weder als vollständig noch als ausschließlich betrachtet werden.

C.2 Anforderungen und detaillierter Entwurf

ANMERKUNG Zutreffende Verfahren und Maßnahmen befinden sich ebenfalls in B.2.

C.2.1 Strukturierte schematische Methoden

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.2 und A.4 der IEC 61508-3 angeführt.

C.2.1.1 Allgemeines

Ziel: Das Hauptziel der strukturierten Methoden ist es, die Qualität der Softwareentwicklung durch besondere Aufmerksamkeit auf frühe Teile des Lebenszyklus zu fördern. Die Methoden zielen darauf ab, dieses sowohl durch genaue als auch intuitive Verfahren und (rechnerunterstützte) Notationen zu erreichen und Anforderungen und Ausführungsmerkmale in einer logischen Ordnung und einer strukturierten Art und Weise zu bestimmen und zu dokumentieren.

Beschreibung: Es existieren mehrere strukturierte Methoden. Einige sind für traditionelle Datenverarbeitungsfunktionen entworfen worden, während andere mehr in Richtung Prozesssteuerungs- und Echtzeitanwendungen ausgerichtet sind (welche eher dazu neigen, sicherheitskritisch zu sein). UML (siehe C.3.12) enthält viele Beispiele von strukturierten Notationen.

Strukturierte Methoden sind im Wesentlichen „gedankliche Werkzeuge“ zur systematischen Wahrnehmung und Unterteilung eines Problems oder Systems. Ihre Hauptmerkmale sind die folgenden:

– eine logische Gedankenfolge zum Herunterbrechen eines großen Problems in handhabbare Teile;
– Analyse und Dokumentation des Gesamtsystems einschließlich sowohl der Umgebung als auch des erforderlichen Systems;
– Zerlegung der Daten und Funktionen in dem erforderlichen System;
– Checklisten, d. h. Auflistung der Art der Gegenstände, die einer Analyse bedürfen;
– geringer intellektueller Überhang – einfach, intuitiv, pragmatisch;
– häufig mit einer starken Gewichtung auf die Entwicklung eines schematischen Modells des beabsichtigten Systems und CASE-Tool-Unterstützung für die Gesamtmethode.

Die unterstützenden Notationen zur Analyse und Dokumentation der Probleme und Systemeinheiten (zum Beispiel Prozess- und Datenflüsse) neigen dazu, genau zu sein, aber Notationen zur Beschreibung der Prozessfunktionen, die durch diese Einheiten ausgeführt werden, neigen eher dazu, informell zu sein. Einige Methoden verwenden jedoch teilweise formale (mathematische) Notationen (zum Beispiel reguläre Beschreibungen oder endliche Zustandsmaschinen). Eine erhöhte Genauigkeit reduziert nicht nur den Spielraum für Missverständnisse, sondern liefert Raum zur automatischen Verarbeitung.

Ein anderer Vorteil der strukturierten Notation ist ihre Transparenz, so dass eine Spezifikation oder ein Entwurf durch den Anwender intuitiv mit Hilfe seiner nicht formal niedergelegten Erfahrung getestet werden kann.

Dieser Überblick beschreibt mehrere strukturierte Methoden ausführlicher.

Literaturhinweise:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192

C.2.1.2 Kontrollierte Anforderungsbeschreibung (en: Controlled Requirements Expression, CORE)

Ziel: Sicherstellung, dass alle Anforderungen bestimmt und beschrieben sind.

Beschreibung: Dieser Ansatz ist dafür vorgesehen, die Lücke zwischen dem Kunden/Endanwender und dem Analytiker zu schließen. Er ist nicht streng mathematisch, sondern unterstützt die Kommunikation. CORE wurde entworfen, um Anforderungen auszudrücken und nicht eine Spezifikation zu erstellen. Der Ansatz ist strukturiert und die Beschreibung umfasst verschiedene Grade der Verfeinerungen. Die Methode CORE ermöglicht eine breitere Sichtweise des Problems unter Einbeziehung der Umgebung, in der das System verwendet wird, sowie eine differenzierte Sichtweise der verschiedenen Anwendergruppen. CORE enthält Richtlinien und Vorgehensweisen zur Erkennung von Abweichungen vom Entwurf. Abweichungen können korrigiert oder explizit identifiziert und dokumentiert werden. Deswegen können Spezifikationen unvollständig sein, aber ungelöste Probleme und Bereiche mit hohem Risiko werden erkannt und müssen im anschließenden Entwurf behandelt werden.

Literaturhinweise:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.2.1.3 Jackson Systementwicklung (en: Jackson System Development, JSD)

Ziel: Eine Entwicklungsmethode, die die Entwicklung von Softwaresystemen von den Anforderungen bis hin zur Codierung abdeckt, mit speziellem Bezug zu Echtzeitsystemen.

Beschreibung: JSD ist ein abgestuftes Entwicklungsverfahren, in dem der Entwickler die realen Prozesse, auf denen die Systemfunktionen basieren, mit Hilfe eines Modells darstellt, die erforderlichen Funktionen bestimmt, diese in das Modell einfügt und die resultierende Spezifikation in eine in der Zielumgebung realisierbare umwandelt. Deshalb betrifft es die traditionellen Phasen der Spezifikation, des Entwurfs und der Implementierung, aber nimmt im Gegensatz zu den traditionellen Methoden eine etwas unterschiedliche Sichtweise ein, da sie nicht „top-down“ abläuft.

Außerdem legt es besonderes Gewicht auf die frühe Identifikation der Einheiten der realen Prozesse, welche das zu erstellende System betreffen, sowie auf ein Modell der Einheiten und was mit ihnen geschieht. Ist diese Analyse der realen Prozesse abgeschlossen und ist ein Modell erstellt worden, werden die vom System geforderten Funktionen analysiert, um zu bestimmen, wie sie sich in dieses Modell der realen Prozesse einfügen können. Die resultierende Systemgestaltung wächst mit den strukturierten Beschreibungen aller Prozesse in dem Modell. Anschließend wird alles in Programme übertragen, die mit der Zielsoftware und mit der Hardwareumgebung zusammenarbeiten.

Literaturhinweise:

Systems Analysis and Design. D. Yeates, A. Wakefield. Pearson Education, 2003, ISBN 0273655361, 9780273655367

An Overview of JSD. J. R. Cameron. IEEE Transactions on Software Engineering, SE-12, No. 2, February 1986

C.2.1.4 Echtzeit-Yourdon

Ziel: Die Spezifikation und der Entwurf von Echtzeitsystemen.

Beschreibung: Das diesem Verfahren zugrunde liegende Entwicklungsschema nimmt eine Drei-Phasenentwicklung des zu entwickelnden Systems an. Die erste Phase beinhaltet die Erstellung eines Basismodells, welches das erforderliche Systemverhalten beschreibt. Die zweite Phase beinhaltet die Erstellung eines Implementierungsmodells, das die Strukturen und Mechanismen beschreibt, die nach der Implementierung das erforderliche Verhalten verkörpern. Die dritte Phase beinhaltet die aktuelle Systemverwirklichung in Hardware und Software. Die drei Phasen entsprechen ungefähr den traditionellen Phasen der Spezifikation, des Entwurfs und der Entwicklung, aber legen größeren Wert auf die Tatsache, dass der Entwickler in jeder Phase in eine Modellbildung eingebunden ist.

Das wesentliche Modell besteht aus zwei Teilen:

– dem Umgebungsmodell, welches eine Beschreibung der Grenzen zwischen dem System und dessen Umgebung und eine Beschreibung der externen Ereignisse, auf die das System reagieren muss, beinhaltet, und
– dem Verhaltensmodell, welches Schemata beinhaltet, die die vom System als Antwort auf Ereignisse auszuführenden Umformungen und die vom System zu speichernden Daten beschreiben.

Das Ausführungsmodell teilt sich ebenfalls in Untermodelle auf, die die Zuordnung von einzelnen Prozessen zu Zentraleinheiten und die Verteilung der Prozesse auf Softwaremodule abdecken.

Um diese Modelle zu erfassen, kombiniert dieses Verfahren einige bekannte Verfahren: Datenflussdiagramme, Umformungsgraphen, strukturierte natürliche Sprache, Zustands-Übergangsdiagramme und Petri-Netze. Zusätzlich beinhaltet diese Methode Verfahren zur Simulation eines vorgeschlagenen Systementwurfs entweder papier- oder rechnergestützt mit Hilfe der angefertigten Modelle.

Literaturhinweise:

Real-time Systems Development. R. Williams. Butterworth-Heinemann, 2006, ISBN 0750664711, 9780750664714

Structured Development for Real-Time Systems (3 Bände). P. T. Ward and S. J. Mellor. Yourdon Press, 1985

C.2.2 Datenflussdiagramme

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.5 und B.7 der IEC 61508-3 angeführt.

Ziel: Beschreibung des Datenflusses innerhalb eines Programms in Diagrammform.

Beschreibung: Datenflussdiagramme dokumentieren, wie Eingabedaten zu Ausgabedaten umgeformt werden, wobei jede Stufe des Diagramms eine einzelne Umformung darstellt.

Datenflussdiagramme haben drei Aspekte:

– kommentierte Pfeile – sie stellen den Datenfluss zu und von Umformungszentren dar und sind mit einem Kommentar versehen, der angibt, um welche Daten es sich handelt;
– kommentierte Blasen – sie stellen Umformungszentren dar mit Kommentaren, die die Umformungen beschreiben;
– Operatoren (AND, XOR) – diese Operatoren werden verwendet, um die kommentierten Pfeile zu verbinden.

Jede Blase in einem Datenflussdiagramm kann als eine eigenständige Black-Box betrachtet werden, die, sobald Eingaben zur Verfügung stehen, diese zu Ausgaben umformt. Einer ihrer Hauptvorteile ist, dass sie Umformungen ohne irgendwelche Annahmen über ihre Verwirklichung zeigt. Ein reines Datenflussdiagramm enthält keine Steuerungs- oder Abfolgeinformationen, dafür sorgen Echtzeiterweiterungen dieser Notation wie bei Echtzeit-Yourdon (siehe C.2.1.4).

Die Erstellung von Datenflussdiagrammen erfolgt am besten durch die Betrachtung von Systemeingaben und das Durcharbeiten in Richtung der Systemausgaben. Jede Blase muss eine klare Umformung darstellen – ihre Ausgabe sollte sich von ihrer Eingabe unterscheiden. Es gibt keine Regeln für die Festlegung der Gesamtstruktur der Diagramme, ihre Erstellung ist einer der kreativen Gesichtspunkte des Systementwurfs. Wie jeder Entwurf ist dies ein iterativer Prozess, wobei die ersten Versuche schrittweise bis zum endgültigen Diagramm verbessert werden.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

ISO 5807:1985, *Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts*

ISO/IEC 8631:1989, *Information technology – Program constructs and conventions for their representation*

C.2.3 Strukturdiagramme

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.5 der IEC 61508-3 angeführt.

Ziel: Darstellung der Programmstruktur in Diagrammform.

Beschreibung: Strukturdiagramme sind eine Notation, die Datenflussdiagramme ergänzt. Sie beschreiben das Programmiersystem und die Hierarchie von Programmteilen und stellen dies in einem Baum graphisch dar. Sie dokumentieren, wie Elemente eines Datenflussdiagramms als eine Hierarchie von Programmeinheiten verwirklicht werden können.

Eine Strukturübersicht zeigt die Beziehungen zwischen den Programmmodulen, ohne irgendwelche Informationen über die Aktivierungsreihenfolge dieser Einheiten zu beinhalten. Sie werden unter Verwendung folgender vier Symbole dargestellt:

– ein Rechteck, versehen mit dem Namen des Moduls;
– eine Linie zur Verbindung der Rechtecke, um Strukturen zu erzeugen;
– ein kreisförmiger Pfeil (leerer Kreis), versehen mit dem Namen der Daten, die zu und von den Elementen in der Strukturübersicht fließen (normalerweise wird der kreisförmige Pfeil neben der Linie dargestellt, die die Rechtecke in der Übersicht verbindet);
– ein kreisförmiger Pfeil (gefüllter Kreis), versehen mit dem Namen der Steuersignale, die in der Strukturübersicht von einem Modul zum nächsten fließen. Auch hier wird der kreisförmige Pfeil neben den Linien dargestellt, die die zwei Module verbinden.

Aus jedem nicht-trivialen Datenflussdiagramm kann eine Anzahl unterschiedlicher Strukturübersichten abgeleitet werden.

Datenflussdiagramme stellen die Beziehungen zwischen Informationen und Funktionen des Systems dar. Strukturdiagramme stellen die Lösung dar, wie Elemente des Systems verwirklicht werden. Beide Verfahren führen zu richtigen, jedoch verschiedenen Systemansichten.

Literaturhinweise:

Software Design & Development. G. Lancaster. Pascal Press, 2001, ISBN 1741251753, 9781741251753

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.2.4 Formale Methoden

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.1, A.2, A.4 und B.5 der IEC 61508-3 angeführt.

C.2.4.1 Allgemeines

Ziel: Mathematisch basierte Softwareentwicklung. Dies beinhaltet einen formalen Entwurf und formale Codierungsverfahren.

Beschreibung: Formale Methoden liefern Mittel zur Entwicklung einer Systembeschreibung an gewissen Stufen ihrer Spezifikation, Entwurf oder Ausführung. Die resultierende Beschreibung ist in einer strengen Notation abgefasst, die einer mathematischen Analyse unterworfen werden kann, um verschiedene Klassen von Unstimmigkeiten oder Unrichtigkeiten aufzudecken. Außerdem kann die Beschreibung in einigen Fällen maschinell mit einer ähnlichen Strenge, mit der ein Compiler die Syntax eines Source-Codes überprüft, analysiert oder animiert werden, um die verschiedenen Verhaltensaspekte des beschriebenen Systems anzuzeigen. Die Animation führt zu zusätzlichem Vertrauen, dass das System sowohl die wirklichen wie auch die formal festgelegten Anforderungen erfüllt, da es die menschliche Wiedererkennung des festgelegten Verhaltens verbessert.

Eine formale Methode wird im Allgemeinen eine Notation (normalerweise wird irgendeine Form diskreter Mathematik verwendet), ein Verfahren zur Herleitung einer Beschreibung in dieser Notation und verschiedene Formen der Analyse zur Nachprüfung einer Beschreibung auf unterschiedliche Eigenschaften der Korrektheit anbieten.

Mehrere formale Methoden werden in den folgenden Unterabschnitten dieser Übersicht beschrieben: CCS, CSP, HOL, LOTOS, OBJ, temporäre Logik, VDM und Z. Man beachte, dass Verfahren wie endliche Zustandsmaschine und Petri-Netze (siehe Anhang B) als formale Methoden angesehen werden können, je nachdem, wie detailliert diese verwendeten Verfahren einer strengen mathematischen Basis entsprechen.

Literaturhinweise:

Formal Specification: Techniques and Applications. N.Nissanke, Springer-Verlag Telos, 1999, ISBN 1852330023

The Practice of Formal Methods in Safety-Critical Systems. S. Liu, V. Stavridou, B. Dutertre, J. Systems Software, 28, 77-87, Elsevier, 1995

Formal Methods: Use and Relevance for the Development of Safety-Critical Systems. L. M. Barroca, J. A. McDermid, The Computer Journal 35 (6), 579-599, 1992

How to Produce Correct Software – An Introduction to Formal Specification and Program Development by Transformations. E. A. Boiten et al, The Computer Journal 35 (6), 547-554, 1992

C.2.4.2 Berechnung von kommunizierenden Systemen (en: Calculus of Communicating Systems, CCS)

Ziel: CCS ist ein Mittel zur Beschreibung und Diskussion des Verhaltens von Systemen mit gleichzeitig ablaufenden und miteinander kommunizierenden Prozessen.

Beschreibung: CCS ist eine mathematische Vorgehensweise, die sich auf das Systemverhalten bezieht. Der Systementwurf wird mit Hilfe eines Modells, bestehend aus einem Netzwerk aus unabhängigen Prozessen, die aufeinander folgend oder parallel arbeiten, dargestellt. Prozesse können über Schnittstellen (ähnlich zu CSP-Kanälen) kommunizieren. Die Kommunikation findet nur statt, wenn beide Prozesse dazu bereit sind. Unbestimmtes Verhalten (Nicht-Determiniertheit) kann dabei dargestellt werden. Beginnend bei einer Beschreibung des Gesamtsystems mit hohem Abstraktionsgrad ist es möglich, eine schrittweise Verfeinerung des Systems bis zu einer Zusammenstellung von kommunizierenden Prozessen auszuführen, deren Ge-

samtverhalten so ist, wie es vom Gesamtsystem gefordert wird. Ebenso ist es möglich, in einem „Bottom-up"-Ansatz zu arbeiten, in dem Prozesse zusammengeführt und die Eigenschaften des resultierenden Systems unter ähnlichen Regeln, wie sie bei der Verfeinerung verwendet wurden, abgeleitet werden.

Literaturhinweis:

Communication and Concurrency. R. Milner. Pearson Education, 1989, ISBN 9780131150072

C.2.4.3 Sequentielle Kommunikationsprozesse (en: Communicating Sequential Processes, CSP)

Ziel: CSP ist ein Verfahren zur Spezifikation von Softwaresystemen, in denen die Prozesse miteinander kommunizieren und gleichzeitig ablaufen.

Beschreibung: CSP stellt eine Sprache zur Spezifikation von Systemen aus Prozessen bereit sowie die Möglichkeit zur Verifikation, dass die Implementierung der Prozesse ihre Spezifikationen erfüllt (dargestellt als eine Spur (en: trace), d. h. eine zulässige Abfolge von Ereignissen).

Ein System wird mit Hilfe eines Modells, bestehend aus einem Netzwerk aus unabhängigen, aufeinander folgenden oder parallelen Prozessen dargestellt. Jeder Prozess wird durch alle seiner möglichen Verhaltensweisen beschrieben. Prozesse können über Kanäle kommunizieren (synchronisiert oder mit Austausch von Daten). Die Kommunikation findet nur dann statt, wenn beide Prozesse dazu bereit sind. Der Ablauf von Ereignissen kann ebenfalls dargestellt werden.

Die Theorie von CSP wurde direkt in die Architektur des INMOS-Transputers übernommen. Die Sprache OCCAM gestattet einem in CSP beschriebenem System, direkt in einem Transputernetzwerk ausgeführt zu werden.

Literaturhinweis:

Communicating Sequential Processes: The First 25 Years. A. Abdallah, C. Jones, J. Sanders (Eds.). Springer, 2004, ISBN 3540258132, 9783540258131

C.2.4.4 Logik höherer Ordnung (en: High Order Logic, HOL)

Ziel: Dies ist eine formale Sprache, die als Basis zur Hardwarespezifikation und -verifikation vorgesehen ist.

Beschreibung: HOL bezieht sich auf eine besondere Logik-Notation und das dazugehörige maschinelle System. Beides wurde im Rechnerlabor der Universität von Cambridge entwickelt. Die Logik-Notationen sind größtenteils aus der „Simple Theory of Types" von Church entnommen und das maschinelle System basiert auf dem System der Logik von berechenbaren Funktionen (en: Logic of Computable Functions, LCF).

Literaturhinweis:

Higher-Order Computational Logic. J. Lloyd. In Computational Logic: Logic Programming and Beyond, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, ISBN 978-3-540-43959-2

C.2.4.5 LOTOS

Ziel: LOTOS ist ein Mittel zur Beschreibung und Diskussion des Verhaltens von Systemen mit gleichzeitig ablaufenden und miteinander kommunizierenden Prozessen.

Beschreibung: LOTOS (en: Language for Temporal Ordering Specification) basiert auf CCS mit zusätzlichen Merkmalen der Algebra der verwandten Methoden CSP und CIRCAL (en: Circuit Calculus). Es vermeidet die Schwächen von CCS in der Verarbeitung von Datenstrukturen und Wertzuweisungen, indem diese mit Aspekten der abstrakten Datentypsprache ACT ONE kombiniert werden. Der Aspekt der Prozessbeschreibung von LOTOS könnte jedoch mit anderen Formalismen für die Beschreibung abstrakter Datentypen verwendet werden.

Literaturhinweise:

Model Checking for Software Architectures. R. Mateescu. In Software Architecture, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, ISBN 978-3-540-22000-8

ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*

C.2.4.6 OBJ

Ziel: Bereitstellung einer genauen Systemspezifikation mit Rückmeldungen des Anwenders und einer Systemvalidierung vor der Ausführung.

Beschreibung: OBJ ist eine algebraische Spezifikationssprache. Anwender legen die Anforderungen mit algebraischen Gleichungen fest. Die Verhaltensaspekte oder konstruktiven Aspekte des Systems werden durch Befehle festgelegt, die abstrakte Datentypen (ADT) verwenden. Ein ADT ist wie ein ADA-Paket, wobei das Bedienerverhalten sichtbar ist, während die Ausführungsdetails „versteckt“ sind.

Eine OBJ-Spezifikation und die nachfolgende schrittweise Implementierung sind den gleichen formalen Beweistechniken wie andere formale Ansätze zugänglich. Da die konstruktiven Aspekte der OBJ-Spezifikation maschinell ausführbar sind, ist es naheliegend, die Systemvalidierung mit Hilfe der Spezifikationen selbst durchzuführen. Die Ausführung ist im Wesentlichen die Auswertung einer Funktion durch Gleichungssubstitution (en: rewriting), die fortgesetzt wird, bis ein bestimmter Ausgabewert erreicht wird. Diese Ausführbarkeit gestattet es dem Endanwender des vorgestellten Systems, einen „Ausblick“ auf das in Frage kommende System im Stadium der Systemspezifikation zu gewinnen, ohne die Notwendigkeit, mit den zugrunde liegenden formalen Spezifikationsverfahren vertraut sein zu müssen.

Wie bei allen anderen ADT-Verfahren ist OBJ nur auf sequentielle Systeme oder auf sequentielle Aspekte gleichzeitig ablaufender Systeme anwendbar. OBJ ist sowohl für die Spezifikation von kleinen als auch großen industriellen Anwendungen eingesetzt worden.

Literaturhinweis:

Software Engineering with OBJ: Algebraic Specification in Action. J. Goguen, G. Malcolm. Springer, 2000, ISBN 0792377575, 9780792377573

C.2.4.7 Temporäre Logik

Ziel: Der direkte Ausdruck der Sicherheit und der betrieblichen Anforderungen sowie der formale Beweis, dass diese Eigenschaften in den nachfolgenden Entwicklungsschritten eingehalten werden.

Beschreibung: Die übliche Aussagenlogik erster Ordnung beinhaltet keine zeitlichen Vorstellungen. Die temporäre Logik erweitert die Logik erster Ordnung durch Hinzufügen modaler Operatoren (z. B. „von jetzt an“ und „unter Umständen“). Diese Operatoren können verwendet werden, um Aussagen zum System zu qualifizieren. Zum Beispiel könnten Sicherheitseigenschaften „von jetzt an“ erforderlich sein, während andere gewünschte Systemzustände gefordert werden könnten, um „unter Umständen“ bei anderen Startbedingungen erreicht zu werden. Temporäre Formeln werden als Folgen von Zuständen (Verhaltensweisen) interpretiert. Was ein „Zustand“ ist, hängt von der gewählten Stufe der Beschreibung ab. Er kann sich auf das gesamte System, ein Element des Systems oder ein Rechnerprogramm beziehen.

Quantifizierte Zeitintervalle und Einschränkungen werden ausdrücklich nicht in der temporären Logik behandelt. Absolutes Zeitverhalten muss durch die Erzeugung zusätzlicher Zeitzustände im Rahmen der Zustandsbeschreibung behandelt werden.

Literaturhinweis:

Mathematical Logic for Computer Science. M. Ben-Ari. Springer, 2001, ISBN 1852333197, 9781852333195

C.2.4.8 VDM, VDM++ – Wiener Entwicklungsmethode (en: Vienna Development Method)

Ziel: Systematische Spezifikation und Implementierung nacheinander (VDM) und gleichzeitig ablaufender Echtzeit (VDM++)-Programme.

Beschreibung: VDM ist ein mathematisch basiertes Spezifikationsverfahren und ein Verfahren zur Verfeinerung der Implementierung in der Art, dass der Beweis ihrer Korrektheit hinsichtlich der Spezifikation ermöglicht wird.

Das Spezifikationsverfahren beruht auf einem Modell, das den Systemzustand mit Hilfe der Mengenlehre darstellt und Invarianten (Prädikanten) definiert. Die auf ihm definierten Operationen werden durch ihre Vor- und Nachbedingungen dargestellt. Die Operationen können daraufhin überprüft werden, ob sie die Invarianten erhalten.

Die Spezifikation wird implementiert, indem der Systemzustand in Datenstrukturen und die Operationen in ein Programm der Zielsprache abgebildet wird. Es gibt die Möglichkeit, Prüfvorschriften für die Korrektheit dieser Abbildungen festzulegen. Der Entwickler legt fest, ob sie ausgeführt werden oder nicht.

VDM wird hauptsächlich in der Spezifikationsphase verwendet, sie kann aber auch in der Entwurfs- und der zum Source-Code führenden Implementierungsphase verwendet werden. Sie kann nur auf nacheinander ablaufenden Programmen oder nacheinander ablaufenden Prozessen in Systemen mit gleichzeitig ablaufenden Programmen angewendet werden.

Die objektorientierte Echtzeiterweiterung von VDM und VDM++ für gleichzeitig ablaufende Programme ist eine formale Spezifikationssprache, die auf der ISO-Sprache VDM-SL und der objektorientierten Sprache Smalltalk basiert.

VDM++ stellt ein umfassendes Sortiment von Konstrukten bereit, so dass ein Anwender Echtzeitsysteme mit gleichzeitig ablaufenden Programmen in einer objektorientierten Weise formal festlegen kann. In VDM++ besteht eine vollständige formale Spezifikation aus einer Ansammlung von Spezifikationsklassen und optional aus einem Arbeitsbereich.

VDM++ liefert für die Echtzeitentwicklung:

– „temporäre Ausdrücke“ werden bereitgestellt, um den aktuellen Moment und den Moment des Methodenaufrufs innerhalb des Programmteils mit der Methode zu kennzeichnen;
– Ausdrücke mit Eckwerten für einen zeitlichen Ablauf können einer Methode hinzugefügt werden, um die obere (oder untere) Grenze der Ausführungszeit für korrekte Ausführungen festzulegen;
– kontinuierliche Variablen sind eingeführt worden. Mit „Annahme- und Wirkungs-Aussagen“ kann man Zusammenhänge (z. B. Differentialgleichungen) zwischen diesen Funktionen der Zeit festlegen. Diese Eigenschaft hat sich bei der Spezifikation der Anforderungen an Systeme, die in einer kontinuierlichen Umgebung betrieben werden, als sehr nützlich erwiesen. Für diese Art von Systemen führen Verfeinerungsschritte zu diskreten Softwarelösungen.

Literaturhinweise:

ISO/IEC 13817-1:1996, *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*

Systematic Software Development using VDM. C. B. Jones. Prentice-Hall. 2nd Edition, 1990

Conformity Clause for VDM-SL, G. I. Parkin and B. A. Wichmann, Lecture Notes in Computer Science 670, FME‘93 Industrial-Strength Formal Methods, First International Symposium of Formal Methods in Europe. Editors: J. C. P. Woodcock and P. G. Larsen. Springer Verlag, 501-520

C.2.4.9 Z

Ziel: Z ist die Notation einer Spezifikationssprache für sequentielle Systeme und ein Entwurfsverfahren, das es dem Entwickler gestattet, aus einer Z-Spezifikation derart zu ausführbaren Algorithmen zu gelangen, dass der Beweis ihrer Korrektheit gegen die Spezifikation erbracht werden kann.

Z wird hauptsächlich in der Spezifikationsphase verwendet, aber es ist ebenfalls eine Methode entworfen worden, um von der Spezifikation zu einem Entwurf und von dort zu einer Implementierung zu gelangen. Sie ist bestens zur Entwicklung von datenorientierten sequentiellen Systemen geeignet.

Beschreibung: Ebenso wie VDM beruht dieses Spezifikationsverfahren auf einem Modell, das den Systemzustand mit Hilfe der Mengenlehre darstellt und Invarianten (Prädikanten) definiert. Die auf ihm definierten Operationen werden durch ihre Vor- und Nachbedingungen dargestellt. Die Operationen können daraufhin überprüft werden, ob sie die Invarianten erhalten, womit auch die Konsistenz nachgewiesen wird. Der formale Teil einer Spezifikation wird in Schemata unterteilt, die die Strukturierung der Spezifikationen durch Verfeinerung gestatten.

Eine Z-Spezifikation ist typischerweise eine Mischung aus formalem Z und einer informellen Beschreibung in natürlicher Sprache. (Der formale Text selbst kann für die Lesbarkeit zu kurz sein und sein Zweck muss häufig erklärt werden, wohingegen die informelle natürliche Sprache leicht unklar und ungenau werden kann.)

Anders als VDM ist Z eher eine Notation als eine vollständige Methode. Jedoch ist eine verwandte Methode (genannt B) entwickelt worden, die in Verbindung mit Z verwendet werden kann. Die Methode B basiert auf dem Prinzip der schrittweisen Verfeinerung.

Literaturhinweise:

Formal Specification using Z, 2nd Edition. D. Lightfoot. Palgrave Macmillan, 2000, ISBN 9780333763278

The B-Method. S. Schneider. Palgrave Macmillan, 2001, ISBN 9780333792841

C.2.5 Defensive Programmierung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-3 angeführt.

Ziel: Erstellung von Programmen, die anomale Steuerflüsse, Datenflüsse oder Datenwerte während ihrer Ausführung erkennen und auf diese in vorbestimmter und annehmbarer Art und Weise reagieren.

Beschreibung: Während der Programmierung können viele Verfahren verwendet werden, um auf Steuerungs- oder Datenanomalien zu prüfen. Diese können während der Programmierung eines Systems systematisch angewendet werden, um die Wahrscheinlichkeit einer fehlerhaften Datenverarbeitung zu reduzieren.

Es gibt zwei sich überschneidende Bereiche defensiver Verfahren. In sich selbst fehlersichere Software wird entworfen, um den eigenen Entwurfsunzulänglichkeiten entgegenzuwirken. Diese Unzulänglichkeiten können in Irrtümern im Entwurf, in der Codierung oder in den Anforderungen begründet sein. Die folgende Aufzählung zeigt einige der defensiven Verfahren:

– Variablen sollten auf ihren Bereich hin überprüft werden;
– wenn möglich, sollten Werte auf Plausibilität geprüft werden;
– Prozedurparameter sollten bei Einsprung in die Prozedur auf Typ, Dimension und Bereich getestet werden.

Diese ersten drei Empfehlungen helfen sicherzustellen, dass die vom Programm veränderten Zahlen sowohl hinsichtlich der Programmfunktion als auch der physikalischen Bedeutung der Variablen plausibel sind.

Nur-Lese- und Schreib-Lese-Parameter sollten getrennt und ihr Zugriff getestet werden. Funktionen sollten alle Parameter als Nur-Lese-Parameter behandeln, Konstanten nicht beschreibbar sein. Dies hilft, unbeabsichtigtes Überschreiben oder irrtümliches Verwenden von Variablen zu erkennen.

Fehlertolerante Software wird entworfen, um ein Versagen in der eigenen Umgebung oder Abweichungen von nominellen oder erwarteten äußeren Bedingungen zu „erwarten“ und darauf in einer vorbestimmten Art und Weise zu reagieren. Die Verfahren beinhalten das Folgende.

– Eingabevariablen und Zwischenvariablen mit physikalischer Bedeutung sollten auf Plausibilität getestet werden.
– Die Auswirkung von Ausgabevariablen sollte vorzugsweise durch unmittelbare Beobachtung zugehöriger Veränderungen des Systemzustands getestet werden.
– Die Software sollte ihre Konfiguration testen einschließlich der Existenz und der Zugänglichkeit der erwarteten Hardware sowie die Vollständigkeit der Software selbst. Dies ist besonders für die Aufrechterhaltung der Integrität nach Instandhaltungstätigkeiten wichtig.

Einige der defensiven Programmierverfahren, wie der Test des Steuerflusses, sind auch externem Versagen gewachsen.

Literaturhinweise:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Dependability of Critical Computer Systems: Guidelines Produced by the European Workshop on Industrial Computer Systems, Technical Committee 7 (EWICS TC7, Systems Reliability, Safety, and Security). Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811

C.2.6 Entwurfs- und Programmierrichtlinien

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-3 angeführt.

C.2.6.1 Allgemeines

Ziel: Erleichterung der Verifizierbarkeit, Aufforderung zum teamorientierten, objektiven Ansatz und Erzwingung einer vereinheitlichten Entwurfsmethode.

Beschreibung: Die einzuhaltenden Regeln werden zu Beginn des Projektes zwischen den Teilnehmern abgestimmt. Diese Regeln umfassen die zu befolgenden Entwurfs- und Entwicklungsmethoden (zum Beispiel JSP, Petri-Netze usw.) und die zugehörigen Programmierrichtlinien (siehe C.2.6.2).

Diese Regeln werden aufgestellt, um die Entwicklung, Verifikation, Beurteilung und Instandhaltung zu erleichtern. Deshalb sollten sie die verfügbaren Werkzeuge und im Besonderen die Analysatoren und Reverse-Engineering-Werkzeuge berücksichtigen.

Literaturhinweise:

IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

Verein Deutscher Ingenieure. Software-Zuverlässigkeit – Grundlagen, Konstruktive Maßnahmen, Nachweisverfahren. VDI-Verlag 1993. ISBN 3-18-401185-2

C.2.6.2 Programmierrichtlinien

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.1 der IEC 61508-3 angeführt.

Ziel: Reduzierung der Wahrscheinlichkeit von Abweichungen im sicherheitsbezogenen Code und Erleichterung seiner Verifikation.

Beschreibung: Die folgenden Grundsätze zeigen auf, wie sicherheitsbezogene Programmierregeln (für jede Programmiersprache) unterstützen können, den normativen Anforderungen der IEC 61508-3 zu ent-

sprechen und die informativen „wünschenswerten Eigenschaften“ (siehe Anhang F) zu erreichen. Verfügbare unterstützende Werkzeuge sollten berücksichtigt werden.

Anforderungen und Empfehlungen der IEC 61508-3	Vorschläge für Programmierrichtlinien
Modularer Ansatz (Tabelle A.2-7, Tabelle A.4-4)	Begrenzung der Softwaremodulgröße (Tabelle B.9-1) und Lenkung der Softwarekomplexität (Tabelle B.9-2). Beispiele: – Spezifikation der „lokalen“ Größe und Komplexitätsmetriken und Begrenzungen (für Module) – Spezifikation der „globalen“ Komplexitätsmetriken und Begrenzungen (für Gesamtorganisation der Module) – Begrenzung der Anzahl der Parameter/festgelegte Anzahl von Unterprogrammparametern (Tabelle B.9-4) Geheimnisprinzip/Kapselung (Tabelle B.9-3): z. B. Anreize zur Verwendung besonderer Spracheigenschaften. Vollständig definierte Schnittstelle (Tabelle B.9-6). Beispiele: – ausführliche Spezifikation der Funktionssignaturen – Assertion-Programmierung (en: failure assertion programming, Tabelle A.2-3a) und Verifikation der Daten (7.9.2.7), mit ausführlicher Spezifikation der Vorbedingungen und Nachbedingungen für Funktionen, für Assertions und für unveränderliche Datentypen
Verständlichkeit des Codes – Förderung der Verständlichkeit des Codes (7.4.4.13) – lesbar, verständlich und prüfbar (7.4.6)	Vereinbarungen zur Namensgebung, die bedeutungsvolle, eindeutige Namen fördert. Beispiel: Vermeidung von Namen, die verwechselt werden könnten (z. B. IO und I0). Symbolische Namen für numerische Werte. Verfahren und Richtlinien zur Dokumentation des Source-Codes (7.4.4.13). Zum Beispiel: – Erklärung des Grundes und der Bedeutung (und nicht nur des Inhalts) – Widersprüche – Seiteneffekte Sofern möglich, müssen die folgenden Informationen im Source-Code vorhanden sein (7.4.4.13): – juristische Person (z. B. Firma, Autor(en) usw.) – Beschreibung – Ein- und Ausgaben – Historie des Konfigurationsmanagements (Siehe auch Modularer Ansatz)
Verifizierbarkeit und Prüfbarkeit – Erleichterung der Verifikation und des Prüfens (7.4.4.13) – Erleichterung des Erkennens von Entwurfs- oder Programmierfehlern (7.4.4.10) – formale Verifikation (Tabelle A.5-9)	– Schutzmechanismen (en: wrappers) für „kritische“ Bibliotheksfunktionen, um Vor- und Nachbedingungen zu prüfen – Anreize zur Verwendung von Spracheigenschaften, die Einschränkungen beim Gebrauch von besonderen Datenelementen oder Funktionen formulieren können (z. B. const) – für Verifikationen, die mit Werkzeugen unterstützt werden: Regeln, um den Einschränkungen der ausgewählten Werkzeuge zu entsprechen (vorausgesetzt, dies beeinflusst nicht wesentlichere Zielsetzungen)

Anforderungen und Empfehlungen der IEC 61508-3	**Vorschläge für Programmierrichtlinien**
– formaler Beweis (Tabelle A.9-1)	– eingeschränkte Verwendung von Rekursionen (Tabelle B.1-6) und andere Formen von umlaufenden Abhängigkeiten (Siehe auch Modularer Ansatz)
Statische Verifikation der Übereinstimmung mit dem spezifizierten Entwurf (7.9.2.12)	Programmierrichtlinien für die Realisierung von spezifischen Entwurfskonzepten oder -einschränkungen. Zum Beispiel: – Programmierrichtlinien für zyklisches Verhalten mit gesicherter maximaler Zykluszeit (Tabelle A.2-13a) – Programmierrichtlinien für zeitgesteuerte Architekturen (Tabelle A.2-13b) – Programmierrichtlinien für ereignisgesteuerte Architekturen mit gesicherter maximaler Reaktionszeit (Tabelle A.2-13c) – Schleifen mit einer statisch festgelegten maximalen Zahl von Wiederholungen (mit Ausnahme der Endlosschleife des zyklischen Entwurfs) – Programmierrichtlinien für die statische Zuordnung der Ressourcen (Tabelle A.2-14) und Vermeidung von dynamischen Objekten (Tabelle B.1-2) – Programmierrichtlinien für die statische Synchronisation des Zugriffs auf gemeinsame Ressourcen (Tabelle A.2-15) – Programmierrichtlinien zur Entsprechung der eingeschränkten Verwendung von Interrupts (Tabelle B.1-4) – Programmierrichtlinien zur Vermeidung dynamischer Variablen (Tabelle B.1-3a) – Online-Test der Erzeugung von dynamischen Variablen (Tabelle B.1-3b) – Programmierrichtlinien zur Sicherstellung der Kompatibilität mit anderen verwendeten Programmiersprachen (7.4.4.10) Richtlinien zur Erleichterung der Nachvollziehbarkeit mit dem Entwurf
Sprachenteilmenge (Tabelle A.3-3) – Verbot unsicherer Sprachmerkmale (7.4.4.13) – nur Verwendung definierter Sprachmerkmale (7.4.4.10) – strukturierte Programmierung (Tabelle A.4-6) – streng typisierte Programmiersprache (Tabelle A.3-2) – keine automatische Typ-Konvertierung (Tabelle B.1-8)	Ausschluss von Sprachmerkmale, die zu unstrukturierten Entwürfen führen. Z. B. – eingeschränkte Verwendung von Pointern (Tabelle B.1-5) – eingeschränkte Verwendung von Rekursionen (Tabelle B.1-6) – eingeschränkte Verwendung von Unions wie in C – eingeschränkte Verwendung von Exceptions wie in Ada oder C++ – kein unstrukturierter Kontrollfluss in Programmen höherer Programmiersprache (Tabelle B.1-7) – ein Eingang und ein Ausgang für Unterprogramme und Funktionen (Tabelle B.9-5) – keine automatische Typ-Konvertierung – eingeschränkte Verwendung von Seiteneffekten, nicht ersichtlich aus den Funktionssignaturen (z. B. der statischen Variablen). Keine Seiteneffekte bei der Beurteilung der Bedingungen und aller Arten von Erklärungen (en: Assertions). Eingeschränkte oder nur dokumentierte Verwendung von compiler-

Anforderungen und Empfehlungen der IEC 61508-3	Vorschläge für Programmierrichtlinien
	spezifischen Merkmalen. Eingeschränkte Verwendung von eventuell irreführenden Sprachkonstrukten. Anzuwendende Regeln, wenn diese Sprachmerkmale dennoch verwendet werden.
Gute Programmiertechnik (7.4.4.13)	Wenn anwendbar: – Programmierrichtlinien zur Sicherstellung, dass, wenn notwendig, Fließkomma-Ausdrücke in der richtigen Reihenfolge bewertet werden (z. B. ist „a-b+c“ nicht immer gleich „a+c-b“) – Bei Vergleichen mit Fließkommawerten: Verwendung nur der Ungleichheit (weniger als, weniger oder gleich, größer als, größer oder gleich) anstatt der genauen Gleichheit – Richtlinien, die bedingte Kompilation und Preprozessoranweisungen berücksichtigen – Systematisches Prüfen der Rücksprungbedingungen (erfolgreich/fehlerhaft) Dokumentation und, wenn möglich, Automatisierung der Erstellung des ausführbaren Codes (makefiles). Vermeidung von Seiteneffekten, die nicht ersichtlich aus den Funktionssignaturen sind. Wenn solche Seiteneffekte bestehen, Richtlinien zu deren Dokumentation Verwendung von Klammern, wenn die Prioritäten des Programmierers nicht absolut offensichtlich sind. Abfangen von vermutlich auszuschließenden Situationen (z. B. „default“-Fall beim Befehl „switch“ der Programmiersprache C). Verwendung von „Schutzmechanismen“ für kritische Module, um insbesondere Vor- und Nachbedingungen und Rücksprungbedingungen zu überprüfen. Programmierrichtlinien, um bekannten Compiler-Fehlern und den Einschränkungen, ermittelt durch eine Beurteilung des Compilers, zu entsprechen.

C.2.6.3 Keine dynamische Variablen oder dynamische Objekte

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.2 und B.1 der IEC 61508-3 angeführt.

Ziel: Ausschluss von

– unerwünschtem oder unerkanntem Überschreiben des Speichers;

– Ressourcen-Engpässen während der (sicherheitsbezogenen) Laufzeit.

Beschreibung: Im Sinne dieser Maßnahme sind dynamische Variablen und dynamische Objekte solche Variablen und Objekte, denen zur Laufzeit ihr Speicher zugewiesen wird und ihre absoluten Adressen festgelegt werden. Der Wert des zugewiesenen Speichers und dessen Adresse ist abhängig vom Zustand des Systems zum Zeitpunkt der Zuweisung. Dies bedeutet, dass diese nicht durch einen Compiler oder ein anderes Offline-Werkzeug geprüft werden können.

Da die Anzahl der dynamischen Variablen und Objekte und der freie Speicherbereich, der neuen dynamischen Variablen und Objekten zugewiesen werden kann, vom Zustand des Systems zum Zeitpunkt der Zuweisung abhängen, ist es möglich, dass Fehler bei der Zuweisung oder der Verwendung der Variablen oder Objekte auftreten. Wenn zum Beispiel der Umfang des freien Speichers an der durch das System zugewie-

senen Stelle nicht ausreichend ist, kann der Speicherinhalt von vorhandenen Variablen unabsichtlich überschrieben werden. Diese Fehler werden vermieden, wenn dynamische Variablen und Objekte nicht verwendet werden.

Einschränkungen bei der Verwendung von dynamischen Objekten sind erforderlich, wenn das dynamische Verhalten durch statische Analyse (d. h. vor der Programm-Ausführung) nicht genau vorausgesagt werden kann und deshalb voraussagbare Programm-Ausführung nicht gewährleistet werden kann.

C.2.6.4 Online-Test während der Erstellung von dynamischen Variablen oder dynamischen Objekten

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle B.1 der IEC 61508-3 angeführt.

Ziel: Test, dass der Speicher, der für die Zuweisung von dynamischen Variablen oder Objekten vorgesehen ist, frei ist, bevor die Zuweisung stattfindet. Sicherstellung, dass die Zuweisung von dynamischen Variablen und Objekten während der Laufzeit keinen Einfluss auf bestehende Variablen, Daten oder Code hat.

Beschreibung: Im Sinne dieser Maßnahme sind dynamische Variablen und dynamische Objekte solche Variablen und Objekte, denen zur Laufzeit ihr Speicher zugewiesen wird und ihre absolute Adressen festgelegt werden (in diesem Sinn sind auch Attribute der Objektinstanzen Variablen).

Der Speicher wird durch Hardware- oder Softwaremittel getestet, um sicherzustellen, dass er frei ist, bevor er dynamischen Variablen oder Objekten zugewiesen wird (zum Beispiel, um Stacküberlauf zu vermeiden). Wird die Zuweisung nicht zugelassen (zum Beispiel, wenn der Speicher an der festgelegten Adresse nicht ausreichend ist), müssen geeignete Maßnahmen eingeleitet werden. Nach Verwendung einer dynamischen Variable oder eines dynamischen Objektes (zum Beispiel nach Verlassen eines Unterprogramms) muss der ganze Speicher, der diesen zugewiesen worden war, freigegeben werden.

ANMERKUNG 2 Eine Alternative ist der statische Nachweis, dass der Speicher unter allen Umständen ausreichend sein wird.

C.2.6.5 Eingeschränkte Verwendung von Interrupts

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.1 der IEC 61508-3 angeführt.

Ziel: Software verifizierbar und testbar halten.

Beschreibung: Die Verwendung von Interrupts sollte eingeschränkt werden. Interrupts dürfen verwendet werden, wenn sie das System vereinfachen. Die Bearbeitung von Interrupts durch die Software sollte während der Ausführung kritischer Teile von Funktionen (zum Beispiel zeitkritisch oder kritisch bezogen auf Datenveränderungen) verhindert werden. Nicht unterbrechbare Programmteile sollten bei Verwendung von Interrupts eine festgelegte höchste Rechendauer haben, so dass die längste Zeit, in der ein Interrupt unterbunden wird, berechnet werden kann. Interruptverwendung und -maskierung sollten sorgfältig dokumentiert werden.

C.2.6.6 Eingeschränkte Verwendung von Zeigern (en: pointers)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.1 der IEC 61508-3 angeführt.

Ziel: Vermeidung von Problemen, die durch Datenzugriffe verursacht werden, ohne zuvor den Bereich und den Typ des Zeigers getestet zu haben. Unterstützung modularer Tests und Verifikation der Software. Begrenzung der Auswirkungen von Ausfällen.

Beschreibung: Zeigerarithmetik darf auf der Ebene des Source-Codes in der Anwendungssoftware verwendet werden, wenn der Datentyp und der Wertebereich des Zeigers (um sicherzustellen, dass der Zeigerverweis innerhalb des korrekten Adressbereiches liegt) vor dem Zugriff getestet werden. Eine Kommunikation zwischen Programmteilen in der Anwendungssoftware sollte nicht durch direkten Zeigerverweis zwischen den Programmteilen ausgeführt werden. Der Datenaustausch sollte über das Betriebssystem ausgeführt werden.

C.2.6.7 Eingeschränkte Verwendung von Rekursion

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.1 der IEC 61508-3 angeführt.

Ziel: Vermeidung von nicht verifizierbaren und nicht testbaren Aufrufen von Unterprogrammen.

Beschreibung: Bei Verwendung einer Rekursion sollte ein eindeutiges Kriterium vorhanden sein, das die Tiefe der Rekursion vorhersagbar macht.

C.2.7 Strukturierte Programmierung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.4 der IEC 61508-3 angeführt.

Ziel: Entwurf und Implementierung des Programms derart, dass die Analyse ohne Ausführung durchführbar wird. Das Programm darf nur so wenig wie möglich statisch nicht testbares Verhalten beinhalten.

Beschreibung: Die folgenden Prinzipien sollten zu Verringerung struktureller Komplexität angewendet werden:

– Unterteilung des Programms in angemessen kleine Softwaremodule, um sicherzustellen, dass diese so weit wie möglich entkoppelt und alle Wechselwirkungen deutlich sind;
– Entwurf des Steuerflusses der Softwaremodule unter Verwendung strukturierter Konstrukte, diese sind Abfolge (Sequenz), Wiederholung (Iteration) und Auswahl (Selektion);
– möglichst wenig Pfade durch ein Softwaremodul und einfache Beziehungen zwischen den Eingabe- und Ausgabeparametern;
– Vermeidung komplizierter Verzweigungen und insbesondere unbedingter Sprünge (GOTO) in Hochsprachen;
– wenn möglich, sind Abbruch- und Aussprungbedingungen von Schleifen an Eingabeparameter zu knüpfen;
– Vermeidung komplizierter Berechnungen als Grundlage von Verzweigungen und Schleifenbedingungen.

Diejenigen Eigenschaften einer Programmiersprache, die das oben genannte Vorgehen fördern, sollten gegenüber anderen, die (angeblich) leistungsfähiger sind, bevorzugt verwendet werden, außer wenn Effizienz absolute Priorität hat (zum Beispiel bei einigen sicherheitskritischen Systemen).

Literaturhinweise:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

A Discipline of Programming. E. W. Dijkstra. Englewood Cliffs N J, Prentice-Hall, 1976

C.2.8 Geheimnisprinzip/Kapselung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.9 der IEC 61508-3 angeführt.

Ziel: Verhinderung des unbeabsichtigten Zugriffs auf Daten und Prozeduren und dadurch Unterstützung einer guten Programmstruktur.

Beschreibung: Daten, die global für alle Softwareelemente zugänglich sind, können versehentlich oder fälschlicherweise durch jedes dieser Elemente geändert werden. Alle Veränderungen dieser Datenstrukturen können detaillierte Prüfungen des Codes und umfangreiche Anpassungen erfordern.

Geheimnisprinzip ist ein allgemeiner Ansatz zur Verringerung dieser Schwierigkeiten. Die Strukturen der Schlüsseldaten sind „verborgen“ und können nur durch eine definierte Menge von Zugangsverfahren verändert werden. Dieses erlaubt es, die internen Strukturen zu verändern oder zukünftige Prozeduren hinzuzufügen, ohne das funktionale Verhalten der verbleibenden Software zu beeinflussen. Zum Beispiel kann ein

Namensverzeichnis die Zugriffsprozeduren „Einfügen“, „Löschen“ und „Finden“ haben. Die Zugriffsprozeduren und internen Datenstrukturen könnten neu geschrieben werden (zum Beispiel um einen unterschiedlichen Zugriffsalgorithmus zu verwenden oder um Namen auf einer Festplatte zu speichern), ohne das logische Verhalten der verbleibenden Software, die diese Zugriffsprozeduren verwendet, zu verändern.

In diesem Zusammenhang sollte das Konzept abstrakter Datentypen verwendet werden. Sofern eine direkte Unterstützung nicht bereitgestellt wird, könnte ein Test notwendig werden, ob die Abstraktion unabsichtlich verletzt wurde.

Literaturhinweise:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

On the Design and Development of Program Families. D. L. Parnas, IEEE Trans SE-2, March 1976

C.2.9 Modularer Ansatz

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.4 und B.9 der IEC 61508-3 angeführt.

Ziel: Aufteilung des Softwaresystems in kleine verständliche Teile, um die Komplexität des Systems zu begrenzen.

Beschreibung: Ein modularer Ansatz oder eine Modularisierung beinhaltet mehrere Regeln für die Entwurfs-, Codierungs- und Instandhaltungsphase eines Softwareprojekts. Diese Regeln variieren entsprechend der verwendeten Entwurfsmethode. Die meisten Methoden enthalten die folgenden Regeln:

– Ein Softwaremodul (oder ein gleichwertiges Unterprogramm) sollte eine einzige wohldefinierte Aufgabe oder zu erfüllende Funktion haben;
– Verbindungen zwischen Softwaremodulen sollten begrenzt und streng definiert werden, der innere Zusammenhang eines Softwaremoduls muss solide sein;
– Sammlungen von Unterprogrammen sollten zur Bereitstellung mehrerer Ebenen von Softwaremodulen gebildet werden;
– die Größe der Unterprogramme sollte auf einen bestimmten Wert eingeschränkt werden, typisch sind zwei bis vier Bildschirmseiten;
– Unterprogramme sollten nur einen Eingang und einen Ausgang haben;
– Softwaremodule sollten mit anderen Softwaremodulen durch ihre Schnittstellen kommunizieren – werden globale oder gemeinsame („global“ oder „common“) Variablen verwendet, sollten diese gut strukturiert sein, der Zugriff auf sie kontrolliert erfolgen und ihre Verwendung in jedem Fall gerechtfertigt werden;
– alle Schnittstellen der Softwaremodule sollten vollständig dokumentiert sein;
– jede Schnittstelle eines Softwaremoduls sollte nur die Parameter enthalten, die für dessen Funktion notwendig sind. Jedoch wird diese Empfehlung durch die Möglichkeit verkompliziert, dass eine Programmiersprache Default-Parameter zulässt oder dass ein objektorientierte Ansatz verwendet wird.

Literaturhinweise:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

C.2.10 Verwendung bewährter/verifizierter Softwareelemente

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.2, C.2, A.4 und C.4 der IEC 61508-3 angeführt.

Ziel: Vermeidung der Notwendigkeit einer umfangreichen Revalidierung oder eines Neuentwurfs der Software oder von Elementen für jede neue Anwendung. Verwenden von Entwürfen, die nicht formal oder streng verifiziert worden sind, für die jedoch eine umfangreiche Betriebserfahrung verfügbar ist. Verwenden von bereits existierenden Softwareelementen, die für eine andere Anwendung verifiziert worden sind und für die ein allgemeiner Verifikationsnachweis besteht.

Beschreibung: Diese Maßnahme verifiziert, dass die Softwareelemente ausreichend frei von Entwurfsfehlern und/oder systematischen Ausfällen während des Betriebs sind.

Es ist im Allgemeinen unpraktisch, ein komplexes System aus einzelnen Bauteilen aufzubauen. Es ist im Allgemeinen erforderlich, von größeren Unterbaugruppen Gebrauch zu machen („Elemente“, siehe IEC 61508-4, 3.4.5 und 3.2.8), die vorher entwickelt worden sind, um irgendeine nützliche Funktion zur Verfügung zu stellen, und die wieder verwendet werden können, um irgendein Teil des neuen Systems zu realisieren.

Gut entworfene und strukturierte PES bestehen aus einer Anzahl von Softwareelementen, die eindeutig sind und miteinander in einer eindeutig festgelegten Art und Weise zusammenwirken. Die Erstellung einer Bibliothek solcher allgemein anwendbaren Softwareelemente, die in mehreren Anwendungen wiederverwendet werden können, gestattet es, den größten Teil des notwendigen Aufwands für die Validierung des Entwurfs für mehr als eine Anwendung zu nutzen.

Es ist jedoch für sicherheitsbezogene Anwendungen notwendig, genügend Vertrauen zu haben, dass das neue System, das diese bereits existierenden Elemente aufnimmt, die erforderliche Sicherheitsintegrität hat, und dass die Sicherheit durch irgendein fehlerhaftes Verhalten des bereits existierenden Elements nicht beeinträchtigt wird.

Zwei Gesichtspunkte sind möglich, um Vertrauen zu gewinnen, dass das Verhalten des bereits existierenden Elements genau bekannt sein kann:

– Analyse einer umfangreichen Betriebserfahrung mit dem Element, um nachzuweisen, dass das Element „betriebsbewährt“ geworden ist;
– Beurteilung eines allgemeinen Verifikationsnachweises, der aus dem Verhalten des Elements erarbeitet worden ist, um festzustellen, ob das Element den Anforderungen dieser Norm entspricht.

C.2.10.1 Betriebsbewährt

Nur in seltenen Fällen wird Betriebsbewährung (siehe IEC 61508-4, 3.8.18) ein ausreichendes Argument sein, dass ein bewährtes Softwareelement die notwendige Sicherheitsintegrität erreicht. Für komplexe Elemente mit vielen möglichen Funktionen (z. B. für ein Betriebssystem) ist es wesentlich, festzustellen, welche Funktionen des Elements tatsächlich ausreichend betriebsbewährt sind. Wenn zum Beispiel eine Selbsttest-Routine zur Erkennung von Fehlern dient und innerhalb der Betriebsdauer keine Ausfälle auftraten, kann man sie auch nicht als betriebsbewährt ansehen.

Ein Softwareelement kann als betriebsbewährt angesehen werden, wenn es die folgenden Kriterien erfüllt:

– unveränderte Spezifikation;
– Systeme in unterschiedlichen Anwendungen im Einsatz;
– mindestens ein Jahr Betriebsaufzeichnung;

– Betriebsdauer entsprechend des Sicherheits-Integritätslevels oder einer angemessenen Anzahl von Anforderungen; Nachweis einer nicht sicherheitsbezogenen Ausfallrate von weniger als
  – 10^{-2} je Anforderung (Jahr) bei einem Vertrauensniveau von 95 % erfordert 300 betriebliche Durchläufe (Jahre),
  – 10^{-5} je Anforderung (Jahr) bei einem Vertrauensniveau von 99,9 % erfordert 690 000 betriebliche Durchläufe (Jahre);

ANMERKUNG 1 Siehe Anhang D bzgl. einiger mathematischer Hinweise zum Verständnis der obigen numerischen Abschätzung. Siehe auch B.5.4 für eine ähnliche Maßnahme und einen statistischen Ansatz.

– alle Betriebserfahrungen müssen sich auf ein bekanntes Anforderungsprofil der Funktionen des Softwareelements beziehen, um sicherzustellen, dass eine wachsende Betriebserfahrung wirklich zu einem wachsenden Wissen über das Verhalten des Softwareelements bezogen auf dieses Anforderungsprofil führt;
– keine sicherheitsbezogenen Ausfälle.

ANMERKUNG 2 Ein Ausfall, der in einem bestimmten Zusammenhang nicht sicherheitskritisch sein mag, kann in einem anderen sicherheitskritisch sein und umgekehrt.

Um die Verifikation zu ermöglichen, dass ein Softwareelement diese Kriterien erfüllt, muss Folgendes dokumentiert werden:

– genaue Identifizierung jedes Systems und seiner Elemente, einschließlich Versionsnummern (sowohl für Software als auch für Hardware);
– Identifizierung der Anwender und des Zeitpunkts der Anwendung;
– Betriebsdauer;
– Verfahren zur Auswahl des vom Anwender angewandten Systems und der Anwendungsfälle;
– Verfahren zur Erkennung und Aufzeichnung von Ausfällen und zur Beseitigung von Fehlern.

C.2.10.2 Beurteilung eines allgemeinen Verifikationsnachweises

Ein bereits existierendes Softwareelement (siehe IEC 61508-4, 3.2.8) ist ein Element, das bereits existiert und nicht speziell für das gegenwärtige Projekt oder SRS entwickelt worden ist. Die bereits existierende Software könnte ein kommerziell erhältliches Produkt sein oder sie könnte von irgendeiner Organisation für ein vorheriges Produkt oder System entwickelt worden sein. Bereits existierende Software kann oder kann nicht in Übereinstimmung mit den Anforderungen dieser Norm entwickelt worden sein.

Zur Beurteilung der Sicherheitsintegrität des neuen Systems, das die bereits existierende Software enthält, ist ein allgemeiner Verifikationsnachweis erforderlich, um das Verhalten des bereits existierenden Elements zu bestimmen. Dieser kann (1) von der Dokumentation des Anbieters des Elements und von den Aufzeichnungen des Entwicklungsprozesses des Elements abgeleitet werden, oder er kann (2) geschaffen oder durch zusätzliche Qualifikationstätigkeiten ergänzt werden, die vom Entwickler des neuen sicherheitsbezogenen Systems oder von Dritten vorgenommen worden sind. Dies ist das „Sicherheitshandbuch eines konformen Objekts“, das die Fähigkeiten und Beschränkungen des möglicherweise wieder verwendbaren Softwareelements definiert.

Auf jeden Fall muss ein Sicherheitshandbuch eines konformen Objekts existieren (oder muss geschaffen werden), das geeignet ist, eine Beurteilung der Integrität einer speziellen Sicherheitsfunktion zu ermöglichen, die ganz oder teilweise von dem wieder verwendeten Element abhängt. Ist dies nicht der Fall, dann muss die konservative Schlussfolgerung gezogen werden, dass das Element für die sicherheitsbezogene Wiederverwendung nicht berechtigt ist. (Dies soll nicht aussagen, dass das Element in jedem Fall nicht dazu berechtigt werden kann, sondern einfach, dass in diesem besonderen Fall kein ausreichender Nachweis gefunden wurde.)

Diese Norm nennt besondere Anforderungen für den Inhalt des Sicherheitshandbuchs eines konformen Objekts, siehe IEC 61508-2, Anhang D, IEC 61508-3, Anhang D, und IEC 61508-3, 7.4.2.12 und 7.4.2.13.

Folgende Punkte liefern einen Anhaltspunkt zum Inhalt des Sicherheitshandbuchs eines konformen Objekts:

– Der Entwurf des Elements ist bekannt und dokumentiert;
– das Element wurde einer Verifikation und einer Validierung unterzogen, wobei ein systematischer Ansatz mit dokumentierten Prüfungen verwendet wurde, mit Überprüfungen aller Teile des Entwurfs des Elements und des Codes;
– unbenutzte und nicht benötigte Funktionen des Elements werden das neue System nicht davon abhalten, seinen Anforderungen an die Sicherheit zu entsprechen;
– alle realistischen Ausfallmechanismen des Elements sind in dem neuen System identifiziert worden und geeignete Maßnahmen sind realisiert worden.

Eine Beurteilung der funktionalen Sicherheit des neuen Systems muss ermitteln, ob das wieder verwendete Element ausschließlich innerhalb der Grenzen des Leistungsvermögens eingesetzt wird, die durch die Nachweise und Annahmen im Sicherheitshandbuchs eines konformen Objekts bestimmt worden sind.

Literaturhinweise:

Component-Based Software Development: Case Studies. Kung-Kiu Lau. World Scientific, 2004, ISBN 9812388281, 9789812388285

Software Reuse and Reverse Engineering in Practice. P. A. V. Hall (ed.), Chapman & Hall, 1992, ISBN 0-412-39980-6

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

C.2.11 Nachvollziehbarkeit

Ziel: Aufrechterhaltung der Übereinstimmung zwischen den Lebenszyklusphasen.

Beschreibung: Es ist notwendig, Übereinstimmung zwischen den Lebenszyklusphasen zu gewährleisten, um sicherzustellen, dass die Software, die aus den Lebenszyklus-Tätigkeiten entsteht, den Anforderungen für den ordnungsgemäßen Betrieb des sicherheitsbezogenen Systems entspricht. Die „Nachvollziehbarkeit“ zwischen den Tätigkeiten ist hierzu ein Schlüsselkonzept. Das ist im Wesentlichen eine Einflussanalyse, um zu überprüfen, (1) dass die in einer früheren Phase getroffenen Entscheidungen in späteren Phasen (Vorwärtsverfolgbarkeit) entsprechend ausgeführt werden und (2) dass die in einer späteren Phase getroffenen Entscheidungen wirklich erforderlich sind und durch frühere Entscheidungen angeordnet worden sind.

Vorwärtsverfolgbarkeit ist weitgehend mit der Überprüfung beschäftigt, dass sich in späteren Lebenszyklusphasen angemessen mit einer Anforderung befasst wird. Vorwärtsverfolgbarkeit ist an mehreren Stellen im Sicherheitslebenszyklus nützlich:

– von den Anforderungen an die Sicherheit des Systems zu den Anforderungen an die Sicherheit der Software;
– von der Spezifikation der Anforderungen an die Sicherheit der Software zu der Softwarearchitektur;
– von der Spezifikation der Anforderungen an die Sicherheit der Software zu dem Softwareentwurf;
– von der Spezifikation des Softwareentwurfs zu den Spezifikationen der Modul- und Integrationstests;
– von den Anforderungen an den System- und Softwareentwurf der Hardware-/Softwareintegration zu den Spezifikationen der Hardware-/Softwareintegrationstests;
– von der Spezifikation der Anforderungen an die Sicherheit der Software zu dem Plan der Validierung der Softwaresicherheit;
– von der Spezifikation der Anforderungen an die Sicherheit der Software zu dem Plan der Softwaremodifikation (einschließlich Neuverifikation und Neuvalidierung);
– von der Spezifikation des Softwareentwurfs zum Plan der Softwareverifikation (einschließlich Verifikation der Daten);
– von den Anforderungen der IEC 61508-3, Abschnitt 8 zu dem Plan zur Beurteilung der funktionalen Softwaresicherheit.

Rückwärtsverfolgbarkeit ist weitgehend mit der Überprüfung beschäftigt, dass jede Implementierungsentscheidung (in einem breiten Zusammenhang und nicht auf die Implementierung des Codes begrenzt zu interpretieren) durch irgendeine Anforderung klar gerechtfertigt wird. Wenn diese Rechtfertigung fehlt, dann enthält die Implementierung etwas Unnötiges, das zur Komplexität beiträgt, sich aber nicht notwendigerweise an irgendeine tatsächliche Anforderung des sicherheitsbezogenen Systems richtet. Rückwärtsverfolgbarkeit ist an mehreren Stellen im Sicherheitslebenszyklus nützlich:

– von den Anforderungen an die Sicherheit zu den anerkannten Sicherheitsbedürfnissen;
– von der Softwarearchitektur zu der Spezifikation der Anforderungen an die Sicherheit der Software;
– vom detaillierten Softwareentwurf zu der Softwarearchitektur;
– vom Software Code zum detaillierten Softwareentwurf;
– vom Plan der Validierung der Softwaresicherheit zu der Spezifikation der Anforderungen an die Sicherheit der Software;
– vom Plan der Softwaremodifikation zu der Spezifikation der Anforderungen an die Sicherheit der Software;
– vom Plan der Softwareverifikation (einschließlich Verifikation der Daten) zu der Spezifikation des Softwareentwurfs.

Literaturhinweis:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.2.12 Zustandsloser Softwareentwurf (oder Entwurf eingeschränkter Zustände)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Begrenzung der Komplexität des Softwareverhaltens.

Beschreibung: Man betrachte ein Softwareprogramm, das eine Folge von Transaktionen verarbeitet: Es erhält eine Folge von Eingaben und erzeugt eine Ausgabe als Antwort zu jeder Eingabe. Das Programm kann ebenfalls einige oder alle seiner Historie in einem „Zustand“ abspeichern und kann diesen Zustand bei der Berechnung, wie man auf zukünftige Eingaben antwortet, berücksichtigen.

Sofern die Ausgabe des Programms als Antwort auf eine spezielle Eingabe durch diese Eingabe vollständig bestimmt wird, wird das Programm gedächtnislos oder zustandslos genannt. Jede Eingabe/Ausgabe-Transaktion ist in dem Sinne abgeschlossen, dass keine Transaktion durch irgendeine vorangegangene Transaktion beeinflusst wird und eine spezielle Eingabe immer dieselbe zugehörige Ausgabe zur Folge hat.

Wenn dagegen das Programm sowohl seine spezielle Eingabe als auch seinen gespeicherten Zustand bei der Berechnung seiner Ausgabe berücksichtigt, dann ist das Programm in der Lage, komplexere Verhalten aufzuweisen, weil es verschiedene Ausgaben als Antwort auf dieselbe Eingabe bei verschiedenen Gelegenheiten liefern kann. Die Antwort auf eine spezielle Eingabe kann vom Zusammenhang abhängen (d. h. von den vorangegangenen Eingaben und Ausgaben), in welchem die Eingabe empfangen wird. Ein weiterer Gesichtspunkt, der für einige Anwendungen relevant ist (typischerweise Kommunikationen), ist, dass das Verhalten des Programms besonders auf Änderungen im gespeicherten Zustand, entweder versehentlich oder absichtlich eingebracht, empfindlich sein kann.

Der zustandslose Entwurf (oder der Entwurf eingeschränkter Zustände) ist ein allgemeiner Ansatz, der beabsichtigt, die mögliche Komplexität des Softwareverhaltens durch Vermeidung oder Minimierung der Verwendung von Zustandsinformationen im Softwareentwurf zu minimieren.

Literaturhinweise:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN:0321462254

Stateless connections. T. Aura, P Nikander. In Proc International Conference on Information and Communications Security (ICICS’97), ed Yongfei Han. Springer, 1997, ISBN 354063696X, 9783540636960

C.2.13 Numerische Offline-Analyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.9 der IEC 61508-3 angeführt.

Ziel: Sicherstellung der Genauigkeit von numerischen Berechnungen.

Beschreibung: Bei der Berechnung von mathematischen Funktionen können als Folge der Verwendung von begrenzten Darstellungen von idealen Funktionen und Zahlen numerische Ungenauigkeiten entstehen. Ein Abbruchfehler tritt auf, wenn eine Funktion mit einer endlichen Anzahl von Größen einer unendlichen Reihe wie einer Fourier-Reihe angenähert wird. Ein Rundungsfehler tritt durch die begrenzt genaue Darstellung von reellen Zahlen in einem physikalischen Rechner auf. Wenn alles andere als einfachste Berechnungen in Fließkommadarstellung durchgeführt werden, muss die Gültigkeit der Berechnungen überprüft werden, um sicherzustellen, dass die durch die Anwendung erforderliche Genauigkeit wirklich erreicht wird.

Literaturhinweis:

Guide to Scientific Computing. P.R. Turner. CRC Press, 2001, ISBN 0849312426, 9780849312427

C.2.14 Nachrichtenverlaufstabellen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.7 und C.17 der IEC 61508-3 angeführt.

Ziel: Unterstützung der Erfassung von Anforderungen an das System in den frühen Entwurfsphasen der Softwareentwicklung einschließlich der Anforderungen und des Entwurfs der Softwarearchitektur. In UML wird die Bezeichnung „System-Sequenzdiagramm“ für diesen Begriff verwendet.

Beschreibung: Die Nachrichtenverlaufstabelle ist ein schematischer Mechanismus zur Beschreibung des Verhaltens eines Systems hinsichtlich der Kommunikation, die zwischen den Akteuren des Systems stattfindet (ein Akteur kann ein Mensch, ein Rechnersystem oder ein Softwareelement oder -objekt, abhängig von der Entwurfsphase, sein). Für jeden Akteur wird im Diagramm eine vertikale „Lebenslinie“ (en: lifeline) gezogen und es werden Pfeile zwischen den Lebenslinien verwendet, um Nachrichten darzustellen. Aktionen nach Empfang von Nachrichten können in den Diagrammen optional als Kasten dargestellt werden. Eine Sammlung von Szenarien, die sowohl erwünschtes als auch unerwünschtes Verhalten beschreiben, wird als Spezifikation des erforderlichen Systemverhaltens ermittelt. Diese Szenarien haben mehrere Nutzen. Sie können animiert werden, um den Endbenutzern das Systemverhalten zu zeigen. Sie können in eine ausführbare Implementierung des Systems überführt werden. Sie können die Basis von Testdaten bilden.

UML enthält Erweiterungen des originalen Konzepts der Nachrichtenverlaufstabellen in Form von Konstrukten zur Auswahl und Iterationen, die Verzweigungen und Schleifen erlauben. Dies stellt eine kompaktere Notation zur Verfügung. Es können ebenfalls Subdiagramme definiert werden, die von mehreren Sequenzdiagrammen höherer Ordnung referenziert werden können. Timer und externe Ereignisse können auch dargestellt werden.

Literaturhinweise:

„*Message Sequence charts“*, D. Harel, P. Thiagarajan. In UML for Real: Design of Embedded Real-Time Systems. ed. L. Lavagno. Springer, 2003, ISBN 1402075014, 9781402075018

ISO/IEC 19501:2005, *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

C.3 Entwurf der Architektur

C.3.1 Fehlererkennung und Diagnose

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Erkennung von Fehlern in einem System, die zu Ausfällen führen können. Dies bildet die Basis für Gegenmaßnahmen, um die Auswirkungen der Ausfälle zu verringern.

Beschreibung: Fehlererkennung ist die Tätigkeit, bei der ein System auf abweichende Zustände (verursacht durch einen Fehler innerhalb des zu testenden (Teil-)Systems) getestet wird. Das hauptsächliche Ziel der Fehlererkennung ist es, die Auswirkungen fehlerhafter Ergebnisse zu verhindern. Ein System, das in Kombination mit parallelen Elementen arbeitet, wird selbsttestend genannt, wenn es bei Erkennung eigener fehlerhafter Ergebnisse die Steuerungsfunktion abgibt.

Fehlererkennung basiert auf den Prinzipien der Redundanz (hauptsächlich, um Hardwarefehler aufzudecken, siehe IEC 61508-2, Anhang A) und der Diversität (für Softwarefehler). Eine Art Mehrheitsentscheidung zur Entscheidung über die Korrektheit der Ergebnisse wird benötigt. Geeignete Methoden sind: Programmierung von Assertions, Programmierung von *N* diversitären Softwarekanälen und diversitären Überwachungseinrichtungen sowie für Hardware das Einbinden von zusätzlichen Sensoren und Regelschleifen, Fehlersicherungscodes usw.

Fehlererkennung kann durch Test des Wertebereichs oder des Zeitverhaltens auf verschiedenen Ebenen erreicht werden, insbesondere physikalisch (Temperatur, Spannung usw.), logisch (fehlererkennende Codes), funktional (Assertions) oder extern (Plausibilitätskontrollen). Die Ergebnisse dieser Tests können abgespeichert und mit den betroffenen Daten in Bezug gebracht werden, um eine Nachverfolgung der Ausfälle zu ermöglichen.

Komplexe Systeme werden aus Teilsystemen zusammengesetzt. Die Effizienz der Fehlererkennung, Diagnose und Fehlerkompensation hängt von der Komplexität der Wechselwirkungen unter den Teilsystemen ab, die die Fortpflanzung von Fehlern beeinflusst.

Fehlerdiagnose sollte auf die kleinste Ebene von Teilsystemen angewendet werden, da kleinere Teilsysteme eine detailliertere Diagnose von Fehlern erlauben (Erkennung von abweichenden Zuständen).

In einem integrierten unternehmensweiten Informationssystem kann der Zustand eines Sicherheitssystems einschließlich Diagnoseinformationen regelmäßig an übergeordnete Systeme übermittelt werden. Wird ein außergewöhnliches Vorkommnis erkannt, dann kann dieses gemeldet und eine korrigierende Aktion veranlasst werden, bevor eine Gefährdung entsteht. Kommt es dennoch zu einem Unfall, kann die Dokumentation solcher besonderen Vorkommnisse die nachfolgende Untersuchung erleichtern.

Literaturhinweis:

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

C.3.2 Fehlererkennende und korrigierende Codes

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Erkennung und Korrektur von Fehlern in kritischen Informationen.

Beschreibung: Für eine Information von *n* Bit wird ein codierter Block von *k* Bit generiert, der es ermöglicht, *r* Abweichungen zu erkennen und zu korrigieren. Zwei Beispiele sind Hammingcodes und polynomische Codes.

Da nur ein bestimmter Anteil von Fehlern korrigiert werden kann, besteht in einem sicherheitsbezogenen System normalerweise die Notwendigkeit, fehlerhafte Daten zu verwerfen, anstatt zu versuchen, sie zu korrigieren.

Literaturhinweis:

Fundamentals of Error-correcting Codes, W. Huffman, V. Pless. Cambridge University Press, 2003, ISBN 0521782805, 9780521782807

C.3.3 Assertion-Programmierung (en: failure assertion programming)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.17 der IEC 61508-2 und in den Tabellen A.2 und C.2 der IEC 61508-3 angeführt.

Ziel: Erkennung von Restfehlern im Softwareentwurf während der Ausführung eines Programms, um sicherheitskritische Ausfälle des Systems zu verhindern und um den Betrieb mit hoher Zuverlässigkeit fortzusetzen.

Beschreibung: Die Methode der Assertion-Programmierung folgt der Idee des Tests einer Vorbedingung (vor Ausführung einer Befehlsfolge werden die Startbedingungen auf Gültigkeit getestet) und einer Nachbedingung (Ergebnisse werden nach der Ausführung einer Befehlsfolge geprüft). Ist entweder die Vorbedingung oder die Nachbedingung nicht erfüllt, wird eine Abweichung gemeldet.

Zum Beispiel:

```
Behauptung <Vorbedingung>;
    Aktion 1;
        :
        :
    Aktion x;
Behauptung <Nachbedingung>;
```

Literaturhinweise:

Exploiting Traces in Program Analysis. A. Groce, R. Joshi. Lecture Notes in Computer Science vol 3920, Springer Berlin / Heidelberg, 2006, ISBN 978-3-540-33056-1

Software Development – A Rigorous Approach. C. B. Jones, Prentice-Hall, 1980

C.3.4 Diversitäre Überwachungseinrichtung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Schutz gegen Spezifikations- und Ausführungsrestfehler in der Software, die die Sicherheit nachteilig beeinflussen.

Beschreibung: Es können zwei Ansätze zur Überwachung unterschieden werden: (1) die Überwachungseinrichtung und die überwachte Funktion in demselben Rechner mit einem bestimmten Unabhängigkeitsgrad zwischen beiden und (2) die Überwachungseinrichtung und die überwachte Funktion in getrennten Rechnern.

Eine diversitäre Überwachungseinrichtung wird auf einem unabhängigen Rechner nach einer eigenständigen Spezifikation ausgeführt. Diese diversitäre Überwachungseinrichtung ist ausschließlich damit betraut, sicherzustellen, dass der Hauptrechner sicher, nicht notwendigerweise richtig arbeitet. Sie überwacht kontinuierlich den Hauptrechner und verhindert, dass das System in einen unsicheren Zustand gerät. Wenn sie weiterhin erkennt, dass der Hauptrechner einen möglicherweise gefährlichen Zustand einnimmt, muss das System entweder durch die diversitäre Überwachungseinrichtung oder durch den Hauptrechner zurück in einen sicheren Zustand gebracht werden.

Die Hardware und Software der diversitären Überwachungseinrichtung sollte entsprechend eines angemessenen SIL klassifiziert und qualifiziert werden.

Literaturhinweis:

Requirements based Monitors for Real-Time Systems, D. Peters, D. Parnas. IEEE Transactions on Software Engineering, vol. 28, no. 2, 2002

C.3.5 Diversitäre Programmierung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Erkennung und Maskierung von Restfehlern im Softwareentwurf und der Softwarerealisierung während der Ausführung eines Programms, um sicherheitskritische Ausfälle des Systems zu verhindern und um den Betrieb mit hoher Zuverlässigkeit fortzusetzen.

Beschreibung: Bei der diversitären Programmierung wird entsprechend einer vorhandenen Programmspezifikation ein Programm *n*-mal auf unterschiedliche Weise entworfen und implementiert. Die gleichen Eingangswerte werden den *n* Kanälen übergeben und die Ergebnisse dieser *n* Kanäle werden verglichen. Wenn das Ergebnis für gültig erklärt wird, wird es zu den Ausgängen des Rechners übertragen.

Eine wesentliche Anforderung ist, dass die *n* Kanäle in dem Sinne voneinander unabhängig sind, dass sie wegen derselben Ursache nicht alle gleichzeitig ausfallen. In der Praxis kann es sehr schwierig sein, die Unabhängigkeit der Kanäle, die die Grundlage des Ansatzes der *n*-Kanaligkeit ist, zu erreichen und nachzuweisen.

Die *n* Kanäle können gleichzeitig auf verschiedenen Rechnern ablaufen oder alle auf dem gleichen und die Ergebnisse einer internen Bewertung unterzogen werden. Je nach den Anforderungen der Anwendung können für die *n* Kanäle die im Folgenden aufgezeigten unterschiedlichen Entscheidungsstrategien verwendet werden.

– Wenn es für das System einen sicheren Zustand gibt, ist es sinnvoll, vollständige Übereinstimmung (alle *n* Kanäle stimmen zu) zu fordern, ansonsten wird ein Ausgabewert verwendet, der das System in den sicheren Zustand bringt. In einfach stillzusetzenden Systemen kann die sichere Richtung bei der Entscheidung bevorzugt werden. In diesem Fall wäre die sicherheitsgerichtete Aktion das Stillsetzen, wenn sie von einem Kanal gefordert wird. Dieser Ansatz verwendet typischerweise nur zwei Kanäle ($n = 2$).
– Bei Systemen, für die es keinen sicheren Zustand gibt, kann Mehrheitsentscheidung angewendet werden. In Fällen, in denen keine Übereinstimmung vorliegt, können probabilistische Ansätze verwendet werden, um mit größter Wahrscheinlichkeit den richtigen Wert zu erhalten, zum Beispiel durch Mittelwertbildung, vorübergehende Beibehaltung der Ausgangsgrößen bis zu erneuten Übereinstimmung usw.

Dieses Verfahren beseitigt weder Restfehler im Softwareentwurf noch vermeidet es Abweichungen in der Auslegung der Spezifikation, aber es stellt eine Maßnahme zur Erkennung und Maskierung dieser Fehler bereit, bevor sie die Sicherheit beeinflussen können.

Literaturhinweise:

Modelling software design diversity – a review, B. Littlewood, P. Popov, L. Strigini. ACM Computing Surveys, vol 33, no 2, 2001

The N-Version Approach to Fault-Tolerant Software, A.Avizienis, IEEE Transactions on Software Engineering, vol. SE-11, no. 12 pp.1491-1501, 1985

An experimental evaluation of the assumption of independence in multi-version programming, J.C. Knight, N.G. Leveson. IEEE Transactions on Software Engineering, vol. SE-12, no 1, 1986

In Search of Effective Diversity: a Six Language Study of Fault-Tolerant Flight Control Software. A. Avizienis, M. R. Lyu and W. Schutz. 18th Symposium on Fault-Tolerant Computing, Tokyo, Japan, 27-30 June 1988, IEEE Computer Society Press, 1988, ISBN 0-8186-0867-6

C.3.6 Rückwärtsregeneration

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Ermöglichen, dass die Funktionen in Anwesenheit eines oder mehrerer Fehler richtig ausgeführt werden.

Beschreibung: Wenn ein Fehler erkannt worden ist, wird das System in einen früheren internen Zustand zurückgesetzt, dessen Richtigkeit zuvor bewiesen worden war. Diese Methode erfordert das häufige Abspeichern interner Zustände an ausgewählten Testpunkten. Dieses kann global (für den vollständigen Datensatz) oder inkremental (nur für Veränderungen zwischen den Testpunkten) erfolgen. Das System muss dann die Änderungen, die in der Zwischenzeit stattgefunden haben, durch Verwendung eines Logbuchs (Audit der Aktionspfade), durch Kompensation (alle Änderungseffekte werden annulliert) oder durch externe (manuelle) Einwirkung unwirksam machen.

Literaturhinweise:

Looking into Compensable Transactions. Jing Li, Huibiao Zhu, Geguang Pu, Jifeng He. In Software Engineering Workshop, 2007. SEW 2007. IEEE, 2007, ISBN 978-0-7695-2862-5

Software Fault Tolerance (Trends in Software, No. 3). M. R. Lyu (ed.), John Wiley & Sons, April 1995, ISBN 0471950688

C.3.7 Regeneration durch Wiederholung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Versuch der funktionalen Regeneration (Erholung) von einem erkannten Fehlzustand durch Wiederholungsmechanismen.

Beschreibung: Im Falle eines erkannten Fehlers oder eines abweichenden Zustands wird versucht, die Situation durch erneute Ausführung des gleichen Codes zu retten. Regeneration durch Wiederholung nach einem Software-Timeout oder einem Eingriff des Task-Managers kann dabei bis zum Neustart (erneutes Booten und Hochfahren) und/oder erneutem Aufsetzen der Task gehen. Wiederholungsverfahren werden häufig zur Fehlererholung nach Kommunikationsfehlern verwendet, wobei deren Erkennung im Kommunikationsprotokoll festgelegt sein kann, z. B. bei Prüfsummenfehler oder Zeitüberschreitung

Literaturhinweis:

Reliable Computer Systems: Design and Evaluation, D.P. Siewiorek, R.S. Schwartz. A.K. Peters Ltd., 1998, ISBN 156881092X, 9781568810928

C.3.8 Abgestufte Funktionseinschränkungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Aufrechterhaltung der kritischeren Systemfunktionen, trotz vorhandener Ausfälle, durch Verzicht auf die weniger kritischen Funktionen.

Beschreibung: Dieses Verfahren priorisiert die verschiedenen Funktionen, die von dem System ausgeführt werden. Der Entwurf stellt sicher, dass die höher priorisierten Funktionen vor den niedrigeren bevorzugt ausgeführt werden, wenn die Ressourcen zur Ausführung aller Systemfunktionen nicht ausreichen. Logbuch-Funktionen können zum Beispiel von niedrigerer Priorität als die Systemsteuerfunktionen sein. In diesem Fall würde die Systemsteuerung weiter fortgeführt werden, wenn die Hardware für die Logbuch-Funktion ausfällt. Fällt dagegen die Hardware zur Systemsteuerung aus und nicht die für die Logbuch-Funktion, sollte letztere die Systemsteuerung übernehmen.

Diese Methode wird vorwiegend bei Hardware angewendet, sie ist aber auch auf das Gesamtsystem einschließlich Software anwendbar. Sie muss bereits am Anfang der Entwurfsphase berücksichtigt werden.

Literaturhinweise:

Towards the Integration of Fault, Resource, and Power Management, T. Siridakis. In Computer Safety, Reliability, and Security: 23rd International Conference, SAFECOMP 2004. Eds. Maritta Heisel et. al. Springer, 2004, ISBN 3540231765, 9783540231769

Achieving Critical System Survivability Through Software Architectures, J.C. Knight, E.A. Strunk. Springer Berlin / Heidelberg, 2004, 978-ISBN 3-540-23168-4

The Evolution of Fault-Tolerant Computing. Vol. 1 of Dependable Computing and Fault-Tolerant Systems, Edited by A. Avizienis, H. Kopetz and J. C. Laprie, Springer Verlag, 1987, ISBN 3-211-81941-X

Fault Tolerance, Principle and Practices. T. Anderson and P. A. Lee, Vol. 3 of Dependable Computing and Fault-Tolerant Systems, Springer Verlag, 1987, ISBN 3-211-82077-9

C.3.9 Künstliche Intelligenz – Fehlerkorrektur

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Auf mögliche Gefährdungen in einer sehr flexiblen Art reagieren, indem eine Kombination aus Methoden und Prozessmodellen und einer Art von Online-Sicherheits- und Zuverlässigkeitsanalysen eingeführt wird.

Beschreibung: Durch Systeme auf der Grundlage künstlicher Intelligenz (en: artificial intelligence, AI) können Fehlervorhersagen (Trendrechnungen), Fehlerkorrekturen sowie Instandhaltungs- und Überwachungstätigkeiten sehr wirkungsvoll in diversitären Kanälen eines Systems unterstützt werden, weil die Regeln direkt von der Spezifikation abgeleitet und gegen diese getestet werden können. Einige Fehler gemeinsamer Ursache, die aufgrund bestimmter Realisierungsvorstellungen bei der Erstellung der Spezifikation gemacht worden sind, können durch diesen Ansatz wirksam vermieden werden. Dies gilt besonders, wenn eine Kombination von Modellen und Methoden in einer funktionalen oder beschreibenden Art angewendet wird.

Um die gewünschte Sicherheitsintegrität zu erreichen, werden die Methoden derart ausgewählt, dass Fehler korrigiert und die Auswirkungen der Ausfälle vermindert werden.

ANMERKUNG 2 Siehe C.3.2 als Hinweis zur Korrektur fehlerhafter Daten und Eintrag 5 der Tabelle A.2 der IEC 61508-3 bezüglich negativer Empfehlungen zu diesem Verfahren.

Literaturhinweis:

Fault Diagnosis: Models, Artificial Intelligence, Applications. J. Korbicz, J. Koscielny, Z. Kowalczuk, W. Cholewa. Springer, 2004, ISBN 3540407677, 9783540407676

C.3.10 Dynamische Rekonfiguration

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Aufrechterhaltung der Systemfunktionalität trotz eines internen Fehlers.

Beschreibung: Die logische Architektur des Systems muss derart sein, dass es auf eine Teilmenge der verfügbaren Systemressourcen abgebildet werden kann. Die Architektur muss zur Erkennung eines Ausfalls einer physikalischen Ressource und zur Umschaltung der logischen Architektur auf das eingeschränkte, noch funktionierende System in der Lage sein. Obwohl das Konzept bisher eher auf die Regeneration ausgefallener Hardwareeinheiten angewendet wurde, kann es ebenso auf ausgefallene Softwareeinheiten angewendet werden, wenn ausreichende Laufzeitreserven eine softwaremäßige Wiederholung gestatten oder wenn ausreichend redundante Daten zur Verfügung stehen, um den Ausfall zu isolieren und seine Auswirkungen gering zu halten.

Dieses Verfahren muss bei der ersten Phase des Systementwurfs berücksichtigt werden.

Literaturhinweis:

Dynamic Reconfiguration of Software Architectures Through Aspects. C. Costa et al. Lecture Notes in Computer Science, Volume 4758/2007, Springer Berlin / Heidelberg, 2007, ISBN 978-3-540-75131-1

C.3.11 Sicherheit und Leistungsfähigkeit in Echtzeit: zeitgesteuerte Architektur

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Komponierbarkeit und transparente Implementierung der Fehlertoleranz in sicherheitskritische Echtzeitsysteme mit vorhersagbarem Verhalten.

Beschreibung: In einem System mit zeitgesteuerter Architektur (en: time-triggered architecture, TTA) leitet das Fortschreiten einer global synchronisierten Zeitbasis alle Systemtätigkeiten ein und dient als deren Basis. Jeder Anwendung wird ein festgelegter Zeitschlitz auf dem zeitgesteuerten Bus zugeteilt, der die Nachrichten enthält, die zwischen den Aufgaben jeder Anwendung ausgetauscht werden, die deshalb nur nach einem definierten Ablaufplan ausgetauscht werden können. In ereignisgesteuerten Systemen werden die Systemtätigkeiten durch beliebige Ereignisse zu unvorhersehbaren Zeitpunkten ausgelöst. Die entscheidenden Vorteile eines TTA-Systems sind (siehe Literaturhinweis Scheidler, Heiner et. al.):

– Komponierbarkeit, die die Aufwendungen in hohem Maße reduziert, die erforderlich sind, um das System zu prüfen und zu zertifizieren;
– transparente Implementierung der Fehlertoleranz, die die Architektur sehr empfehlenswert für sicherheitskritische Anwendungen macht;
– Bereitstellung einer global synchronisierten Zeitbasis, die den Entwurf von verteilten Echtzeitsystemen erleichtert.

Die Kommunikation zwischen den Knoten wird unter Verwendung des zeitgesteuerten Protokolls TTP/C (en: time-triggered protocol class C, siehe Literaturhinweis Kopetz, Hexel et. al.) gemäß einem statischen Ablaufplan ausgeführt, der festlegt, wann eine Nachricht zu übertragen ist und ob eine empfangene Nachricht für das einzelne elektronische Modul relevant ist oder nicht. Der Zugriff auf den Bus wird über ein Zugriffsschema mit Zeitmultiplexverfahren (TDMA, en: time-division multiple access) gesteuert, das von einem globalen Zeitbegriff abgeleitet ist.

Das TTP/C-Protokoll gewährleistet (siehe Literaturhinweis Rushby) in einem Netz von TTA-Knoten (siehe Literaturhinweis Kopetz, Bauer) vier grundlegende Dienste (Kerndienste):

– Deterministische und rechtzeitige Nachrichtenübertragung: Übertragung von Nachrichten vom Ausgangsport des sendenden Elements zum Eingangsport des empfangenden Elements innerhalb einer von vornherein bekannten zeitlichen Frist. Ein fehlertoleranter Übertragungsdienst wird von einen zeitgesteuerten Kommunikationsdienst angeboten, der über die zeitliche Firewall-Schnittstelle verfügbar ist, die die Fortpflanzung von Steuerungsfehlern durch den Entwurf beseitigt und die Kopplung zwischen den Elementen minimiert. Die rechtzeitige Übertragung von Nachrichten mit minimaler Latenz und Jitter ist für das Erreichen der Stabilität der Steuerung in Echtzeitanwendungen entscheidend.
– Fehlertolerante Takt-Synchronisation: Der Kommunikationsmaster erzeugt eine fehlertolerante synchronisierte globale Zeitbasis (mit einer Genauigkeit von wenigen Takten), die dem Host-Teilsystem zur Verfügung gestellt wird.
– Konsequente Diagnose ausgefallener Knoten (Membership-Service): Der Kommunikationsmaster informiert jede SRU (kleinste austauschbare Einheit, en: smallest replaceable unit) über den Zustand jeder anderen SRU in der Gruppe mit einer Latenz von weniger als einem TDMA-Zyklus.
– Strikte Fehler-Isolierung: Ein fehlerhaftes Host-Teilsystem (einschließlich seiner Software) kann fehlerhafte Ausgangsdaten erzeugen, kann aber niemals auf irgendeiner Weise den ordnungsgemäßen Betrieb des Rests der TTP/C-Gruppe stören. Ein Fail Silence in der Temporal Domain wird durch das zeitgesteuerte Verhalten des Kommunikationsmasters gewährleistet.

ANMERKUNG 2 Weitere zeitgesteuerte Protokolle sind FlexRay und TT-Ethernet (time-triggered Ethernet).

Literaturhinweise:

Time-Triggered Architecture (TTA). C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, C. Temple. In Advances in Information Technologies: The Business Challenge, ed. J-Y. Roger. IOS Press, 1998, ISBN 9051993854, 9789051993851

A Synchronisation Strategy for a TTP/C Controller. H. Kopetz, R. Hexel, A. Krueger, D. Millinger, A. Schedl. SAE paper 960120, Application of Multiplexing Technology SP 1137, Detroit, SAE Press, Warrendale, 1996

The Time-Triggered Architecture. H. Kopetz, G. Bauer. Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, October 2002

An Overview of Formal Verification for the Time-Triggered Architecture. J. Rushby: Invited paper, Oldenburg, Germany, September 9-12, 2002. Proceedings FTRTFT 2002, Springer LNCS 2469, 2002, ISBN 978-3-540-44165-6

C.3.12 UML

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.7 der IEC 61508-3 angeführt.

Ziel: Bereitstellung einer umfangreichen Auswahl von Notationen, um das gewünschte Verhalten von komplexen Systemen zu modellieren.

Beschreibung: UML ist, wie der Name andeutet, eine Sammlung von Anforderungen und Entwurfsnotationen, die dazu vorgesehen ist, umfassende Unterstützung für die Softwareentwicklung zur Verfügung zu stellen. Einige Bestandteile von UML basieren auf in anderen Methoden zuerst eingeführten Notationen (wie System-Sequenzdiagrammen und Zustandsübergangsdiagrammen), während andere Notationen aus UML sind. UML ist stark auf objektorientierte Konzepte gerichtet, obwohl einige der Notationen verwendet werden können, ohne mit einer objektorientierten Programmierung fortzufahren. UML wird von mehreren kommerziell erhältlichen CASE-Tools unterstützt, von denen viele in der Lage sind, automatisch Programmcodes aus den UML-Modellen zu erzeugen.

Folgende UML-Notationen sind allgemein auf die Spezifikation und den Entwurf von sicherheitsbezogenen Systemen am anwendbarsten:

– Klassendiagramme;
– Anwendungsfälle;
– Aktivitätsdiagramme;
– Zustandsübergangsdiagramme (Zustandsdiagramme);
– System-Sequenzdiagramme.

Andere UML-Notationen sind für die Darstellung des Entwurfs der Softwarearchitektur (Softwarestruktur) wichtig, werden aber hier nicht besonders aufgelistet.

Zustandsübergangsdiagramme werden in B.2.3.2 und System-Sequenzdiagramme in C.2.14 beschrieben. Die anderen Notationen werden in den folgenden drei Unterabschnitten beschrieben.

C.3.12.1 Klassendiagramme

Klassendiagramme definieren die Klassen von Objekten, mit denen sich die Software befassen muss. Sie basieren auf früheren Entity-Relationship-Attribut-Diagrammen, werden aber an den objektorientierten Entwurf angepasst. Jede Klasse (von denen es eine oder mehrere Instanzen, bekannt als Objekte, zur Laufzeit geben wird) wird als ein Rechteck dargestellt und die verschiedenen Beziehungen zwischen den Klassen werden als Linien oder Pfeile gezeigt. Die Operationen oder Methoden, die durch jede Klasse bereitgestellt werden, und die Dateneigenschaften jeder Klasse können zum Diagramm hinzugefügt werden. Die Beziehungen, die dargestellt werden können, sind die Bezugsbeziehungen mit ihrer Kardinalität (eine Instanz der Klasse A kann sich auf eine oder mehrere Instanzen der Klasse B beziehen) und Spezialisierungsbeziehungen (Klasse X ist eine Verbesserung der Klasse Y) mit möglicherweise zusätzlichen Methoden und Eigenschaften. Mehrfachvererbung kann wiedergegeben werden.

C.3.12.2 Anwendungsfälle

Anwendungsfälle stellen eine textuelle Beschreibung des gewünschten Systemverhaltens als Reaktion auf bestimmte Szenarien, gewöhnlich aus dem Blickwinkel von externen Akteuren einschließlich menschlichen

Anwendern des Systems und externen Systemen, zur Verfügung. Alternative untergeordnete Szenarien innerhalb eines gegebenen Anwendungsfalls können verwendet werden, um optionales Verhalten, besonders in Fehlerreaktionsfällen, darzustellen. Eine Sammlung von Anwendungsfällen wird entwickelt, um eine ausreichend vollständige Spezifikation der Anforderungen an das System zur Verfügung zu stellen. Anwendungsfälle können der Ausgangspunkt für die Entwicklung strenger Modelle wie zum Beispiel System-Sequenzdiagrammen und Aktivitätsdiagrammen sein.

Diagramme der Anwendungsfälle stellen eine bildliche Darstellung des Systems und der Akteure, die an den Anwendungsfällen beteiligt sind, zur Verfügung. Sie sind aber nicht entscheidend. Nur der Text der Anwendungsfälle ist für die Spezifikation wichtig.

C.3.12.3 Aktivitätsdiagramme

Ein Aktivitätsdiagramm zeigt die beabsichtigte Abfolge von durch ein Softwareelement ausgeführten Handlungen (häufig ein Objekt in einem objektorientierten Entwurf) einschließlich des sequentiellen und iterativen Verhaltens (einige Aspekte sind bemerkenswerterweise einem Flussdiagramm ähnlich). Aktivitätsdiagramme erlauben jedoch den Handlungen mehrerer Elemente nebeneinander mit den Interaktionen zwischen den Elementen, die durch Pfeile im Diagramm gezeigt werden, beschrieben zu werden. Synchronisationspunkte an denen eine Tätigkeit auf eine oder mehrere Eingänge anderer Tätigkeiten warten muss, bevor es weitergehen kann, werden durch ein Symbol ähnlich einem Petri-Netz-Knoten gezeigt.

Literaturhinweis:

ISO/IEC 19501:2005, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*

C.4 Entwicklungswerkzeuge und Programmiersprachen

C.4.1 Streng typisierte Programmiersprache

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-3 angeführt.

Ziel: Reduzierung der Wahrscheinlichkeit von Fehlern durch Verwendung einer Sprache, die einen hohen Grad des Tests durch den Compiler gestattet.

Beschreibung: Beim Kompilieren einer streng typisierten Programmiersprache werden viele Tests zur Verwendung von Variablentypen verwendet, zum Beispiel in Prozeduraufrufen und bei externen Datenzugriffen. Bei jeder Verwendung, die von den vordefinierten Regeln abweicht, wird die Kompilation fehlschlagen und eine Meldung über die Abweichung erzeugt.

Solche Sprachen gestatten normalerweise anwenderdefinierte Datentypen, die mit Hilfe der Grunddatentypen der Sprache definiert werden (wie z. B. INTEGER, REAL). Diese Typen können dann genauso wie die Grundtypen verwendet werden. Detaillierte Tests werden ausgeführt, um zu gewährleisten, dass der korrekte Typ verwendet wird. Sie werden im gesamten Programm ausgeführt, auch wenn dieses aus getrennt kompilierten Einheiten besteht. Sie stellen auch sicher, dass die Anzahl und der Typ der Argumente der Unterprogramme zutreffend sind, sogar wenn sie von separat kompilierten Softwaremodulen aufgerufen werden.

Streng typisierte Sprachen unterstützen auch andere Aspekte einer geeigneten Programmiertechnik, wie leicht analysierbare Kontrollflussstrukturen (zum Beispiel IF .. THEN .. ELSE, DO .. WHILE usw.), die zu gut strukturierten Programmen führen.

Typische Beispiele von streng typisierten Sprachen sind Pascal, Ada und Modula 2.

Literaturhinweis:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

C.4.2 Sprachenteilmenge

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-3 angeführt.

Ziel: Reduzierung der Wahrscheinlichkeit von Programmierfehlern und Erhöhung der Wahrscheinlichkeit der Erkennung von Restfehlern.

Beschreibung: Die Sprache wird zum Beispiel unter Verwendung statischer Analysemethoden geprüft, um Programmierungskonstrukte zu ermitteln, die entweder anfällig für Abweichungen oder schwierig zu analysieren sind. Dann wird eine Sprachenteilmenge definiert, die diese Konstrukte ausschließt.

Literaturhinweise:

Practical Experiences of Safety- and Security-Critical Technologies, P. Amey, A.J. Hilton. Ada User Journal, June, 2004

Safer C: Developing Software for High-integrity and Safety-critical Systems. L. Hatton, McGraw-Hill, 1994, ISBN 0077076400, 9780077076405

Requirements for programming languages in safety and security software standard. B. A. Wichmann, Computer Standards and Interfaces. Vol, 14, pp 433-441, 1992

C.4.3 Zertifizierte Werkzeuge und Compiler

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-3 angeführt.

Ziel: Werkzeuge sind notwendig, um den Entwicklern in den unterschiedlichen Phasen der Softwareentwicklung Hilfestellung zu gewähren. Die Werkzeuge sollten möglichst zertifiziert sein, so dass ein gewisser Vertrauensgrad bezüglich der Korrektheit der Ausgaben angenommen werden kann.

Beschreibung: Die Zertifizierung eines Werkzeugs wird im Allgemeinen von einer unabhängigen, meist nationalen Institution nach unabhängigen Kriterien, typischerweise nationalen oder internationalen Normen, ausgeführt. Idealerweise sollten die in allen Entwicklungsphasen (Spezifikation, Entwurf, Codierung, Test und Validierung) sowie die für das Konfigurationsmanagement verwendeten Werkzeuge zertifiziert sein.

Zurzeit werden in der Regel nur Compiler Zertifizierungsverfahren unterworfen, welche von nationalen Zertifizierungsstellen festgelegt wurden. Sie führen die Prüfung der Compiler nach Internationalen Normen durch, z. B. für Ada und Pascal.

Es ist wichtig, zur Kenntnis zu nehmen, dass zertifizierte Werkzeuge und zertifizierte Compiler gewöhnlich nur nach ihren jeweiligen Sprach- oder Prozessnormen, jedoch nicht unter Berücksichtigung der Sicherheit zertifiziert werden.

Literaturhinweise:

The certification of software tools with respect to software standards, P. Bunyakiati et al. In IEEE International Conference on Information Reuse and Integration, IRI 2007, IEEE, 2007, ISBN 1-4244-1500-4

Certified Testing of C Compilers for Embedded Systems. O. Morgan. In: 3rd Institution of Engineering and Technology Conference on Automotive Electronics. IEEE, 2007, ISBN 978-0-86341-815-0

The Ada Conformity Assessment Test Suite (ACATS), version 2.5, Ada Conformity Assessment Authority, http://www.ic.org/compilerstesting.html , Apr. 2002

C.4.4 Betriebsbewährte Werkzeuge und Compiler

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-3 angeführt.

Ziel: Vermeidung aller Schwierigkeiten durch Versagen eines Compilers, die während der Entwicklung, Verifikation und Pflege eines Softwarepakets auftreten können.

Beschreibung: Es wird ein Compiler verwendet, bei dem in vielen früheren Projekten keine Anzeichen für nicht korrekte Ausführung auftraten. Compiler ohne Betriebserfahrung oder mit irgendwelchen schwerwiegenden bekannten Fehlern sollten vermieden werden, außer die korrekte Ausführung wird anderweitig sichergestellt (z. B. siehe C.4.4.1).

Zeigte der Compiler geringfügige Mängel, werden die zugehörigen Sprachkonstrukte festgehalten und während eines sicherheitsbezogenen Projekts sorgfältig vermieden.

Eine andere Art dieser Arbeitsweise ist es, die Verwendung der Sprache auf ihre normalerweise verwendeten Möglichkeiten zu beschränken.

Diese Empfehlung basiert auf der Erfahrung aus vielen Projekten. Unausgereifte Compiler erschweren erfahrungsgemäß jede Softwareentwicklung und machen eine sicherheitsbezogene Softwareentwicklung generell unmöglich.

Ebenso gibt es bekanntlich gegenwärtig keine Methode, um die Korrektheit für alle Teile der Werkzeuge oder Compiler nachzuweisen.

C.4.4.1 Vergleich des Source-Codes mit dem ausführbaren Code

Ziel: Prüfung, ob die Werkzeuge zur Erzeugung eines PROM-Abbildes keine Fehler in dieses eingebracht haben.

Beschreibung: Das PROM-Abbild wird zurückübersetzt, um die einzelnen Objektmodule zu erhalten. Diese Objektmodule werden in Assemblerdateien zurückübersetzt. Unter Verwendung geeigneter Verfahren werden die rückwärts erzeugten Assemblerdateien mit denjenigen aktuellen Quelldateien verglichen, die ursprünglich zur Erzeugung des PROMs verwendet wurden.

Der Hauptvorteil dieses Verfahrens ist, dass die zur Erzeugung des PROM-Abbildes verwendeten Werkzeuge (Compiler, Linker usw.) nicht für alle Programme validiert werden müssen. Das Verfahren verifiziert, dass die für ein sicherheitsbezogenes System verwendeten Quelldateien korrekt umgesetzt wurden.

Literaturhinweise:

Demonstrating Equivalence of Source Code and PROM Contents. D. J. Pavey and L. A. Winsborrow. The Computer Journal Vol. 36 No. 7, 1993

Formal demonstration of equivalence of source code and PROM contents: an industrial example. D. J. Pavey and L. A. Winsborrow. Mathematics of Dependable Systems, Ed. C. Mitchell and V. Stavridou, Clarendon Press, 1995, ISBN 0-198534-91-4

Assuring Correctness in a Safety Critical Software Application. L. A. Winsborrow and D. J. Pavey. High Integrity Systems, Vol.1, No. 5, pp 453-459, 1996

C.4.5 Geeignete Programmiersprache

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.3 der IEC 61508-3 angeführt.

Ziel: Möglichst weitgehende Unterstützung der Anforderungen dieser Internationalen Norm durch defensive Programmierung, strenge Typisierung, strukturierte Programmierung und möglicherweise Assertions. Die gewählte Programmiersprache sollte mit einem geringen Aufwand leicht zu einem verifizierbarem Code führen und die Programmentwicklung, Verifikation und Pflege erleichtern.

Beschreibung: Die Sprache sollte vollständig und unzweideutig definiert sein. Die Sprache sollte eher anwender- oder problemorientiert als prozessor- oder maschinenorientiert sein. Weitverbreitete Sprachen oder ihre Teilmengen werden gegenüber Sprachen für spezielle Anwendungen bevorzugt.

Zusätzlich zu den bereits angeführten Eigenschaften sollte die Sprache Folgendes bieten:

– Blockstruktur;
– Tests während des Compilerlaufs und
– Tests von Datentypen und Grenzen der Datenfelder während der Laufzeit.

Die Sprache sollte Folgendes unterstützen:

– die Verwendung kleiner und handlicher Softwaremodule;
– Beschränkung des Datenzugriffs in bestimmten Softwaremodulen;
– Definition von eingeschränkten Wertebereichen für Variable und
– alle weiteren Konstrukte zur Fehlereingrenzung.

Wenn der sichere Betrieb des Systems von Echtzeitbedingungen abhängt, sollte die Sprache auch die Behandlung von Interrupts bieten.

Es ist wünschenswert, dass die Sprache von einem geeigneten Compiler, angemessenen Bibliotheken bereits bestehender Softwaremodule, einem Debugger und Werkzeugen zur Versionsverwaltung und Entwicklung unterstützt wird.

Zum Zeitpunkt der Entwicklung dieser Norm war es nicht absehbar, ob objektorientierte Sprachen anderen konventionellen vorgezogen werden.

Sprachmöglichkeiten, die die Verifikation erschweren und deshalb vermieden werden sollten, sind:

– unbedingte Sprünge, außer Unterprogrammaufrufen;
– Rekursion;
– Zeiger, Stapelspeicher oder jede Art von dynamischen Variablen oder Objekten;
– Behandlung von Interrupts auf Ebene des Source-Codes;
– mehrfache Ein- oder Aussprünge von Schleifen, Blöcken oder Unterprogrammen;
– implizite Initialisierung oder Deklaration von Variablen;
– variable Datensätze und Äquivalenzen und
– prozedurale Parameter.

Maschinennahe Sprachen, insbesondere Assemblersprachen, bereiten infolge ihrer Maschinenorientierung Probleme.

Eine wünschenswerte Eigenschaft der Sprache ist es, dass ihr Entwurf und ihre Anwendung zu Programmen führt, deren Ausführung vorhersagbar ist. Bei einer geeignet definierten Programmiersprache gibt es eine Teilmenge, die gewährleistet, dass die Programmausführung vorhersagbar ist. Diese Teilmenge kann (im Allgemeinen) nicht statisch bestimmt werden, obwohl viele statische Einschränkungen bei einer vorhersagbaren Ausführung helfen mögen. Dies erfordert typischerweise einen Nachweis, dass die Indizes von Datenfeldern innerhalb ihrer Grenzen sind, numerischer Überlauf nicht auftreten kann usw.

Tabelle C.1 gibt Empfehlungen für bestimmte Programmiersprachen.

Literaturhinweise:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

IEC 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

ISO/IEC 1539-1: 2004, *Information technology – Programming languages – Fortran – Part 1: Base language*

ISO/IEC 7185:1990, *Information technology – Programming languages – Pascal*

ISO/IEC 8652:1995, *Information technology – Programming languages – Ada*

ISO/IEC 9899:1999, *Programming languages – C*

ISO/IEC 10206:1991, *Information technology – Programming languages – Extended Pascal*

ISO/IEC 10514-1:1996, *Information technology – Programming languages – Part 1: Modula-2, Base Language*

ISO/IEC 10514-3:1998, *Information technology – Programming languages – Part 3: Object Oriented Modula-2*

ISO/IEC 14882:2003, *Programming languages – C++*

ISO/IEC/TR 15942:2000, *Information technology – Programming languages – Guide for the use of the Ada programming language in high integrity systems*

Tabelle C.1 – Empfehlungen für bestimmte Programmiersprachen

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

Tabelle C.1 *(fortgesetzt)*

	Programmiersprache	SIL1	SIL2	SIL3	SIL4
18	Funktionsbausteinsprache mit definierter Sprachenteilmenge	++	++	++	++
19	Strukturierter Text	+	+	+	+
20	Strukturierter Text mit definierter Sprachenteilmenge	++	++	++	++
21	Ablaufsprache	+	+	+	+
22	Ablaufsprache mit definierter Sprachenteilmenge	++	++	++	++
23	Anweisungsliste	+	o	--	--
24	Anweisungsliste mit definierter Sprachenteilmenge	++	+	+	+

ANMERKUNG 1 Die Empfehlungen ++, +, o und -- werden im Anhang A der IEC 61508-3 erklärt.

ANMERKUNG 2 Systemsoftware beinhaltet das Betriebssystem, Treiber, eingebettete Funktionen und Softwaremodule, die als Teil des Systems bereitgestellt werden. Die Software wird typischerweise von dem Hersteller des Sicherheitssystem bereitgestellt. Die Sprachenteilmenge sollte sorgfältig ausgewählt werden, um komplexe Strukturen zu vermeiden, aus denen Fehler in der Implementierung folgen könnten. Die richtige Anwendung der Sprachenteilmenge sollte getestet werden.

ANMERKUNG 3 Die Anwendungssoftware ist die Software, die für eine bestimmte Sicherheitsanwendung entwickelt wird. In vielen Fällen wird diese Software durch den Endanwender oder durch ein anwendungsorientiertes Ingenieurbüro entwickelt. Hat eine Anzahl von Programmiersprachen die gleichen Empfehlungen, dann sollte der Entwickler diejenige auswählen, die gewöhnlich in der Branche oder vom Endanwender verwendet wird. Die Sprachenteilmenge sollte sorgfältig ausgewählt werden, um komplexe Strukturen zu vermeiden, die zu Fehlern führen könnten. Die richtige Anwendung der Sprachenteilmenge sollte getestet werden.

ANMERKUNG 4 Wird eine bestimmte Sprache in der Tabelle nicht aufgelistet, muss nicht angenommen werden, dass diese ausgeschlossen ist. Sie sollte jedoch dieser Internationalen Norm entsprechen.

ANMERKUNG 5 Es gibt mehrere Erweiterungen der Sprache Pascal einschließlich Free Pascal. Verweise auf Pascal schließen diese Erweiterungen ein.

ANMERKUNG 6 Java wurde einschließlich einer automatischen Speicherbereinigung (en: garbage collection) entworfen. Eine Teilmenge von Java kann ohne Erfordernis der automatischen Speicherbereinigung definiert werden. Einige Implementierungen von Java bieten eine progressive automatische Speicherbereinigung, die freien Speicher zurückgewinnt, indem das Programm ausgeführt und das Anhalten der Ausführung für eine Zeitdauer verhindert wird, wenn der verfügbare Speicher ausgeschöpft ist. Strenge Echtzeitanwendungen sollten keine Form der automatischen Speicherbereinigung verwenden.

ANMERKUNG 7 Wenn die Java-Implementierung einen Laufzeit-Interpreter des Java-Zwischencodes verlangt, dann muss der Interpreter als Teil der sicherheitsrelevanten Software und in Übereinstimmung mit den Anforderungen der IEC 61508-3 behandelt werden.

ANMERKUNG 8 Für die Eintragungen 15-24 siehe IEC 61131-3.

C.4.6 Automatische Softwaregenerierung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.2 der IEC 61508-3 angeführt.

Ziel: Automatisierung der fehleranfälligeren Aufgaben der Softwareimplementierung.

Beschreibung: Der Systementwurf wird mit einem Modell (einer ausführbaren Spezifikation) auf einer höheren Abstraktionsebene beschrieben als der herkömmlich ausführbare Code. Das Modell wird durch einen Codegenerator automatisch in die ausführbare Form überführt. Das Ziel ist, die Softwarequalität durch Beseitigung der fehleranfälligen manuellen Aufgaben beim Codieren zu verbessern. Ein weiterer möglicher Vorteil ist, dass komplexere Entwürfe auf einer höheren abstrakten Ebene durchgeführt werden können.

Literaturhinweise:

Embedded Software Generation from System Level Design Languages, H Yu, R. Domer, D. Gajski. In „ASP-DAC 2004: Proceedings of the ASP-Dac 2004 Asia and South Pacific Design Automation Conference, 2004“, IEEE Circuits and Systems Society. IEEE, 2004, ISBN 0780381750, 9780780381759

Transforming Process Algebra Models into UML State Machines: Bridging a Semantic Gap?. M.F. van Amstel et. al. In Theory and Practice of Model Transformations: First International Conference, ICMT". ed. A. Vallecillo. Springer, 2008, ISBN 3540699260, 9783540699262

C.4.7 Testmanagement und Automatisierungswerkzeuge

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-3 angeführt.

Ziel: Förderung des systematischen und gründlichen Ansatzes der Software- und Systemtests.

Beschreibung: Die Verwendung von geeigneten unterstützenden Werkzeugen automatisiert die arbeitsintensiveren und fehleranfälligeren Aufgaben bei der Systementwicklung und bewirkt die Fähigkeit zum systematischen Ansatz des Testmanagements. Die Verfügbarkeit der Unterstützung fördert einen gründlicheren Ansatz sowohl der normalen Tests als auch der Regressionstests.

Literaturhinweis:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R.Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986

C.5 Verifikation und Modifikation

C.5.1 Statistisches Testen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.5, C.15, A.7 und C.17 der IEC 61508-3 angeführt.

Ziel: Erreichen einer quantitativen Aussage über die Zuverlässigkeit der untersuchten Software.

Beschreibung: Diese Methode ermittelt eine statistische Abschätzung der Softwarezuverlässigkeit. Eine quantitative Aussage kann die zugehörigen Vertrauens- und Signifikanzniveaus in Betracht ziehen und Folgendes ergeben:

– eine Wahrscheinlichkeit eines Versagens bei Anforderung;
– eine Wahrscheinlichkeit eines Versagens während einer bestimmten Zeit; und
– eine Wahrscheinlichkeit eines Fehlereinschlusses.

Von diesen Aussagen können weitere Parameter abgeleitet werden, wie:

– Wahrscheinlichkeit einer versagensfreien Ausführung;
– Überlebenswahrscheinlichkeit;
– Verfügbarkeit;
– MTBF oder Versagens(Ausfall-)rate; und
– Wahrscheinlichkeit einer sicheren Ausführung.

Wahrscheinlichkeitsbetrachtungen basieren entweder auf einem statistischen Test oder auf Betriebserfahrung. Normalerweise ist die Anzahl von Testfällen oder beobachteten Einsätzen sehr groß. Typischerweise benötigt der Test der Betriebsart mit einer Anforderungsrate beträchtlich weniger Zeit als der Test der Betriebsart mit kontinuierlicher Anforderung.

Zur Bereitstellung von Testdaten und Überwachung der Testausgaben werden normalerweise automatisierte Testwerkzeuge verwendet. Umfangreiche Tests werden auf großen Zentralrechnern durchgeführt, die über eine angemessene Peripherie zur Prozesssimulation verfügen. Testdaten werden sowohl unter systematischen wie auch zufälligen Hardware-Gesichtspunkten ausgewählt. Zum Beispiel stellt die gesamte Teststeuerung ein Testdatenprofil sicher, während eine zufällige Auswahl individuelle Testfälle im Einzelnen bestimmen kann.

Individuelle Prüfeinrichtungen, Testausführungen und Testüberwachungen werden nach den detaillierten, oben beschriebenen Testzielen festgelegt. Andere wichtige Bedingungen ergeben sich durch die mathematischen Voraussetzungen, die erfüllt werden müssen, damit die Testauswertung ihren Zweck erfüllt.

Aussagen über das wahrscheinliche Verhalten eines Testobjektes können auch von der Betriebserfahrung abgeleitet werden. Vorausgesetzt, es treffen die gleichen Bedingungen zu, kann die gleiche mathematische Berechnung bezüglich der Auswertung der Testergebnisse angewendet werden.

Praktisch ist es nach diesem Verfahren sehr schwierig, einen extrem hohen Zuverlässigkeitsgrad nachzuweisen.

Literaturhinweise:

A discussion of statistical testing on a safety-related application. S Kuball, J H R May, Proc IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007

Estimating the Probability of Failure when Testing Reveals No Failures, W.K. Miller, L.J. Morell, et al.. IEEE Transactions on Software Engineering, VOl. 18, NO.1, pp33-43, January 1992

Reliability estimation from appropriate testing of plant protection software, J. May, G. Hughes, A.D. Lunn. IEE Software Engineering Journal, v10 n6 pp 206-218, Nov 1995 (ISSN: 0268-6961)

Validation of ultra high dependability for software based systems, B. Littlewood and L. Strigini. Comm. ACM 36 (11), 69-80, 1993

C.5.2 Datenaufzeichnung und Analyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.5 und A.8 der IEC 61508-3 angeführt.

Ziel: Dokumentation aller Daten, Entscheidungen und Grundlagen des Softwareprojekts, um eine einfachere Verifikation, Validierung, Beurteilung und Instandhaltung zu ermöglichen.

Beschreibung: Während eines Projekts erfolgt eine detaillierte Dokumentation, die Folgendes enthalten könnte:

– die zu jedem Softwaremodul ausgeführten Tests;
– Entscheidungen und ihre Grundlagen;
– Probleme und ihre Lösungen.

Während und zum Ende des Projekts kann diese Dokumentation analysiert werden, um eine breite Vielfalt von Informationen zu erreichen. Besonders wichtig für die Instandhaltung von Rechnersystemen ist die Datenaufzeichnung als Grundlage für bestimmte Entscheidungen, die während des Entwicklungsprojektes gefällt worden sind, da sie dem Instandhaltungspersonal nicht immer bekannt sind.

Literaturhinweis:

Dependability of Critical Computer Systems 2. F. J. Redmill, Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811

C.5.3 Schnittstellenprüfung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-3 angeführt.

Ziel: Erkennung von Fehlern in den Schnittstellen der Unterprogramme.

Beschreibung: Mehrere Grade der Detaillierung oder Vollständigkeit der Tests sind durchführbar. Die wichtigsten Grade sind Tests

– aller Schnittstellenvariablen mit ihren Extremwerten;

– aller Schnittstellenvariablen individuell bei ihren Extremwerten mit anderen Schnittstellenvariablen bei Normalwerten;
– aller Werte des Gültigkeitsbereichs jeder Schnittstellenvariablen mit anderen Schnittstellenvariablen bei Normalwerten;
– aller Werte aller Variablen in Kombination (dies wird nur für kleine Schnittstellen durchführbar sein);
– der für jeden Aufruf jedes Unterprogramms relevanten festgelegten Testbedingungen.

Diese Tests sind insbesondere wichtig, wenn Schnittstellen keine Assertions beinhalten, die inkorrekte Parameterwerte erkennen. Sie sind auch wichtig, nachdem neue Konfigurationen bereits vorhandener Unterprogramme erzeugt worden sind.

C.5.4 Durchführung von Testfällen nach einer Grenzwertanalyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2, B.3 und B.8 der IEC 61508-3 angeführt.

Ziel: Erkennung von Softwarefehlern an Parameterbegrenzungen oder Grenzen.

Beschreibung: Der Eingabebereich des Programms wird in eine Anzahl von Eingabeklassen entsprechend der Äquivalenzbeziehung (siehe C.5.7) aufgeteilt. Die Tests sollten die Grenzen und die Extremwerte der Klassen abdecken. Die Tests kontrollieren, dass die Grenzen des Eingabebereichs laut Spezifikation mit jenen des Programms übereinstimmen. Die Verwendung des Wertes Null, in direkter wie auch indirekter Verwendung, führt leicht zu Fehlern und erfordert spezielle Aufmerksamkeit:

– Divisor Null;
– ASCII-Leerzeichen;
– leerer Stack oder leeres Listenelement;
– volle Matrix;
– leerer Tabelleneintrag.

Normalerweise haben die Eingabegrenzen eine direkte Beziehung zu den Grenzen des Ausgabebereiches. Es sollten Testfälle geschrieben werden, um die Ausgaben an ihre Grenzwerte zu zwingen. Ebenso ist zu betrachten, ob möglicherweise ein Testfall festgelegt werden kann, der Ausgaben über die spezifizierten Grenzwerte hinaus erzwingt.

Wenn die Ausgabe eine Folge von Daten ist, zum Beispiel eine gedruckte Tabelle, sollte dem ersten und letzten Element sowie Listen, die kein, ein oder zwei Elemente enthalten, besondere Aufmerksamkeit gewidmet werden.

Literaturhinweis:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.5 Durchführung von Testfällen aus der Fehlererwartung („Fehler erraten“)

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.8 der IEC 61508-3 angeführt.

Ziel: Beseitigung allgemeiner Programmierirrtümer.

Beschreibung: Testerfahrung und Intuition, kombiniert mit Kenntnis und Neugierde über das zu testende System, kann einige untypische Testfälle zu der entworfenen Auswahl von Testfällen hinzufügen.

Spezielle Werte oder Kombinationen von Werten können anfällig für Fehler sein. Einige interessante Testfälle können von Inspektions-Checklisten abgeleitet werden. Es kann auch betrachtet werden, ob das System robust genug ist. Können zum Beispiel die Druckknöpfe auf dem Bedientableau zu schnell oder zu oft betätigt werden? Was geschieht, wenn zwei Druckknöpfe gleichzeitig betätigt werden?

Literaturhinweis:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.6 Durchführung von Testfällen nach Fehlereinpflanzung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.2 der IEC 61508-3 angeführt.

Ziel: Feststellen, ob eine Auswahl von Testfällen angemessen ist.

Beschreibung: Einige bekannte Arten von Irrtümern werden in das Programm eingebracht (gesetzt), und das Programm wird mit den Testfällen unter Testbedingungen ausgeführt. Wenn nur einige der eingebrachten Fehler aufgedeckt werden, ist die Auswahl der Testfälle nicht angemessen. Das Verhältnis der aufgedeckten eingebrachten Fehler zu der gesamten Anzahl eingebrachter Fehler ist eine Abschätzung des Verhältnisses der aufgedeckten wirklichen Fehler und der gesamten Anzahl der Fehler. Dies ermöglicht eine Abschätzung der Anzahl verbleibender Fehler und somit des verbleibenden Testaufwands.

$$\frac{\textit{aufgedeckte eingebrachte Fehler}}{\textit{gesamte Anzahl eingebrachter Fehler}} = \frac{\textit{aufgedeckte wirkliche Fehler}}{\textit{gesamte Anzahl der wirklichen Fehler}}$$

Die Erkennung aller eingebrachten Fehler kann entweder anzeigen, dass die Auswahl von Testfällen angemessen war oder dass die eingebrachten Fehler zu leicht aufzudecken waren. Eine Einschränkung der Methode ist, dass die Arten der Irrtümer wie auch die Positionen der Einbringungen die statistische Verteilung realer Fehler wiedergeben müssen, um irgendwelche brauchbaren Ergebnisse zu liefern.

Literaturhinweise:

Software Fault Injection: Inoculating Programs Against Errors. J. Voas, G. McGraw. Wiley Computer Pub., 1998, ISBN 0471183814, 9780471183815

Faults, Injection Methods, and Fault Attacks. Chong Hee Kim, Jean-Jacques Quisquater, IEEE Design and Test of Computers, vol. 24, no. 6, pp. 544-545, Nov., 2007

Fault seeding for software reliability model validation. A. Pasquini, E. De Agostino. Control Engineering Practice, Volume 3, Issue 7, July 1995. Elsevier Science Ltd

C.5.7 Äquivalenzklassen und Test von Partitionen des Eingangsbereichs

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.3 der IEC 61508-3 angeführt.

Ziel: Angemessener Test der Software unter Benutzung einer minimalen Anzahl von Testdaten. Die Testdaten erhält man durch Auswahl der Partitionen des Gültigkeitsbereichs der Eingaben, die zur Ausführung der Software erforderlich sind.

Beschreibung: Diese Teststrategie basiert auf der Äquivalenzbeziehung der Eingaben, die eine Partition des Gültigkeitsbereichs der Eingaben bestimmt.

Es werden Testfälle mit dem Ziel ausgewählt, alle vorher festgelegten Partitionen abzudecken. Mindestens ein Testfall wird aus jeder Äquivalenzklasse ausgewählt.

Es gibt zwei grundlegende Möglichkeiten, die Eingaben aufzuteilen:

– Äquivalenzklassen abgeleitet von der Spezifikation. Die Interpretation der Spezifikation kann entweder eingabeorientiert sein, zum Beispiel werden die ausgewählten Werte alle gleich behandelt, oder ausgabeorientiert, zum Beispiel führt eine Auswahl von Werten zu dem gleichen funktionalen Ergebnis;

– Äquivalenzklassen abgeleitet von der internen Programmstruktur. Die Ergebnisse der Äquivalenzklassen werden aus der statischen Analyse des Programms abgeleitet, zum Beispiel führt eine Auswahl von Werten zu dem gleichen ausgeführten Programmpfad.

Literaturhinweise:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0

C.5.8 Strukturabhängige Tests

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.2 der IEC 61508-3 angeführt.

Ziel: Anwendung von Tests, die bestimmte Teilbereiche der Programmstruktur prüfen.

Beschreibung: Basierend auf einer Programmanalyse wird eine Auswahl von Eingabedaten so ausgewählt, dass ein großer (und oft vorher festgelegter) Prozentsatz des Programmcodes geprüft wird. Maßnahmen zur Codeabdeckung werden wie folgt in Abhängigkeit von dem geforderten Schärfegrad gewählt. In allen Fällen sollten 100 % der ausgewählten Abdeckungsmetrik das Ziel sein. Wenn es nicht möglich ist, eine 100 %-Abdeckung zu erreichen, sollten die Gründe, warum 100 % nicht erreicht werden können, im Prüfbericht dokumentiert werden (zum Beispiel defensiver Code, der nur erreicht werden kann, wenn ein Hardwareproblem entsteht). Besonders die ersten vier Verfahren in der folgenden Liste werden in den Empfehlungen der Tabelle B.3 der IEC 61508-3 erwähnt und werden weithin durch Testwerkzeuge unterstützt. Die restlichen Verfahren könnten auch berücksichtigt werden.

– **Eingangspunkt-Abdeckung (Call Graph):** Stellt sicher, dass jedes Unterprogramm (Subroutine oder Funktion) mindestens einmal aufgerufen worden ist (dies ist die Maßnahme mit der geringsten strengen strukturellen Abdeckung).

  ANMERKUNG In objektorientierten Sprachen kann es mehrere Unterprogramme desselben Namens geben, der für verschiedene Varianten eines polymorphen Typs gilt (überladende Unterprogramme), der durch das dynamische Ausführen aufgerufen werden kann. In diesen Fällen sollte jedes solches überladende Unterprogramm getestet werden.

– **Anweisungen:** Stellt sicher, dass alle Anweisungen im Code mindestens einmal ausgeführt worden sind.

– **Verzweigungen:** Beide Möglichkeiten jeder Verzweigung sollten getestet werden. Bei einigen Arten defensiver Codes kann dies schwierig sein.

– **Zusammengesetzte Bedingungen:** Jede Bedingung einer zusammengesetzten, bedingten Verzweigung (d. h. verbunden durch AND/OR) wird geprüft. Siehe MCDC (modifizierte Abdeckung der Bedingungsentscheidungen (en: modified condition decision coverage), siehe DO178B).

– **LCSAJ:** Eine lineare Codefolge und ein Sprung (en: linear code sequence and jump) stellt eine lineare Folge von Codeanweisungen dar, einschließlich bedingter Anweisungen, die von einem Sprung beendet werden. Viele mögliche Teilpfade können infolge eingeschränkter Eingabedaten, verursacht durch die Ausführung des vorherigen Codes, nicht mehr ausgeführt werden.

– **Datenfluss:** Die Pfade der Ausführung werden auf der Basis der Verwendung der Daten ausgewählt, zum Beispiel ein Pfad, in dem die gleiche Variable beschrieben und gelesen wird.

– **Basis-Pfad:** Die kleinste Menge von endlichen Pfaden vom Start bis zum Ende, von der Art, dass alle Verzweigungen einbezogen sind. (Überdeckende Kombinationen der Pfade in dieser Menge können je-

den Pfad durch den betreffenden Programmteil nachbilden.) Der Test aller derartigen Pfade hat sich als wirksam zur Lokalisierung von Fehlern erwiesen.

Literaturhinweise:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

RTCA, Inc. document DO-178B and EUROCAE document ED-12B, *Software Considerations in Airborne Systems and Equipment Certification*, dated December 1, 1992

C.5.9 Kontrollflussanalyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.8 der IEC 61508-3 angeführt.

Ziel: Erkennung schlechter und möglicherweise inkorrekter Programmstrukturen.

Beschreibung: Die Steuerflussanalyse ist ein statisches Testverfahren zur Aufdeckung verdächtiger Codebereiche, die keiner guten Programmierpraxis folgen. Das Programm wird analysiert, um einen gerichteten Graphen zu erhalten, der weiter analysiert werden kann nach:

– unzugänglichem Code, zum Beispiel bedingungslose Sprünge, die Blöcke von nicht erreichbarem Code hinterlassen;
– verknotetem Code. Gut strukturierter Code hat einen Steuergraphen, der schrittweise auf einen einzelnen Knoten reduzierbar ist. Im Gegensatz dazu kann ein schlecht strukturierter Code nur auf eine Struktur reduziert werden, die aus mehreren Knoten zusammengesetzt ist.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.5.10 Datenflussanalyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.8 der IEC 61508-3 angeführt.

Ziel: Erkennung schlechter und möglicherweise inkorrekter Programmstrukturen.

Beschreibung: Die Datenflussanalyse ist ein statisches Testverfahren, das die durch die Steuerflussanalyse gewonnenen Informationen mit den Informationen darüber kombiniert, welche Variablen in verschiedenen Teilen des Codes gelesen oder geschrieben werden. Die Analyse kann Folgendes testen:

– Variablen, die gelesen werden können, bevor ihnen ein Wert zugewiesen wird. Dies kann vermieden werden, indem immer ein Wert zugewiesen wird, wenn eine neue Variable deklariert wird;
– Variablen, die mehrfach beschrieben werden, ohne gelesen zu werden. Dies könnte auf übersprungenen Code hindeuten;
– Variablen, die beschrieben werden, aber nie gelesen werden. Dies könnte auf redundanten Code hindeuten.

Eine Datenflussanomalie wird nicht immer direkt zu einem Programmfehler führen, aber wenn Anomalien vermieden werden, ist es weniger wahrscheinlich, dass der Code Fehler beinhaltet.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.5.11 Symbolische Ausführung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.8 der IEC 61508-3 angeführt.

Ziel: Darlegung der Übereinstimmung zwischen dem Source-Code und der Spezifikation.

Beschreibung: Die Programmvariablen werden nach Austausch der linken Seite mit der rechten Seite in allen Zuweisungen ausgewertet. Bedingte Verzweigungen und Schleifen werden in Boolesche Ausdrücke übersetzt. Das abschließende Ergebnis ist ein symbolischer Ausdruck jeder Programmvariablen. Dieser Ausdruck entspricht einer Formel für den Wert, den das Programm mit realen Daten berechnen würde. Dieses kann gegen den erwarteten Ausdruck getestet werden.

Eine ähnliche Verwendung der symbolischen Ausführung ist die systematische Erzeugung von Testdaten für die Pfade durch die Programm-Logik. Die Möglichkeit der symbolischen Ausführung kann in einem einheitlichen Werkzeug vereinigt werden, um eine Möglichkeit für den Test und für die Codeanalyse der Softwareelemente zur Verfügung zu stellen.

Literaturhinweise:

Using symbolic execution for verifying safety-critical systems. A. Coen-Porisini, G. Denaro, C. Ghezzi, M. Pezzé. Proceedings of the 8th European software engineering conference, and 9th ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2001, ISBN:1-58113-390-1

Using symbolic execution to guide test generation. G. Lee, J. Morris, K. Parker, G. Bundell, P. Lam. In Software Testing, Verification and Reliability, vol 15, no 1, 2005. John Wiley & Sons, Ltd

C.5.12 Formaler Beweis (Verifikation)

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.5 und A.9 der IEC 61508-3 angeführt.

Ziel: Beweis der Korrektheit eines Programms in Bezug auf ein abstraktes Modell des Programms unter Verwendung theoretischer und mathematischer Modelle und Regeln.

Beschreibung: Das Prüfen ist eine gewöhnliche Maßnahme, um die Korrektheit eines Programms zu untersuchen. Jedoch ist die vollständige Prüfung wegen der Komplexität von Programmen mit praktischem Nutzen im Allgemeinen unerreichbar. Deshalb kann nur ein Bruchteil des möglichen Programm-Verhaltens auf diese Weise untersucht werden. Im Gegensatz hierzu wendet die formale Verifikation mathematische Operationen auf eine mathematische Darstellung eines Programms an, um festzustellen, ob sich das Programm für alle möglichen Eingaben wie definiert verhält.

Die formale Verifikation eines Systems benötigt ein abstraktes Modell des Programms und seines erforderlichen Verhaltens (d. h. eine Spezifikation) in einer Sprache mit einer genauen mathematischen Bedeutung. Die Spezifikation kann vollständig sein oder sie kann auf spezielle Programm-Eigenschaften eingeschränkt werden:

– funktionale Eigenschaften der Korrektheit, d. h. das Programm sollte eine besondere Funktionalität aufweisen;

– Eigenschaften der Sicherheit (d. h. irgendein schlechtes Verhalten wird nie vorkommen) und der Lebendigkeit (d. h. irgendein gutes Verhalten wird irgendwann einmal vorkommen);
– Eigenschaften des zeitlichen Verhaltens, d. h. irgendein Verhalten wird zu einer bestimmten Zeit vorkommen.

Das Ergebnis der formalen Verifikation ist ein entscheidender Beweis, ob das abstrakte Modell des Programms in Bezug auf die Spezifikation für alle möglichen Eingaben korrekt ist, d. h. das Modell genügt den festgelegten Eigenschaften.

Die Korrektheit des Modells beweist jedoch nicht direkt die Korrektheit des tatsächlichen Programms. Ein weiterer notwendiger Schritt ist der Nachweis, dass das Modell eine genaue Abstraktion des tatsächlichen Programms bezüglich der interessanten Eigenschaften ist. Einige Programm-Eigenschaften von praktischem Interesse können nicht mathematisch formalisiert werden (z. B. meistens Zeit- und Ablaufplanung, subjektive Eigenschaften wie eine „klare und einfache“ Benutzerschnittstelle, oder gewiss jede Eigenschaft oder jedes Entwurfsziel, das nicht leicht in einer formale Sprache ausgedrückt werden kann). Die formale Verifikation ersetzt deshalb nicht völlig Simulation und Prüfung, aber ergänzt stattdessen diese Verfahren zur Bereitstellung weiterer Nachweise, um den korrekten Betrieb des Programms für alle Eingaben zu bestätigen. Während die formale Verifikation die Korrektheit eines abstrakten Modells eines Programms gewährleisten kann, gewährleistet das Prüfen, dass sich das tatsächliche Programm wie erwartet verhält.

Die Anwendung der formalen Verifikation in der Entwurfsphase kann die Entwicklungsdauer deutlich reduzieren, indem bedeutende Fehler und Flüchtigkeitsfehler früh in der Entwurfsphase aufgedeckt werden. Somit wird die Zeit für die erforderlichen Wiederholungen zwischen Entwurf und Prüfung reduziert.

Es werden mehrere formale Methoden in praktischer Verwendung in C.2.4 beschrieben, zum Beispiel: CCS, CSP, HOL, LOTOS, OBJ, Temporäre Logik, VDM und Z.

C.5.12.1 Modellprüfung

Die Modellprüfung ist eine Methode zur formalen Verifikation von reaktiven und gleichlaufenden Systemen. In Anbetracht einer endlichen Zustandsstruktur, die das Systemverhalten beschreibt, wird eine als temporale logische Formel niedergeschriebene Eigenschaft gegen die Struktur überprüft, ob sie sie aufweist oder nicht. Effiziente Algorithmen (z. B. SPIN, SMV und UPPAAL) werden eingesetzt, um alle Zustände der Struktur automatisch und erschöpfend zu durchlaufen. Wird eine Eigenschaft nicht aufgewiesen, wird ein Gegenbeispiel erzeugt. Es zeigt, wie die Eigenschaft in der Struktur verletzt wird, und enthält sehr nützliche Informationen, um das System zu untersuchen. Die Modellprüfung kann verborgene Programmfehler aufdecken, die der traditionellen Inspektion und Prüfung entgehen könnten.

Es ist zu beachten, dass die Modellprüfung bei der Analyse tiefsinniger Komplexität hilfreich ist. Dies kann in einigen Anwendungen mit niedrigem SIL nützlich sein, aber Vorsicht ist angebracht, wenn tiefsinnige Komplexitäten in Anwendungen mit hohem SIL vorhanden sind.

Literaturhinweise:

Is Proof More Cost-Effective Than Testing?. S. King, R. Chapman, J. Hammond, A. Pryor. IEEE Transactions on Software Engineering, vol. 26 no. 8, August 2000

Model Checking. E. M. Clarke, O. Grumberg, and D. A. Peled. MIT Press, 1999, ISBN 0262032708, 9780262032704

Systems and Software Verification: Model-Checking Techniques and Tools. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. Mckenzie, Springer, 2001, ISBN 3-540-41523-8

Logic in Computer Science: Modelling and Reasoning about Systems. M.Huth and M. Ryan. Cambridge University Press, 2000, ISBN 0521652006, 0521656028

The Spin Model Checker: Primer and Reference Manual. G. J. Holzmann. Addison-Wesley, 2003, ISBN 0321228626, 9780321228628

C.5.12.2 (leer)

C.5.13 Softwarekomplexitätskontrolle

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.9 und C.19 der IEC 61508-3 angeführt.

Ziel: Vorhersage der Programmmerkmale aus den Softwareeigenschaften selbst oder aus ihrer Entwicklung oder Testaufzeichnung.

Beschreibung: Diese Modelle bewerten einige strukturelle Eigenschaften der Software und bringen diese in Verbindung zu einem gewünschten Merkmal, wie z. B. Zuverlässigkeit oder Komplexität. Für die Bewertung der meisten Maßzahlen sind Softwarewerkzeuge erforderlich. Einige der anwendbaren Metriken sind unten beschrieben:

– Theoretische Komplexität – Diese Maßzahl kann frühzeitig im Lebenszyklus angewendet werden, um Einschränkungen zu bewerten. Sie basiert auf der Komplexität des Kontrollgraphen, dargestellt durch die zyklomatische Zahl;

– Anzahl von Pfaden, um ein bestimmtes Softwaremodul zu aktivieren (Zugänglichkeit) – je häufiger ein Softwaremodul erreicht werden kann, desto besser kann es ausgetestet werden;

– Wissenschaft der Metriken nach Halstead – diese Maßzahl errechnet die Programmlänge durch Zählen der Operatoren und Operanden. Sie stellt eine Maßzahl der Komplexität und Größe bereit, die eine Basis zum Vergleich bildet, wenn zukünftige Entwicklungsressourcen abgeschätzt werden;

– Anzahl der Ein- und Aussprünge je Softwaremodul – Die Verringerung der Anzahl von Ein- und Aussprüngen ist ein Hauptmerkmal eines strukturierten Entwurfs- und Programmierverfahrens.

Literaturhinweis:

Metrics and Models in Software Quality Engineering. S.H. Kan. Addison-Wesley, 2003, ISBN 0201729156, 9780201729153

C.5.14 Formale Inspektion

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.8 der IEC 61508-3 angeführt.

Ziel: Aufdeckung von Mängeln in einem Softwareelement.

Beschreibung: Die formale Inspektion ist ein strukturierter Prozess, um Software zu untersuchen. Sie wird von Personen mit gleichem technischen Sachverstand wie die, die die Software erstellt haben, ausgeführt, um Mängel zu finden und dem Ersteller zu ermöglichen, die Software zu verbessern. Der Ersteller sollte nicht an der Inspektion teilnehmen, außer um die Inspektoren während der Einarbeitungsphase einzuweisen. Formale Inspektionen können für bestimmte Softwareelemente, die in irgendeiner Phase des Software-Entwicklungslebenszyklus erstellt wurden, ausgeführt werden.

Vor der durchzuführenden Inspektion sollten die Inspektoren mit der zu untersuchenden Software vertraut werden. Die Aufgaben der Inspektoren bei der Inspektion sollten klar definiert werden. Eine Agenda der Inspektion sollte vorbereit werden. Die Start- und Abschlusskriterien sollten basierend auf den für das Softwareelement erforderlichen Eigenschaften definiert werden. Startkriterien sind die Kriterien oder Anforderungen, die vor der durchzuführenden Inspektion erfüllt werden müssen. Abschlusskriterien sind die Kriterien oder Anforderungen, die zum Abschluss eines bestimmten Prozesses erfüllt werden müssen.

Während der Inspektion sollten die Ergebnisse der Inspektion durch den Moderator formal dokumentiert werden, dessen Aufgabe die Erleichterung der Inspektion ist. Von allen Inspektoren sollte Einigkeit bezüglich der Ergebnisse erreicht werden. Die Mängel sollten entweder als a) erforderliche Verbesserung vor der Abnahme oder als b) erforderliche Verbesserung zu einem vorgegebenen Zeitpunkt bzw. Meilenstein klassifiziert werden. Nach Abschluss der Inspektion sollten dem Ersteller erkannte Mängel zur darauf folgenden Verbesserung offengelegt werden. Abhängig von der Anzahl und dem Anwendungsbereich der erkannten Mängel kann der Moderator festlegen, ob eine weitere Inspektion der Software notwendig ist.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Fagan, M. *Design and Code Inspections to Reduce Errors in Program Development.* IBM Systems Journal 15, 3 (1976): 182-211

C.5.15 Walk-through (Software)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.8 der IEC 61508-3 angeführt.

Ziel: Aufdeckung von Unstimmigkeiten zwischen der Spezifikation und der Ausführung.

Beschreibung: Der Walkthrough ist ein informelles Verfahren, das vom Ersteller eines Softwareelements in Gegenwart von Personen mit gleichem technischen Sachverstand mit dem Ziel ausgeführt wird, Mängel in dem Softwareelement zu finden. Er kann für bestimmte Softwareelemente, die in irgendeiner Phase des Software-Entwicklungslebenszyklus erstellt wurden, ausgeführt werden.

Festgelegte Funktionen des sicherheitsbezogenen Systems werden untersucht und bewertet, um sicherzustellen, dass das sicherheitsbezogene System den in der Spezifikation gegebenen Anforderungen entspricht. Irgendwelche Schwachpunkte bezüglich der Ausführung und der Verwendung des Produktes werden so dokumentiert, dass sie beseitigt werden können. Im Gegensatz zu einer formalen Inspektion ist der Autor während des Walkthrough-Verfahrens aktiv.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.16 Entwurfsüberprüfung

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.8 der IEC 61508-3 angeführt.

Ziel: Aufdeckung von Mängeln im Softwareentwurf.

Beschreibung: Eine Entwurfsüberprüfung ist eine formale, dokumentierte, umfassende und systematische Überprüfung des Softwareentwurfs, um die Anforderungen an den Entwurf und die Fähigkeit des Entwurfs, diesen Anforderungen zu entsprechen, zu bewerten, Probleme zu identifizieren und Lösungen vorzuschlagen.

Entwurfsüberprüfungen stellen die Mittel zur Verfügung, den Status des Entwurfs gegen die Eingangsanforderungen und die Mittel zur Identifikation von weiteren Verbesserungsmöglichkeiten zu bewerten. Während des Fortschritts der Tätigkeiten im Entwicklungslebenszyklus und beim Erreichen der wichtigsten Meilensteinen des Entwurfs sollten Entwurfsüberprüfungen durchgeführt werden, um alle Schnittstellen-Aspekte zu überprüfen, um sicherzustellen, dass der Entwurf verifiziert werden kann, um sicherzustellen, dass der Entwurf seinen Anforderungen entspricht, und um sicherzustellen, dass der geeigneteste Entwurf mit den Anfor-

derungen an die Sicherheit übereinstimmt. Solch eine Überprüfung beabsichtigt in erster Linie, die Arbeit der Entwickler zu verifizieren, und sollte als Tätigkeit zur Bestätigung und Verbesserung behandelt werden.

Ein strenges Inspektionsverfahren wie die „Nebenpfadanalyse“ kann verwendet werden, um fehlerhaftes Verhalten der Software, wie eine unerwarteter Pfadabfolge oder ein unerwarteter logischer Fluss, unbeabsichtigte Ausgaben, fehlerhafte Zeitabläufe und unerwünschte Handlungen, aufzudecken.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

IEC 61160:2005, *Design Review*

Space Product Assurance, Sneak analysis - Part 2: Clue list. ECSS-Q-40-04A Part 2. ESA Publications Division, Noordwijk, 1997, ISSN 1028-396X, http://www.everyspec.com/ESA/ECSS-Q-40-04A_Part-2_14981/

C.5.17 Prototypenerstellung/Animation

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.3 und B.5 der IEC 61508-3 angeführt.

Ziel: Test der Machbarkeit der Implementierung eines Systems gegen vorhandene Randbedingungen. Interpretation des Systems aus der Sicht der Spezifikationen für den Kunden, um Missverständnisse aufzudecken.

Beschreibung: Eine Teilmenge von Systemfunktionen, Randbedingungen und Anforderungen an die Leistungsfähigkeit wird ausgewählt. Ein Prototyp wird unter Verwendung von höherwertigen Werkzeugen erstellt. In diesem Stadium brauchen Randbedingungen wie der Zielrechner, die Implementierungssprache, die Programmgröße, die Instandhaltbarkeit, die Zuverlässigkeit und die Verfügbarkeit nicht betrachtet zu werden. Der Prototyp wird nach den Kriterien des Kunden ausgewertet und die Anforderungen an das System können daraufhin modifiziert werden.

Literaturhinweis:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.18 Simulation des Prozesses

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.7, C.7, B.3 und C.13 der IEC 61508-3 angeführt.

Ziel: Test der Funktion eines Softwaresystems zusammen mit seiner Schnittstelle zur Umgebung, ohne die wirkliche Umgebung in irgendeiner Art zu beeinflussen.

Beschreibung: Die Erstellung eines Systems nur zu Testzwecken, welches das Verhalten der EUC-Einrichtung (EUC) imitiert.

Die Simulation kann durch Software oder eine Kombination von Software und Hardware erfolgen. Sie muss

– Eingaben bereitstellen, die gleichwertig zu den Eingaben sind, die in einer tatsächlich installierten EUC-Einrichtung vorhanden sein werden;
– auf Ausgaben der zu testenden Software so reagieren, dass die gesteuerte Anlage glaubwürdig nachgebildet wird;

– Möglichkeiten für Eingaben des Bedieners anbieten, um alle durch das getestete System zu bewältigenden Störungen darzustellen.

Wenn Software getestet wird, kann die Simulation eine Simulation der Zielhardware mit ihren Ein- und Ausgängen sein.

Literaturhinweise:

EmStar: An Environment for Developing Wireless Embedded Systems Software. J Elson et al. http://cens.ucla.edu/TechReports/9_emstar.pdf

A hardware-software co-simulator for embedded system design and debugging. A. Ghosh et al. In Proceedings of the IFIP International Conference on Computer Hardware Description Languages and Their Applications, IFIP International Conference on Very Large Scale Integration, 1995. IEEE, 1995, ISBN 4930813670, 9784930813671

C.5.19 Anforderungen an die Leistungsfähigkeit

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.6 der IEC 61508-3 angeführt.

Ziel: Erstellung nachweisbarer Anforderungen an die Leistungsfähigkeit eines Softwaresystems.

Beschreibung: Die Spezifikation der Anforderungen, sowohl an das System als auch an die Software, werden analysiert, um alle Anforderungen an die Leistungsfähigkeit zu ermitteln.

Jede Anforderung an die Leistungsfähigkeit wird der Reihe nach untersucht, um

– die zu erreichenden Erfolgskriterien zu bestimmen;
– zu bestimmen, ob es ein Maß für die Erfolgskriterien gibt;
– die mögliche Genauigkeit solcher Messungen zu bestimmen;
– die Projektstufen, an denen die Messergebnisse geschätzt werden können, zu bestimmen und
– die Projektstufen, an denen die Messungen erfolgen können, zu bestimmen.

Danach wird die Durchführbarkeit jeder Anforderung an die Leistungsfähigkeit analysiert, um eine Liste von Anforderungen an die Leistungsfähigkeit, Erfolgskriterien und möglichen Maßen zu erhalten. Die Hauptziele sind:

– jede Anforderung an die Leistungsfähigkeit wird mit mindestens einem Maß in Bezug gebracht;
– wenn möglich, sind genaue und wirkungsvolle Maße auszuwählen, die so früh wie möglich in der Entwicklung verwendet werden können;
– unumgängliche und zusätzlich mögliche Anforderungen an die Leistungsfähigkeit und Erfolgskriterien sind festzulegen; und
– wenn möglich, sollte eine Messung für mehrere Anforderungen an die Leistungsfähigkeit verwendet werden.

Literaturhinweis:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.20 Modellierung der Leistungsfähigkeit

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen B.2 und B.5 der IEC 61508-3 angeführt.

Ziel: Sicherstellen, dass die Betriebskapazität des Systems ausreichend ist, um die festgelegten Anforderungen zu erfüllen.

Beschreibung: Die Spezifikation der Anforderungen beinhaltet Anforderungen an den Durchsatz und das Antwortverhalten für spezielle Funktionen, gegebenenfalls zusammen mit Beschränkungen für die Verwendung der Systemressourcen. Der vorgeschlagene Systementwurf wird mit den festgelegten Anforderungen verglichen, indem

– ein Modell der Systemprozesse und ihrer Wechselwirkungen erzeugt wird;
– die Verwendung der Ressourcen jedes Prozesses bestimmt wird, zum Beispiel Prozessorzeit, Bandbreite der Kommunikation, Speichermedien usw.;
– die Verteilung der Anforderungen an das System unter durchschnittlichen und schlechtesten Bedingungen bestimmt wird;
– der durchschnittliche und der schlechteste Durchsatz und die durchschnittlichen und die schlechtesten Reaktionszeiten einzelner Systemfunktionen berechnet werden.

Für einfache Systeme kann eine analytische Lösung ausreichend sein, wohingegen für komplexere Systeme eine Simulation geeigneter sein kann, um genaue Ergebnisse zu erhalten.

Vor einer genauen Modellierung kann eine einfache Überprüfung der Ressourcen verwendet werden, die die Anforderungen aller Prozesse an die Ressourcen zusammenfasst. Wenn die Anforderungen die projektierte Systemkapazität übersteigen, ist der Entwurf nicht durchführbar. Auch wenn der Entwurf diesen Test besteht, kann eine Modellierung der Leistungsfähigkeit aufzeigen, dass übermäßige Verzögerungen und Ausführungszeiten infolge fehlender Ressourcen auftreten werden. Um dies zu vermeiden, entwerfen die Ingenieure die Systeme oft derart, dass nur ein Teil (zum Beispiel 50 %) der gesamten Ressourcen verwendet wird.

Literaturhinweis:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.21 Belastungstest (en: avalanche/stress testing)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.6 der IEC 61508-3 angeführt.

Ziel: Außergewöhnlich hohe Belastung des Prüflings, um zu zeigen, dass er normalen Belastungen leicht standhalten würde.

Beschreibung: Es gibt eine Vielzahl von Testbedingungen, die für Belastungstests angewendet werden können, unter anderem:

– wird der Prüfling zyklisch aufgerufen (gepollt), wird er mehr Eingabeänderungen je Zeiteinheit ausgesetzt als unter normalen Bedingungen;
– arbeitet der Prüfling ereignisgesteuert, wird die Anzahl der Ereignisse je Zeiteinheit auf mehr als unter normalen Bedingungen erhöht;
– wenn die Größe einer Datenbasis eine wichtige Rolle spielt, dann wird diese auf mehr als unter normalen Bedingungen vergrößert;
– Geräte, die das System beeinflussen, werden jeweils auf ihre größte oder kleinste Geschwindigkeit eingestellt;
– in Extremfällen werden möglichst alle beeinflussenden Faktoren gleichzeitig auf ihre Grenzwerte eingestellt.

Unter diesen Testbedingungen kann das Zeitverhalten des Prüflings bewertet werden. Der Einfluss von Laständerungen kann beobachtet werden. Die richtige Dimensionierung interner Speicher oder dynamischer Variablen, des Stacks usw. kann überprüft werden.

Literaturhinweis:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.22 Reaktionszeiten und Speicherbeschränkungen

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle B.6 der IEC 61508-3 angeführt.

Ziel: Sicherstellen, dass das System seine zeitlichen Anforderungen und Speicheranforderungen erfüllen wird.

Beschreibung: Die Spezifikation der Anforderungen an das System und an die Software beinhaltet Anforderungen an den Speicher und das Antwortverhalten für spezielle Funktionen, gegebenenfalls zusammen mit Beschränkungen für die Verwendung der Systemressourcen.

Es wird eine Analyse durchgeführt, die die Verteilung der Anforderungen unter durchschnittlichen und schlechtesten Bedingungen ermittelt. Diese Analyse erfordert Abschätzungen der Verwendung der Ressourcen und des Zeitverhaltens jeder Systemfunktion. Diese Abschätzungen können auf mehrere Arten erfolgen, zum Beispiel durch Vergleich mit einem bestehenden System oder einem Prototypen sowie durch Benchmark-Tests bei zeitkritischen Systemen.

C.5.23 Einflussanalyse

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-3 angeführt.

Ziel: Ermittlung der Auswirkungen, die eine Veränderung oder Verbesserung an einem Softwaresystem auf andere Softwaremodule in diesem Softwaresystem oder auch auf andere Systeme haben wird.

Beschreibung: Vor der Ausführung von Modifikationen oder Verbesserungen der Software sollte eine Analyse ihrer Auswirkungen sowie die Ermittlung der betroffenen Softwaresysteme und Softwaremodule ausgeführt werden.

Nach dem Abschluss der Analyse ist eine Entscheidung bezüglich der Reverifikation des Softwaresystems erforderlich. Diese hängt von der Anzahl der betroffenen Softwaremodule, der Kritikalität der betroffenen Softwaremodule und der Art der Änderung ab. Mögliche Entscheidungen sind:

– nur das veränderte Softwaremodul wird neu verifiziert;
– alle betroffenen Softwaremodule werden neu verifiziert oder
– das gesamte System wird neu verifiziert.

Literaturhinweis:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.5.24 Software-Konfigurationsmanagement

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-3 angeführt.

Ziel: Das Software-Konfigurationsmanagement soll die Übereinstimmung von zusammengehörigen Teilen (Produkten) der Entwicklungsergebnisse sicherstellen, wenn diese sich ändern. Das Konfigurationsmanagement wird im Allgemeinen sowohl auf die Hardware- als auch auf die Softwareentwicklung angewendet.

Beschreibung: Das Software-Konfigurationsmanagement ist ein Verfahren, das durchgängig während der Entwicklung (siehe IEC 61508-3, 6.2.3) verwendet wird. Im Wesentlichen erfordert es die Dokumentierung der Produktion jeder Version jedes bedeutsamen Produktes und jeder Beziehung zwischen den unterschiedlichen Versionen der unterschiedlichen Produkte. Die resultierende Dokumentation erlaubt es dem Entwickler, die Auswirkung einer Änderung eines Produktes (besonders eines seiner Elemente) auf andere Produkte zu beurteilen. Insbesondere können Systeme oder Teilsysteme aus einem einheitlichen Satz der Versionen der Elemente zuverlässig wiederhergestellt werden.

Literaturhinweise:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

Software Configuration Management: Coordination for Team Productivity. W.A. Babich. Addison-Wesley, 1986, ISBN 0201101610, 9780201101614

CMMI: guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad, Sandy Shrum, Addison-Wesley, 2003, ISBN 0321154967, 9780321154965

C.5.25 Validierung durch Regressionstest

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.8 der IEC 61508-3 angeführt.

Ziel: Sicherstellung, dass zulässige Ergebnisse aus den Regressionstests gezogen werden.

Beschreibung: Vollständige Regressionstests eines großen oder komplexen Systems werden normalerweise einen großen Aufwand und einige Betriebsmittel erfordern. Wo möglich ist es wünschenswert, die Regressionstests einzuschränken, um nur die Systemaspekte von direktem Interesse an diesem Punkt in der Systementwicklung abzudecken. Bei diesem partiellen Regressionstest ist es entscheidend, ein klares Verständnis von dem Anwendungsbereich des partiellen Tests zu haben und nur zulässige Ergebnisse bezüglich des geprüften Zustands des Systems zu ziehen.

Literaturhinweis:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R.Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986

C.5.26 Animation der Spezifikation und des Entwurfs

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.9 der IEC 61508-3 angeführt.

Ziel: Anleiten der Softwareverifikation mittels einer systematischen Überprüfung der Spezifikation.

Beschreibung: Eine Darstellung der Software, die abstrakter ist als der ausführbare Code (d. h. eine Spezifikation oder ein Entwurf auf hohem Niveau), wird untersucht, um das Verhalten der letztendlich ausführbaren Software zu bestimmen. Die Überprüfung wird irgendwie automatisiert (in Abhängigkeit von den Möglichkeiten, die von der Art und dem Niveau der Abstraktion auf einem höheren Niveau der Darstellung ermöglicht werden), um das Verhalten und die Ausgaben der ausführbaren Software zu simulieren. Eine Anwendung dieses Ansatzes ist die Erzeugung von Tests (oder „Weissagungen“), die später auf die ausführbare Software angewendet werden können, demzufolge der Testprozess zu einem ähnlichen Grad automatisiert wird. Eine weitere Anwendung ist die Animation einer Benutzerschnittstelle, so dass nicht technische Endanwender ein ungefähres Verständnis von der genauen Bedeutung der Spezifikation, mit der die Softwareentwickler arbeiten werden, erlangen können. Dies stellt eine nützliche Methode der Kommunikation zwischen den zwei Gruppen bereit.

Literaturhinweise:

Supporting the Software Testing Process through Specification Animation. T.Miller, P.Strooper. In Proceedings of the First International Conference on Software Engineering and Formal Methods (SEFM'03), ed. P.Lindsay. IEEE Computer Society, IEEE Computer Society, 2003, ISBN 0769519490, 9780769519494

B model animation for external verification. H.Waeselynck, S.Behnia, In Proceedings. of the Second International Conference on Formal Engineering Methods, 1998. IEEE Computer Society, 1998, ISBN 0-8186-9198-0

C.5.27 Modellbasiertes Testen (Testfallgenerierung)

ANMERKUNG Dieses Verfahren/Maßnahme ist in Tabelle A.5 der IEC 61508-3 angeführt.

Ziel: Ermöglichung der effizienten automatischen Testfallgenerierung aus Systemmodellen und Erzeugung von leicht wiederholbaren Testreihen.

Beschreibung: Modellbasiertes Testen (en: model-based testing, MBT) ist ein Ansatz, der dem Black-Box-Verfahren entspricht, in dem allgemeine Testaufgaben wie Testfallgenerierung (en: test case generation, TCG) und Beurteilung der Testergebnisse auf einem Modell des zu testenden Systems oder Anwendung (en: system under test, SUT) basieren. Es werden gewöhnlich, aber nicht nur, die Systemdaten und das Anwenderverhalten unter Verwendung von endlichen Zustandsmaschinen, Markov-Prozessen, Entscheidungstabellen oder Ähnlichem (El-Far, 2001, verallgemeinert) modelliert. Zusätzlich kann modellbasiertes Testen mit der Messung der Testabdeckung auf Ebene des Source-Codes kombiniert werden. Funktionale Modelle können auf dem vorhandenen Source-Code basieren.

Modellbasiertes Testen ist die automatische Generierung von effizienten Testfällen/-verfahren unter Verwendung von Modellen der Anforderungen an das System und der festgelegten Funktionalität (SoftwareTech, 2009).

Da das Testen sehr teuer ist, gibt es eine große Nachfrage nach Werkzeugen zur automatischen Testfallgenerierung. Deshalb ist zurzeit das modellbasierte Testen ein sehr aktives Forschungsgebiet, auf eine Vielzahl von verfügbaren Werkzeugen zur Testfallgenerierung (TCG) hinauslaufend. Diese Werkzeuge entnehmen üblicherweise eine Testreihe aus dem Verhaltensteil des Modells und gewährleisten bestimmten Anforderungen an die Testabdeckung zu entsprechen.

Das Modell ist eine abstrakte, teilweise Darstellung des gewünschten Verhaltens des zu testenden Systems (SUT). Von diesem Modell werden Testmodelle abgeleitet, die eine abstrakte Testreihe bilden. Von dieser abstrakten Testreihe werden Testfälle abgeleitet und gegen das System durchgeführt. Ebenso können Tests gegen das Systemmodell durchgeführt werden. MBT mit TCG basiert auf und ist stark verbunden mit dem Gebrauch von formalen Methoden. So sind die Empfehlungen ähnlich mit Bezug auf die Sicherheits-Integritätslevel (SIL): ++ (besonders empfohlen) für höhere SILs und nicht empfohlen für niedrige SILs.

Die spezifischen Tätigkeiten sind im Allgemeinen:

– Erstellen des Modells (aus den Anforderungen an das System)
– Generierung der erwarteten Eingaben
– Generierung der erwarteten Ausgaben
– Durchführung der Tests
– Vergleich der wirklichen Ausgaben mit den erwarteten Ausgaben
– Entscheidung über weitere Handlungen (Modifikation des Modells, Generierung neuer Tests, Bewertung der Zuverlässigkeit/Qualität der Software)

Tests können mit verschiedenen Methoden und Verfahren abgeleitet werden, um Modelle des Verhaltens des Anwenders/Systems wiederzugeben, z. B.:

– durch Anwendung von Entscheidungstabellen
– durch Anwendung von endlichen Zustandsmaschinen
– durch Anwendung der Grammatik
– durch Anwendung von Modellen von Markovketten
– durch Anwendung von Zustandsdiagrammen
– durch Nachweis der Theoreme
– durch Programmierung einschränkender Logik
– durch Prüfung der Modelle
– durch symbolische Ausführung

– durch Anwendung ein ereignisgesteuerten Modells
– reaktive Systemtests: parallele hierarchische endliche Automaten
– usw.

Modellbasiertes Testen nimmt in letzter Zeit besonders die sicherheitskritischen Gebiete ins Visier. Es erlaubt frühe Aufdeckung von Mehrdeutigkeiten in der Spezifikation und dem Entwurf, stellt die Fähigkeit zur Verfügung, viele nichtwiederholende effiziente Tests automatisch zu erzeugen, Regressionstestreihen zu beurteilen, Softwarezuverlässigkeit und -qualität zu bewerten und Testreihen leicht zu aktualisieren.

Eine gründliche Übersicht wird durch ElFar (2001) und SoftwareTech 2009 zur Verfügung gestellt (siehe Literaturhinweise). Andere Details und bereichsspezifische Probleme werden in den anderen Literaturhinweisen besprochen.

Literaturhinweise:

T. Bauer, F. Böhr, D. Landmann, T. Beletski, R. Eschbach, Robert and J.H. Poore, *From Requirements to Statistical Testing of Embedded Systems* Software Engineering for Automotive Systems - SEAS 2007, ICSE Workshops, Minneapolis, USA

Eckard Bringmann, Andreas Krämer; *Model-based Testing of Automotive Systems* In: ICST, pp.485-493, 2008 International Conference on Software Testing, Verification, and Validation, 2008

Broy M., *Challenges in automotive software engineering*, International conference on Software engineering (ICSE '06),Shanghai, China, 2006

I. K. El-Far and J. A. Whittaker, *Model-Based Software Testing*. Encyclopedia of Software Engineering (edited by J. J. Marciniak). Wiley, 2001

Heimdahl, M.P.E.: *Model-based testing: challenges ahead*, Computer Software and Applications Conference (COMPSAC 2005), 25-28 July 2005, Edinburgh, Scotland, UK, 2005

Jonathan Jacky, Margus Veanes, Colin Campbell, and Wolfram Schulte, *Model-Based Software Testing and Analysis with C#*, ISBN 978-0-521-68761-4, Cambridge University Press 2008

A. Paradkar, *Case Studies on Fault Detection Effectiveness of Model-based Test Generation Techniques*, in ACM SIGSOFT SW Engineering Notes, Proc. of the first int. workshop on Advances in model-based testing A-MOST '05, Vol. 30 Issue 4. ACM Press 2005

S. J. Prowell, *Using Markov Chain Usage Models to Test Complex Systems*, HICSS '05: 38th Annual Hawaii, International Conference on System Sciences, 2005

Mark Utting and Bruno Legeard, *Practical Model-Based Testing: A Tools Approach*, ISBN 978-0-12-372501-1, Morgan-Kaufmann 2007

Hong Zhu et al. (2008). AST '08: *Proceedings of the 3rd International Workshop on Automation of Software Test*. ACM Press. ISBN 978-1-60558-030-2

Model-Based Testing of Reactive Systems Advanced Lecture Series, LNCS 3472, Springer-Verlag, 2005, ISBN 978-3-540-26278-7

Model-based Testing, SoftwareTech July 2009, Vol. 12, No. 2, Software Testing: A Life Cycle Perspective, http://www.goldpractices.com/practices/mbt/

C.6 Beurteilung der funktionalen Sicherheit

ANMERKUNG Relevante Verfahren und Maßnahmen können auch in B.6 gefunden werden.

C.6.1 Entscheidungs-/Wahrheitstabellen

ANMERKUNG Dieses Verfahren/Maßnahme ist in den Tabellen A.10 und B.7 der IEC 61508-3 angeführt.

Ziel: Bereitstellung einer eindeutigen und lückenlosen Spezifikation und Analyse komplexer logischer Kombinationen und Beziehungen.

Beschreibung: Diese Methode verwendet zweidimensionale Tabellen zur exakten Beschreibung logischer Beziehungen zwischen booleschen Programmvariablen.

Durch die Exaktheit und tabellarische Form eignet sich diese Methode als Mittel zur Analyse komplexer logischer Kombinationen, die in Codes ausgedrückt sind.

Wird diese Methode zur Spezifikation verwendet, ist sie potentiell ausführbar.

C.6.2 Software-Gefährdungs- und Betreibbarkeitsuntersuchung (en: software hazard and operability study, CHAZOP, FMEA)

Ziel: Bestimmung der Gefährdungen in einem vorgesehenen oder bestehenden System, deren möglichen Ursachen und Auswirkungen sowie Empfehlungen für Maßnahmen, die die Möglichkeit ihres Auftretens verringern.

Beschreibung: Ein Team aus Ingenieuren mit Fachkenntnis über das gesamte zu betrachtende System führt in einer Reihe von Sitzungen eine strukturierte Prüfung des Entwurfs durch. Sie betrachten sowohl die funktionalen Aspekte des Entwurfs und auch, wie das System in der Praxis arbeiten würde (einschließlich menschlicher Tätigkeiten und Instandhaltung). Ein Teamleiter ermutigt die Mitglieder des Teams zur Kreativität bei der Ermittlung von möglichen Gefährdungen und führt das Verfahren, indem er jedes Teilsystem in Verbindung mit mehreren Leitwörtern vorstellt wie: „keines“, „mehr von“, „weniger von“, „Teil von“, „mehr als“ (oder „ebenso wie“) und „anders als“. Jede angesprochene Bedingung und Ausfallart wird auf ihre Möglichkeit beurteilt, wie sie geschehen kann, ihre möglichen Auswirkungen (entsteht eine Gefährdung?), wie sie vermieden werden könnte und ob das Verfahren zur Vermeidung den Aufwand wert ist.

Gefährdungsuntersuchungen können an vielen Stufen der Projektentwicklung stattfinden, sind jedoch am wirksamsten, wenn sie früh genug ausgeführt werden, um die Projektierung und die Entscheidungen zur Betriebsart der Anlage beeinflussen zu können.

Das HAZOP-Verfahren wurde in der Prozessindustrie entwickelt und erfordert Anpassungen für Anwendungen mit Software. Verschiedene abgeleitete Verfahren (oder Rechner-HAZOP (en: computer HAZOPs – „CHAZOPs“)) wurden vorgeschlagen. Diese bringen im Allgemeinen neue Leitwörter ein und/oder schlagen Ansätze zur systematischen Abdeckung des Systems und der Softwarearchitektur vor.

Literaturhinweise:

OF-FMEA: an approach to safety analysis of object-oriented software intensive systems, T. Cichocki, J. Gorski. In Artificial Intelligence and Security in Computing Systems: 9th International Conference, ACS '2002. Ed. J. Soldek. Springer, 2003, ISBN 1402073968, 9781402073960

Software FMEA techniques. P.L.Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

C.6.3 Analyse von Ausfällen infolge gemeinsamer Ursache

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle A.10 der IEC 61508-3 angeführt.

ANMERKUNG 2 Siehe auch Anhang D der IEC 61508-6.

Ziel: Bestimmung möglicher Ausfälle in redundanten Systemen oder redundanten Teilsystemen, die die Vorteile der Redundanz durch das gleichzeitige Auftreten von gleichen Ausfällen in den redundanten Teilen aufheben würden.

Beschreibung: Systeme, die mit der Sicherheit einer Anlage betraut sind, verwenden oft Hardwareredundanz und Mehrheitsentscheider oder Vergleicher. Dies wird eingesetzt, um zufällige Hardwareausfälle in Bauteilen oder Teilsystemen zu vermeiden, die sonst die korrekte Datenverarbeitung verhindern würden.

Einige Ausfälle können jedoch mehr als ein Bauteil oder Teilsystem gleichartig betreffen. Wenn zum Beispiel ein System in einem einzigen Raum installiert ist, können Ausfälle der Klimaanlage die Vorteile der Redundanz zunichte machen. Das Gleiche gilt für andere äußere Einflüsse auf das System wie z. B. Feuer, Hochwasser, elektromagnetische Beeinflussung, Flugzeugabstürze und Erdbeben. Das System kann auch von Zwischenfällen während des Betriebs und der Instandhaltung beeinflusst werden. Deshalb ist es notwendig, dass angemessene und gut dokumentierte Verfahren für Betrieb und Instandhaltung bereitgestellt werden und dass das Betriebs- und Wartungspersonal gründlich ausgebildet wird.

Interne Ursachen können ebenfalls erheblich zu Ausfällen infolge gemeinsamer Ursache beitragen. Sie können von Entwurfsfehlern in gemeinsamen oder identischen Bauteilen und ihren Schnittstellen wie auch aus der Alterung von Bauteilen herrühren. Die Analyse für Ausfälle infolge gemeinsamer Ursache muss das System nach solchen möglichen gemeinsamen Ausfällen durchsuchen. Methoden der Analyse von Ausfällen infolge gemeinsamer Ursache sind: allgemeine Qualitätskontrolle, Überprüfung des Entwurfs, Verifikation und Test durch eine unabhängige Arbeitsgruppe sowie die Analyse tatsächlicher Zwischenfälle unter Berücksichtigung von Erfahrungen mit ähnlichen Systemen. Der zu analysierende Bereich geht jedoch über den der Hardware hinaus. Sogar wenn Softwarediversität in verschiedenen Kanälen eines redundanten Systems verwendet wird, könnten Gemeinsamkeiten in den Softwareansätzen enthalten sein, die zu Ausfällen infolge gemeinsamer Ursache führen. Zum Beispiel Fehler in der gemeinsamen Spezifikation.

Wenn Ausfälle infolge gemeinsamer Ursache nicht genau gleichzeitig auftreten, können Vorkehrungen mit Hilfe von Vergleichsmethoden zwischen den redundanten Kanälen getroffen werden, die zur Erkennung eines Ausfalls führen, bevor dieser in allen Kanälen auftritt. Die Analyse von Ausfällen infolge gemeinsamer Ursache sollte dieses Verfahren berücksichtigen.

Literaturhinweis:

Reliability analysis of hierarchical computer-based systems subject to common-cause failures. L.Xing, L.Meshkat, S.Donohue. Reliability Engineering & System Safety Volume 92, Issue 3, March 2007

C.6.4 Zuverlässigkeitsblockdiagramm

ANMERKUNG 1 Dieses Verfahren/Maßnahme ist in Tabelle A.10 der IEC 61508-3 angeführt und wird im Anhang B der IEC 61508-6 verwendet.

ANMERKUNG 2 Siehe auch B.6.6.7 „Zuverlässigkeitsblockdiagramm“.

Ziel: Graphische Darstellung einer Menge von Ereignissen, die stattfinden müssen, und Bedingungen, die erfüllt sein müssen, damit das System oder eine Anwendung erfolgreich funktioniert.

Beschreibung: Das Ziel der Analyse ist es, einen Erfolgspfad, bestehend aus Blöcken, Linien und logischen Verknüpfungen, darzustellen. Ein Erfolgspfad beginnt an einer Seite des Diagramms und setzt sich über die Blöcke und Verzweigungen bis zur anderen Seite fort. Ein Block stellt eine Bedingung oder ein Ereignis dar, und der Pfad kann ihn durchlaufen, wenn die Bedingung wahr ist oder das Ereignis stattgefunden hat. Kommt der Pfad zu einer Verzweigung, wird er fortgesetzt, wenn die Logik an der Verzweigung erfüllt ist. Wenn er einen Scheitelpunkt erreicht, kann eine Fortsetzung entlang aller abgehenden Linien erfolgen. Wenn mindestens ein erfolgreicher Pfad durch das Diagramm existiert, war die Analyse erfolgreich.

Literaturhinweise:

IEC 61025:2006, *Fault tree analysis (FTA)*

From safety analysis to software requirements. K.M. Hansen, A.P. Ravn, A.P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998

IEC 61078:2006, *Analysis techniques for dependability – Reliability block diagram and boolean methods*

Anhang D
(informativ)

Ein probabilistischer Ansatz zur Bestimmung der Sicherheitsintegrität von vorentwickelter Software

D.1 Allgemeines

Dieser Anhang stellt eine erste erfahrungsbasierte Leitlinie für einen probabilistischen Ansatz zur Bestimmung der Sicherheitsintegrität von vorentwickelter Software dar. Er wird insbesondere als geeigneter Beitrag zur Qualifizierung von Betriebssystemen, Modulen einer Bibliothek, Compilern und anderer Systemsoftware angesehen. Er stellt dar, was möglich ist, jedoch sollte das Verfahren nur von in der statistischen Analyse sachkundigen Personen angewendet werden.

ANMERKUNG Dieser Anhang verwendet den Begriff „Vertrauensniveau“, der in IEEE 352 beschrieben ist. Ein gleichwertiger Begriff „Signifikanzniveau“ wird in IEC 61164 verwendet.

Das Verfahren kann auch dazu verwendet werden, ein zeitliches Wachstum des Sicherheits-Integritätslevels der Software darzulegen. Zum Beispiel kann für eine Software, die nach den Anforderungen des SIL1 der IEC 61508-3 entworfen wurde, nach einer angemessenen Dauer und einer angemessen großen Anzahl von Anwendungen mit erfolgreichem Betrieb gezeigt werden, dass diese auch für SIL2 geeignet ist.

Die Tabelle D.1 zeigt die Anzahl der erfolgreich ausgeführten Anforderungen (d. h. kein Versagen) oder Stunden erfolgreichen Betriebs, die für die Qualifikation für ein bestimmtes Sicherheits-Integritätslevel notwendig sind. Sie ist eine Zusammenfassung der Ergebnisse aus D.2.1 und D.2.3.

Betriebserfahrung kann entsprechend Tabelle D.2 mathematisch behandelt werden, um statistische Tests zu ergänzen oder zu ersetzen. An mehreren Betriebsstätten gewonnene Betriebserfahrungen können nur dann zusammengeführt werden (d. h. durch Hinzufügen der Anzahl der bearbeiteten Anforderungen oder Betriebsstunden), wenn

– die in dem sicherheitsbezogenen E/E/PE-System zu verwendende Softwareversion identisch mit derjenigen ist, mit der die Betriebserfahrung gewonnen wurde;
– vergleichbare Wertebereiche für die Eingänge verwendet wurden;
– es ein wirksames System zur Meldung und Dokumentation von Versagensfällen gibt und
– die zutreffenden Voraussetzungen (siehe D.2) erfüllt sind.

Tabelle D.1 – Notwendige Vorgeschichte zur Zuordnung von Sicherheits-Integritätsleveln bei gegebenem Vertrauensniveau

SIL	Betriebsart mit niedriger Anforderungsrate	Anzahl der bearbeiteten Anforderungen		Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung	Gesamtbetriebsstunden	
	(Wahrscheinlichkeit eines Versagens der entworfenen Funktion bei Anforderung)	$1-\alpha = 0{,}99$	$1-\alpha = 0{,}95$	(Wahrscheinlichkeit eines gefahrbringenden Versagens je Stunde)	$1-\alpha = 0{,}99$	$1-\alpha = 0{,}95$
4	$\geq 10^{-5}$ bis $< 10^{-4}$	$4{,}6 \times 10^5$	3×10^5	$\geq 10^{-9}$ bis $< 10^{-8}$	$4{,}6 \times 10^9$	3×10^9
3	$\geq 10^{-4}$ bis $< 10^{-3}$	$4{,}6 \times 10^4$	3×10^4	$\geq 10^{-8}$ bis $< 10^{-7}$	$4{,}6 \times 10^8$	3×10^8
2	$\geq 10^{-3}$ bis $< 10^{-2}$	$4{,}6 \times 10^3$	3×10^3	$\geq 10^{-7}$ bis $< 10^{-6}$	$4{,}6 \times 10^7$	3×10^7
1	$\geq 10^{-2}$ bis $< 10^{-1}$	$4{,}6 \times 10^2$	3×10^2	$\geq 10^{-6}$ bis $< 10^{-5}$	$4{,}6 \times 10^6$	3×10^6
ANMERKUNG 1	$1-\alpha$ stellt das Vertrauensniveau dar.					
ANMERKUNG 2	Siehe D.2.1 und D.2.3 bezüglich Voraussetzungen und Einzelheiten, wie diese Tabelle hergeleitet wird.					

D.2 Gleichungen für statistische Tests und Beispiele für ihre Anwendung

D.2.1 Einfacher statistischer Test für die Betriebsart mit niedriger Anforderungsrate

D.2.1.1 Voraussetzungen

a) Die Verteilung der Testdaten ist gleich zur Verteilung der Anforderungen während des Online-Betriebs.

b) Die Testdurchläufe sind im Hinblick auf die Ursache eines Versagens statistisch voneinander unabhängig.

c) Es existiert ein angemessener Mechanismus zur Erkennung jedes möglichen Versagens.

d) Für die Anzahl der Testfälle gilt $n > 100$.

e) Es tritt kein Versagen während der n Testfälle ein.

D.2.1.2 Ergebnisse

Die Versagenswahrscheinlichkeit p (je Anforderung) ist bei einem Vertrauensniveau von $1 - \alpha$ gegeben durch

$$p \leq 1 - \sqrt[n]{\alpha} \quad \text{oder} \quad n \geq -\frac{\ln \alpha}{p}$$

D.2.1.3 Beispiel

Tabelle D.2 – Wahrscheinlichkeiten eines Versagens für die Betriebsart mit niedriger Anforderungsrate

$1 - \alpha$	P
0,95	3/n
0,99	4,6/n

Für die Wahrscheinlichkeit eines Versagens bei Anforderung entsprechend SIL 3 und das Vertrauensniveau von 95 % ergibt die Anwendung der Formel 30 000 Testfälle unter den angegebenen Voraussetzungen. Tabelle D.1 fasst die Ergebnisse für jeden Sicherheits-Integritätslevel zusammen.

D.2.2 Test des Wertebereichs der Eingänge für die Betriebsart mit niedriger Anforderungsrate

D.2.2.1 Voraussetzungen

Die einzige Voraussetzung ist, dass die Testdaten so ausgewählt werden, dass eine einheitliche Zufallsverteilung über den Wertebereich der Eingänge erreicht wird.

D.2.2.2 Ergebnisse

Das Ziel ist es, diejenige Anzahl n von Tests zu finden, die auf Basis der Genauigkeitsschwelle δ der Eingänge für eine zu testende Funktion mit niedriger Anforderung (wie eine Sicherheitsabschaltung) notwendig ist.

Tabelle D.3 – Mittlerer Abstand von zwei Testpunkten

Dimension des Gültigkeitsbereichs	Mittlerer Abstand von zwei Testpunkten in Richtung einer beliebig gewählten Achse
1	$\delta = 1/n$
2	$\delta = \sqrt[2]{1/n}$
3	$\delta = \sqrt[3]{1/n}$
k	$\delta = \sqrt[k]{1/n}$
ANMERKUNG *k* kann eine beliebige ganze Zahl sein. Die Werte 1, 2 und 3 sind nur Beispiele.	

D.2.2.3 Beispiel

Betrachtet sei eine Sicherheitsabschaltung, die nur von den beiden Variablen A und B abhängt. Wenn verifiziert wurde, dass die Schwellwerte, mit der die Variablen A und B in einem Paar von Eingangswerten noch richtig unterschieden werden, mit 1 % des Messbereichs von A oder B richtig wiedergegeben sind, dann beträgt die notwendige Anzahl der einheitlich verteilten Testfälle im Wertebereich von A und B

$$n = 1/\delta^2 = 10^4$$

D.2.3 Einfacher statistischer Test für die Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung

D.2.3.1 Voraussetzungen

a) Die Verteilung der Testdaten ist gleich zur Verteilung während des Online-Betriebs.

b) Die relative Reduzierung der Wahrscheinlichkeit, dass kein Versagen eintritt, ist proportional zu der Länge des betrachteten Zeitintervalls und sonst konstant.

c) Es existiert ein angemessener Mechanismus zur Erkennung jedes möglichen Versagens.

d) Der Test erstreckt sich über eine Testdauer *t*.

e) Während *t* tritt kein Versagen auf.

D.2.3.2 Ergebnisse

Zwischen der Wahrscheinlichkeit eines Versagens λ, dem Vertrauensniveau 1-α und der Testdauer *t* gilt die Beziehung

$$\lambda = -\frac{\ln\alpha}{t}$$

Die Wahrscheinlichkeit eines Versagens ist indirekt proportional zur mittleren Betriebsdauer zwischen Versagensfällen (Ausfällen), der MTBF:

$$\lambda = \frac{1}{\mathrm{MTBF}}$$

ANMERKUNG Diese Norm unterscheidet nicht zwischen der Wahrscheinlichkeit eines Versagens je Stunde und der Versagensrate in einer Stunde. Streng genommen gilt zwischen der Wahrscheinlichkeit eines Versagens *F* und der Versagensrate *f* die Gleichung $F = 1 - e^{-ft}$, aber der Anwendungsbereich dieser Norm gilt nur für Versagensraten (Ausfallraten) kleiner als 10^{-5} und für so kleine Werte gilt $F \approx ft$.

D.2.3.3 Beispiel

Tabelle D.4 – Wahrscheinlichkeit eines Versagens für die Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung

1 - α	λ
0,95	3/*t*
0,99	4,6/*t*

Zur Verifikation, dass die MTBF mindestens 10^8 h bei einem Vertrauensniveau von 95 % beträgt, ist eine Testdauer von 3×10^8 h erforderlich, und die Voraussetzungen müssen erfüllt sein. Tabelle D.1 fasst die erforderliche Anzahl der Tests für jeden Sicherheits-Integritätslevel zusammen.

D.2.4 Vollständiger Test

Das Programm wird als eine Urne betrachtet, die eine bekannte Anzahl *N* von Kugeln enthält. Jede Kugel stellt eine interessierende Programmeigenschaft dar. Es werden zufällig Kugeln gezogen und nach Inspektion zurückgelegt. Ein vollständiger Test ist erreicht, wenn alle Bälle gezogen worden sind.

D.2.4.1 Voraussetzungen

a) Die Verteilung der Testdaten ist derart, dass jede der *N* Programmeigenschaften mit gleicher Wahrscheinlichkeit getestet wird.

b) Die Testdurchläufe sind voneinander unabhängig.

c) Jedes auftretende Versagen wird erkannt.

d) Für die Anzahl der Testfälle gilt $n >> N$.

e) Es tritt kein Versagen während der *n* Testfälle ein.

f) Jeder Testdurchlauf testet eine Programmeigenschaft (eine Programmeigenschaft ist das, was während eines Durchlaufs getestet werden kann).

D.2.4.2 Ergebnisse

Die Wahrscheinlichkeit *p* zum Test aller Programmeigenschaften wird bestimmt durch

$$p = \sum_{j=0}^{N-1} (-1)^j \binom{N}{j} \left(\frac{N-j}{N}\right)^n \quad \text{oder} \quad p = 1 + \sum_{j=1}^{N} (-1)^j C_{j,N} \left(\frac{N-j}{N}\right)^n$$

mit

$$C_{j,N} = \frac{N(N-1)...(N-j+1)}{j!}$$

Zur Auswertung dieser Gleichung ist gewöhnlich nur der erste Term wichtig, weil realistische Fälle durch $n >> N$ gekennzeichnet sind. Der letzte Faktor macht alle Terme für große *j* sehr klein. Dies wird auch in Tabelle D.5 ersichtlich.

D.2.4.3 Beispiel

Betrachtet sei ein Programm, das in mehreren Anlagen seit mehreren Jahren verwendet wurde. Insgesamt wurden mindestens $7{,}5 \times 10^6$ Durchläufe ausgeführt. Es wird geschätzt, dass jeder hundertste Durchlauf die obigen Voraussetzungen erfüllt. Daher können zur statistischen Auswertung $7{,}5 \times 10^4$ erfolgte Durchläufe herangezogen werden. Es wird weiter geschätzt, dass 4 000 Testdurchläufe einen erschöpfenden Test darstellen. Diese Schätzungen sind konservativ. Gemäß Tabelle D.5 entspricht die Wahrscheinlichkeit, nicht alles getestet zu haben, $2{,}87 \times 10^{-5}$.

Für $N = 4\,000$ betragen die Werte der ersten Terme in Abhängigkeit von n:

Tabelle D.5 – Wahrscheinlichkeit des Tests aller Programmeigenschaften

n	P
5×10^4	$1 - 1{,}49 \times 10^{-2} + 1{,}10 \times 10^{-4} - \ldots$
$7{,}5 \times 10^4$	$1 - 2{,}87 \times 10^{-5} + 4 \times 10^{-10} - \ldots$
1×10^5	$1 - 5{,}54 \times 10^{-8} + 1{,}52 \times 10^{-15} - \ldots$
2×10^5	$1 - 7{,}67 \times 10^{-19} + 2{,}9 \times 10^{-37} - \ldots$

In der Praxis sollte konservativ geschätzt werden.

D.3 Literaturhinweise

Weitere Informationen zu dem obigen Verfahren können in folgenden Dokumenten gefunden werden:

IEC 61164:2004, *Reliability growth – Statistical test and estimation methods*

Verification and Validation of Real-Time Software, Chapter 5. W. J. Quirk (ed.). Springer Verlag, 1985, ISBN 3-540-15102-8

Combining Probabilistic and Deterministic Verification Efforts. W. D. Ehrenberger, SAFECOMP 92, Pergamon Press, ISBN 0-08-041893-7

Ingenieurstatistik. Heinhold/Gaede, Oldenburg, 1972, ISBN 3-486-31743-1

IEEE 352:1987, *IEEE Guide for general principles of reliability analysis of nuclear power generating station safety systems*

Anhang E
(informativ)

Überblick über Verfahren und Maßnahmen für den Entwurf von ASICs

ANMERKUNG Der in diesem Anhang enthaltene und von IEC 61508-2 angeführte Überblick über Verfahren und Maßnahmen sollte weder als vollständig noch als ausschließlich betrachtet werden.

E.1 Entwurfsbeschreibung in (V)HDL

Ziel: Funktionale Beschreibung mit hohem Niveau in einer Hardwarebeschreibungssprache, zum Beispiel VHDL oder Verilog.

Beschreibung: Funktionale Beschreibung mit hohem Abstraktionsgrad in einer Hardwarebeschreibungssprache, zum Beispiel VHDL oder Verilog. Die angewendete Hardwarebeschreibungssprache sollte eine funktionale und/oder anwendungsorientierte Beschreibung zulassen und sollte von Details der späteren Ausführung abstrahiert werden. Datenflüsse, Verzweigungen, arithmetische und/oder logische Operationen sollten durch die Anweisungen und Operatoren der Hardwarebeschreibungssprache ohne manuelle Umwandlung in logische Gatter der verwendeten Bibliothek durchgeführt werden.

ANMERKUNG Zur Vereinfachung wird „funktionale Beschreibung mit hohem Abstraktionsgrad in einer Hardwarebeschreibungssprache“ im Rest des Dokumentes als (V)HDL bezeichnet.

Literaturhinweis:

IEEE VHDL, *Verilog + Standard VHDL Design guide*

E.2 Schaltplaneingabe

Ziel: Funktionale Beschreibung der Schaltkreise durch Erstellen eines Schaltplans unter Verwendung der Gatter und/oder Makros der Hersteller-Bibliothek.

Beschreibung: Beschreibung der Funktionalität der Schaltkreise durch schematische Eingabe des Schaltplans. Die zu realisierende Funktion sollte mittels Instanzieren (Import) der elementaren logischen Schaltelemente wie zum Beispiel UND, ODER, NICHT zusammen mit den Makros, die aus komplexen arithmetischen und logischen Funktionen bestehen, ausgeführt werden, die dann miteinander verbunden werden. Komplexe Schaltkreise sollten unter Berücksichtigung funktionaler Gesichtspunkte aufgeteilt werden und können auf verschiedenen Zeichnungen verteilt werden, die hierarchisch miteinander verbunden werden. Die Verbindungssignale sollten einmalig definiert werden und sollten eindeutige Signalbezeichnungen über die gesamte Hierarchie haben. Die Verwendung von globalen Signalen (Labels) sollte so weit wie möglich vermieden werden.

E.3 Strukturierte Beschreibungsmethodik

ANMERKUNG Siehe auch C.2.7 „Strukturierte Programmierung“ und E.12 „Modularisierung“.

Ziel: Die Beschreibung der Funktionalität des Schaltkreises sollte derart strukturiert werden, dass sie leicht lesbar ist, d. h. die Funktion des Schaltkreises kann auf der Basis der Beschreibung ohne Simulationsaufwand intuitiv verstanden werden.

Beschreibung: Beschreibung der Funktionalität des Schaltkreises mit (V)HDL oder durch Schaltplaneingabe. Eine leicht erkennbare und modulare Struktur wird empfohlen. Ebenfalls sollte jedes Modul auf dieselbe Art ausgeführt werden und derart beschrieben werden, dass es leicht lesbar ist mit klar definierten Teilfunktionen. Es wird eine strikte Unterscheidung zwischen der ausgeführten Funktion und den Verbindungen empfohlen, d. h. das Modul, das durch Instanzierung andere Teilmodule ausgeführt wird, enthält eindeutige Verbindungen zu den instanzierten Modulen und sollte keine Schaltlogik enthalten.

E.4 Betriebsbewährte Werkzeuge

Ziel: Anwendung von betriebsbewährten Werkzeugen, um systematische Ausfälle durch eine ausreichend lang erprobte Praxis der Werkzeuge in verschiedenen Projekten zu vermeiden.

Beschreibung: Die meisten der zum Entwurf von ASICs und FPGAs verwendeten Werkzeuge enthalten hoch entwickelte Software, die man nicht als fehlerfrei in Bezug auf ihre ordnungsgemäße Funktionalität ansehen kann und es auch ziemlich wahrscheinlich ist, dass sich wegen der fehlerhaften Funktion Fehler ereignen könnten. Deshalb sollten zum Entwurf von ASICs und FPGAs nur betriebsbewährte Werkzeuge bevorzugt werden. Dies schließt ein:

– Anwendung von Werkzeugen, die (mit vergleichbarer Softwareversion) schon seit langer Zeit oder von einer hohen Anzahl von Benutzern in verschiedenen Projekten mit gleichwertiger Komplexität verwendet werden;

– seit langer Zeit angemessene Betriebserfahrung jedes ASIC/FPGA-Entwicklers mit dem Werkzeug;

– Verwendung von üblich eingesetzten Werkzeugen (mit angemessener Zahl von Benutzern), so dass Informationen bezüglich bekannter Ausfälle mit den Überarbeitungen (Versionskontrolle mit „Bug-Liste“) verfügbar werden. Diese Informationen sollten sogleich in den Entwurfsfluss integriert werden und helfen, systematische Ausfälle zu vermeiden;

– die Konsistenzprüfung der internen Datenbasis des Werkzeugs und die Plausibilitätsprüfung vermeiden fehlerhafte Ergebnisse. Standardwerkzeuge überprüfen die Konsistenz der internen Datenbasis, zum Beispiel, um mit eindeutigen Daten zu arbeiten, die Konsistenz der Datenbasis zwischen dem Synthese- und Place-and-Route-Werkzeug.

ANMERKUNG Die Konsistenzprüfung ist eine integrierte Eigenschaft des verwendeten Werkzeugs und der Entwickler hat darauf begrenzten Einfluss. Wenn die Möglichkeit der manuellen Konsistenzprüfung zur Verfügung gestellt wird, sollte deshalb der Entwickler diese entsprechend verwenden.

E.5 (V)HDL-Simulation

ANMERKUNG Siehe auch E.6 „Funktionstest auf Modulebene“.

Ziel: Funktionale Verifikation des Schaltkreises dargestellt in (V)HDL mittels Simulation.

Beschreibung: Verifikation der Funktion durch Simulation des vollständigen Schaltkreises oder jedes Teilmoduls. Der (V)HDL-Simulator erfasst eine Folge von Ausgaben, die durch die internen Änderungen der Zustände des Schaltkreises als das Ergebnis der angewendeten Eingangsstimuli verursacht werden. Die Verifikation der erfassten Folge von Ausgaben kann entweder durch die vorbereitete Folge von Ausgangssignalen („Zeitdiagramm“) oder durch eine spezielle Umgebung, bekannt als „Testbench“, die während des Entwurfsprozesses eingerichtet wird, ausgeführt werden. Der eingesetzte Simulator sollte die Eigenschaft „betriebsbewährt“ haben, um ordnungsgemäße Ergebnisse zu erhalten und um fehlerhaftes Zeitverhalten der Signale auszublenden (Spikes, Tri-State-Ablaufverfolgung), das durch den Simulator selbst oder fehlerhaftes Modellieren verursacht werden könnte.

E.6 Funktionstest auf Modulebene

ANMERKUNG Siehe auch E.5 „(V)HDL-Simulation“ und E.13 „Testabdeckung der Verifikationsszenarien“.

Ziel: Funktionale Verifikation „von unten nach oben“.

Beschreibung: Verifikation der realisierten Funktion auf Modul-Ebene, zum Beispiel durch die Simulation. Das zu testende Modul wird in einer typischen virtuellen Testumgebung, bekannt als „Testbench“, instanziert und durch das im Code enthaltene Testmuster stimuliert. Es ist mindestens eine ausreichend hohe Abdeckung der festgelegten Funktion einschließlich aller Sonderfälle, sofern sie bestehen, erforderlich. Die automatische Verifikation der Folge von Ausgaben durch den Code der „Testbench“ sollte gegenüber der manuellen Inspektion der Ausgangssignale bevorzugt werden.

E.7 Funktionstest auf oberster Ebene

ANMERKUNG Siehe auch E.8 „Funktionstest eingebettet in Systemumgebung“.

Ziel: Verifikation des ASICs (gesamter Schaltkreis).

Beschreibung: Das Ziel des Tests ist die Verifikation des gesamten ASIC-Schaltkreises.

E.8 Funktionstest eingebettet in Systemumgebung

ANMERKUNG Siehe auch E.7 „Funktionstest auf oberster Ebene“.

Ziel: Verifikation der festgelegten Funktion eingebettet in der Systemumgebung.

Beschreibung: Dieser Test wird die gesamte Funktionalität des ASIC-Schaltkreises in seiner Systemumgebung verifizieren, zum Beispiel mit allen anderen Komponenten, die auf den Leiterplatten oder anderswo angeordnet sind. Das Modellieren aller relevanten Komponenten auf der Leiterplatte und die Simulation des ASICs zusammen mit dem zur Verifikation der ordnungsgemäßen Funktionalität einschließlich des zeitlichen Verhaltens erstellten Modell werden empfohlen. Eine vollständige Funktionsprüfung schließt auch die Prüfung der Module ein, die nur im Falle eines Ausfalls aktiviert werden.

E.9 Eingeschränkte Verwendung asynchroner Konstrukte

Ziel: Vermeidung von typischen Timing-Problemen während der Synthese. Vermeidung von Zweideutigkeiten während der Simulation und Synthese, verursacht durch nicht ausreichendes Modellieren oder mangelhaft testbaren Entwurf.

Beschreibung: Asynchrone Konstrukte wie zum Beispiel SET- und RESET-Signale, hergeleitet über eine kombinatorische Logik, sind während der Synthese anfällig und erzeugen Schaltkreise mit Spikes oder inverser zeitlicher Abfolge und sollte deshalb vermieden werden. Ebenso kann nicht ausreichendes Modellieren nicht richtig vom Synthese-Werkzeug interpretiert werden, so dass zweideutige Ergebnisse während der Simulation verursacht werden. Zusätzliche asynchrone Konstrukte sind schlecht oder überhaupt nicht prüfbar, so dass die Testabdeckung der Ergebnisse und des Online-Tests effektiv reduziert wird. Deshalb wird die Ausführung eines vollständig synchronen Entwurfs mit begrenzter Anzahl von Taktsignalen empfohlen. In Systemen mit mehrphasigen Takten sollte jeder Takt von einem zentralen Takt abgeleitet werden. Der Takteingang einer sequentiellen Logik sollte immer exklusiv mit dem Takt-Signal gespeist werden, das keine kombinatorische Logik enthält. Asynchrone SET- und RESET-Eingänge von sequentiellen Zellen sollten mit synchronen Signalen gespeist werden, die keine kombinatorische Logik enthalten. Master SET- und RESET-Signale sollten mit zwei Flip-Flops synchronisiert werden.

E.10 Synchronisation von primären Eingängen und Kontrolle von Metastabilitäten

Ziel: Vermeidung eines mehrdeutigen Schaltkreis-Verhaltens infolge von Nichteinhaltung der Konfiguration und der Haltezeit.

Beschreibung: Eingangssignale von der externen Peripherie sind im Allgemeinen asynchron und können ihren Zustand willkürlich ändern. Eine direkte Verarbeitung solcher Signale durch die synchronen sequentiellen Schaltelemente der ASICs/FPGAs, zum Beispiel Flip-Flops, führen zur Nichteinhaltung der Konfiguration und der Haltezeit, die unvorhersehbares zeitliches und funktionales Verhalten der ASICs/FPGAs ergeben. Schließlich könnten Metastabilitäten der Speicherelemente entstehen. Jedes asynchrone Eingangssignal sollte deshalb in Bezug auf den synchronen ASIC-Schaltkreis synchronisiert werden, um funktionale Mehrdeutigkeiten zu vermeiden. Folgende Maßnahmen werden empfohlen:

– Eingangssignale sollten mit zwei aufeinander folgenden Speicherelementen (Flip-Flops) oder einem gleichwertigen Schaltkreis synchronisiert werden, um ein vorhersagbares funktionales Verhalten zu erreichen;
– jedes asynchrone Eingangssignal sollte ebenfalls in der oben definierten Methode synchronisiert werden, d. h. jedes asynchrone Signal wird mit genau einem dieser synchronisierten Schaltkreise verbunden.

Falls erforderlich kann der Ausgang des synchronisierten Schaltkreises für mehrfachen Zugriff verwendet werden;

– der synchronisierte Schaltkreis sollte für Prüfungen der Störfestigkeit von parallelen Bussignalen und zur Kontrolle der Datenkonsistenz von benachbarten Messpunkten verwendet werden.

E.11 Entwurf für Testbarkeit

ANMERKUNG 1 Siehe auch E.31 „Implementierung von Teststrukturen“.

Ziel: Vermeidung von nicht prüfbaren oder schlecht prüfbaren Strukturen, um eine hohe Testabdeckung des Fertigungstests oder des Online-Tests zu erreichen.

Beschreibung: Der Entwurf für Testbarkeit wird bestimmt durch die Vermeidung von

– asynchronen Konstrukten,

– Latches und On-Chip-Tri-State-Signalen,

– verdrahteter UND-/ verdrahteter ODER-Logik und redundanter Logik.

Die kombinatorische Tiefe der untergeordneten Schaltkreise spielt eine wichtige Rolle während der Prüfung. Das für eine vollständige Prüfung erforderliche Testmuster steigt exponentiell mit der kombinatorischen Tiefe des Schaltkreises. Deshalb sind Schaltkreise mit hoher kombinatorischer Tiefe mit entsprechenden Mitteln nur schlecht prüfbar.

Ein an der Testbarkeit des Entwurfs orientierter Ansatz stellt sicher, dass die gewünschte Testabdeckung erreicht wird. Weil die tatsächliche Testabdeckung erst in einer sehr späten Phase des Entwurfprozesses bestimmt werden kann, könnte die nicht ausreichende Berücksichtigung der Testbarkeit des Entwurfs die erreichbare Testabdeckung drastisch reduzieren und zu zusätzlichem Aufwand führen.

ANMERKUNG 2 Die Testabdeckung wird normalerweise durch den Prozentsatz entdeckter Stuck-at-Fehler bestimmt.

E.12 Modularisierung

ANMERKUNG Siehe auch C.2.8 „Geheimnisprinzip/Kapselung“, C.2.9 „Modularer Ansatz“ und E.3 „Strukturierte Beschreibungsmethodik“.

Ziel: Modulare Beschreibung der Funktionen des Schaltkreises.

Beschreibung: Klare Unterteilung der Gesamtfunktionalität in verschiedene Module mit begrenzten Funktionen. Damit wird die Transparenz der Module mit genau definierter Schnittstelle gebildet. Jedes Teilsystem ist an allen Ebenen des Entwurfs klar definiert und von eingeschränkter Größe (nur einige Funktionen). Die Schnittstellen zwischen den Teilsystemen werden so einfach wie möglich gehalten und gemeinsame Bereiche (d. h. gemeinsame Daten, Austausch von Informationen) werden minimiert. Die Komplexität der einzelnen Teilsysteme wird ebenfalls eingeschränkt.

E.13 Testabdeckung der Verifikationsszenarien (Testbenches)

Ziel: Quantitative und qualitative Bewertung der angewendeten Verifikationsszenarien während der Funktionstests.

Beschreibung: Die Qualität der Verifikationsszenarien, die während der Funktionstests definiert wird, d. h. die angewendeten Testmuster (Stimuli) zur Verifikation der festgelegten Funktion einschließlich aller Sonderfälle, sofern sie bestehen, sollte qualitativ und/oder quantitativ dokumentiert werden. Während eines quantitativen Ansatzes sollten die erreichte Testabdeckung und die Tiefe der angewendeten Funktionstests dokumentiert werden. Die resultierende Testabdeckung sollte den Zielwerten, die für jede der Testabdeckungs-Metriken ermittelt wurden, entsprechen. Jede Ausnahme wird dokumentiert. Im Falle eines qualitativen Ansatzes sollte die Anzahl der verifizierten Codezeilen, Befehle oder Pfade („Testabdeckung des Codes“) des zu verifizierenden Codes des Schaltkreises abgeschätzt werden.

ANMERKUNG Eine ausschließliche Analyse der Testabdeckung des Codes hat wegen der hohen Parallelität der Hardwarebeschreibung nur eine eingeschränkte Relevanz und wird durch erschöpfende Überprüfungen gerechtfertigt. Die Testabdeckung des Codes dient allgemein zum Nachweis des nicht abgedeckten funktionalen Codes.

E.14 Beachtung von Programmierrichtlinien

Ziel: Strikte Überwachung des Programmierstils ergibt einen syntaktisch und semantisch korrekten Code des Schaltkreises.

Beschreibung: Syntaktische Programmierregeln helfen, einen leicht lesbaren Code zu schaffen, und ermöglichen eine bessere Dokumentation einschließlich der Versionskontrolle. Üblicherweise können die Regeln zur Organisation und Kommentierung der Befehlsblöcke oder Module hier erwähnt werden.

Semantische Programmierregeln helfen, typische Probleme bei der Implementierung durch Vermeidung von Konstrukten, die zur fehlerhaften Synthese mit der mehrdeutigen Implementierung der Funktion des Schaltkreises führen, zu vermeiden. Typische Regeln sind zum Beispiel die Vermeidung von asynchronen Konstrukten oder Konstrukten, die unvorhersehbare zeitliche Abläufe erzeugen. Der Gebrauch von Latches oder die Kopplung von Daten mit Takt-Signalen führen im Allgemeinen zu solchen Mehrdeutigkeiten.

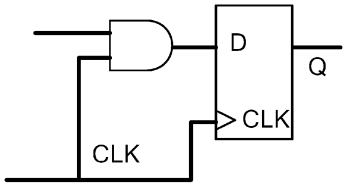
Entwurfsrichtlinien werden zur Vermeidung von systematischen Ausfällen des Entwurfs während des ASIC-Entwicklungsprozesses empfohlen. Ein Programmierstil beschränkt in bestimmter Hinsicht die Effizienz des Entwurfs, bietet jedoch im Gegensatz den Vorteil der Vermeidung von Ausfällen während des ASIC-Entwicklungsprozesses an. Diese sind im Besonderen:

– Vermeidung der typischen Programmierschwäche oder Ausfall;
– einschränkende Verwendung von problematischen Konstrukten, die mehrdeutige Synthese-Ergebnisse erzeugen;
– Entwurf für Testbarkeit;
– transparenter und leicht zu verwendender Code.

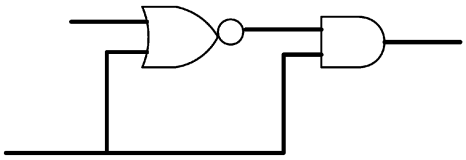
Beispiel eines Programmierstils

1. Der Code sollte so viele Kommentare wie notwendig enthalten, um die Funktion und die Details der Implementierung zu verstehen. Die verwendeten Vereinbarungen müssen vor dem Beginn des Entwurfs definiert werden. Die Übereinstimmung mit den definierten Vereinbarungen sollte während der Entwurfsphase überprüft werden.
   1.1. Standard-Header enthalten die Versionshistorie, Querverweise zur Spezifikation, Verantwortlichkeiten und begleitende Informationen des Entwurfs wie Versionsnummer, Änderungsanforderungen usw.
   1.2. Leicht lesbare Vorlagen: gleichwertige Prozesse sollten nach demselben Verfahren beschrieben werden, d. h. Verwendung von vordefinierten Vorlagen für wiederkehrende Prozesse (if then else, for usw.).
   1.3. Genaue und lesbare Vereinbarungen zur Namensgebung, zum Beispiel Groß-/Kleinbuchstaben, Präfix und Postfix, genaue Unterscheidung zwischen Port-Namen, interne Signale, Konstanten, Variablen, aktive Zustände bei niedrigem Energieniveau (xxx_n) usw.
   1.4. Die Einschränkung der Modul-Größe und die Anzahl der Ports je Modul sollten begrenzt werden, um die Lesbarkeit des Codes zu erhöhen.
   1.5. Entwicklung von strukturierten und defensiven Codes, z. B. sollten Zustandsinformationen in FSM (Geheimnisprinzip) gekapselt werden, um die leichte Modifikation des Codes zur ermöglichen.
   1.6. Plausibilitätstests wie Bereichsüberprüfungen usw. sollten ausgeführt werden.
   1.7. Vermeidung von folgenden Konstrukten/Befehlen:
      – Verwendung von aufsteigenden Bereichen (x zu y) für Bussignale;
      – Befehl „disable“ in Verilog (entspricht dem Befehl goto);
      – mehrdimensionale Felder (> 2), Records;
      – Kombination von vorzeichenbehafteten und nicht vorzeichenbehafteten Datentypen.

2. Vollständig synchroner Entwurf (von einem zentralen Takt abgeleitete Takte sind zulässig)
   2.1 Ausgänge der Module sollten synchronisiert werden. Dies unterstützt auch die Testbarkeit und die statische Analyse des Zeitverhaltens.
   2.2 Verknüpfte Takte sollten mit speziellen Vorsichtsmaßnahmen behandelt werden.
3. Die Vermeidung der Kopplung von Daten mit Takten erhöht die Testbarkeit, Reproduzierbarkeit zwischen den Daten des Prä- und Postlayouts und die Übereinstimmung mit dem Verhalten auf Registertransferebene (en: register transfer level, RTL).
4. Redundante Logik ist nicht prüfbar und sollte vermieden werden:



Kopplung von Daten und Taktsignalen



Redundante Logik

5. Rückführzweige in kombinatorischer Logik sollten vermieden werden, weil sie einen instabilen Entwurf erzeugen und nicht prüfbar sein werden.
6. Ein Full-Scan-Entwurf wird empfohlen.
7. Die Vermeidung von Latches erhöht die Testbarkeit und reduziert die zeitlichen Einschränkungen während der Synthese.
8. Der Master-Reset und alle asynchronen Eingänge sollten mit zwei aufeinander folgenden Speicherelementen (Flip-Flops) oder einem gleichwertigen Schaltkreis (Metastabilität) synchronisiert werden.
9. Es wird empfohlen, asynchrones Setzen oder Zurücksetzen mit Ausnahme des Master-Resets zu vermeiden.
10. Die Signale an den Modulports sollten vom Typ std_logic oder std_logic_vector sein.

E.15 Verwendung eines Codecheckers

Ziel: Automatische Verifikation der Programmierregeln („Programmierstil“) mittels eines Codecheckers.

Beschreibung: Die Anwendung eines Codecheckers hilft in hohem Maße, den Programmierstil automatisch zu überwachen, und erzeugt eine Online-Dokumentation. Im Allgemeinen kann jedoch ein automatischer Codechecker die Syntax und die Semantik des Codes überprüfen. Die Anwendung solcher Werkzeuge sollte deshalb durch die Erweiterung von allgemeinen Programmierregeln („Werkzeug spezifisch“) mit projektspezifischen Programmierregeln, die der Entwickler erstellen und getrennt bewerten muss, begleitet werden.

E.16 Defensive Programmierung

Siehe C.2.5.

E.17 Dokumentation von Simulationsergebnissen

Ziel: Dokumentation aller für eine erfolgreiche Simulation erforderlichen Daten, um die spezifizierte Funktion des Schaltkreises zu verifizieren.

Beschreibung: Alle Daten, die für die funktionale Simulation auf Module-, Bauteil- oder Systemebene erforderlich sind, sollten gut dokumentiert und mit den folgenden Zielen archiviert werden:

– Wiederholung der Simulation zu jeder späteren Phase in schlüsselfertiger Art.
– Nachweis der Korrektheit und Vollständigkeit aller spezifizierten Funktionen.

Die folgende Datenbasis sollte für diesen Zweck archiviert werden:

– Konfiguration der Simulation einschließlich der vollständigen Software der verwendeten Werkzeuge, zum Beispiel des Simulators, des Synthesizers mit der entsprechenden Versionsnummer und der notwendigen Simulationsbibliothek.
– Protokolldateien der Simulation mit vollständigen Details bezüglich der Zeit der Simulation, angewendeten Werkzeugen mit Versionsnummer und dem vollständigen Bericht der zugehörigen Arbeit, sofern sie notwendig war.
– Alle relevanten Simulationsergebnisse einschließlich dem Signalfluss, besonders im Falle der manuellen Inspektion, und der Dokumentation der erworbenen Ergebnisse.

E.18 Codeinspektion

ANMERKUNG 1 Siehe auch C.5.14 „Formale Inspektionen“.

Ziel: Überprüfung der Beschreibung des Schaltkreises.

Beschreibung: Die Überprüfung der Beschreibung des Schaltkreises sollte ausgeführt werden durch die

– Überprüfung des Programmierstils;
– Verifikation der beschriebenen Funktionalität gegen die Spezifikation;
– Überprüfung der defensiven Codierung, der Behandlung der Fehler und Sonderfälle.

ANMERKUNG 2 Wenn die (V)HDL-Simulation nicht ausgeführt wird, sollte die Vollständigkeit der Codeinspektion und der erreichten Ergebnisse die gleichwertige Qualität haben, die mit der (V)HDL-Simulation erreicht würde.

E.19 Walkthrough

ANMERKUNG 1 Siehe auch C.5.15 „Walkthrough“.

Ziel: Überprüfung der Beschreibung des Schaltkreises mittels Walkthrough.

Beschreibung: Ein Walkthrough des Codes wird von einer Arbeitsgruppe ausgeführt, die einige Testfälle mit einer repräsentativen Auswahl von Eingaben und entsprechend erwarteten Ausgaben für das Programm auswählt. Die Testdaten werden anschließend manuell Schritt für Schritt durch die Logik des Programms verfolgt.

ANMERKUNG 2 Als alleinige Maßnahme sollte diese Maßnahme nur auf sehr einfache Schaltkreise angewendet werden. Im Falle einer fehlenden (V)HDL-Simulation sollten die Vollständigkeit des Walkthroughs und die Qualität der erreichten Ergebnisse die gleichwertige Qualität haben, die mit der (V)HDL-Simulation erreicht wird.

Literaturhinweis:

IEC 61160:2005, *Design review*

E.20 Anwendung validierter Soft-Cores

Ziel: Vermeidung von Ausfällen während des Betriebs der Soft-Cores durch Anwendung von validierten Soft-Cores.

Beschreibung: Bei der Validierung der Soft-Cores durch den Hersteller sollten folgende Anforderungen erfüllt werden:

– Die Validierung des Soft-Cores sollte für den Betrieb des sicherheitsbezogenen Systems durchgeführt werden mit mindestens einem gleichwertigen oder höheren Sicherheits-Integritätslevel als das in Planung befindliche System.
– Allen Annahmen und Einschränkungen, die für die Validierung des Soft-Cores notwendig sind, sollte nachgekommen werden.

– Alle für die Validierung des Soft-Cores notwendige Dokumente sollten leicht verfügbar sein, siehe auch E.17 „Dokumentation von Simulationsergebnissen“.
– Jede Spezifikation des Herstellers sollte strikt überwacht werden und der Nachweis der Übereinstimmung sollte dokumentiert werden.

E.21 Validierung von Soft-Cores

ANMERKUNG Siehe auch E.6 „Funktionstest auf Modulebene“.

Ziel: Vermeidung von Ausfällen während des Betriebs des Soft-Cores durch Validierung des Soft-Cores während des Entwurfslebenszyklus.

Beschreibung: Wenn der Soft-Core nicht explizit für den Betrieb in einem sicherheitsbezogenen System entwickelt wird, sollte der erzeugte Code unter den gleichen Voraussetzungen validiert werden, die bei der Validierung jedes Source-Codes angewendet werden. Dies bedeutet, dass alle möglichen Testfälle definiert und ausgeführt werden sollten. Die funktionale Verifikation sollte dann durch Simulation abgeleitet werden.

E.22 Simulation der Gatter-Netzliste zur Überprüfung der Zeitvorgaben

Ziel: Unabhängige Verifikation der während der Synthese ermittelten Zeitvorgaben.

Beschreibung: Simulation der Gatter-Netzliste, erzeugt durch die Synthese einschließlich der Back-Annotation von Leitungsverzögerungen und Gatterverzögerungen. Zur Stimulation des Schaltkreises sollten die Stimuli derart hergeleitet werden, dass sie einen hohen Prozentsatz der Zeitvorgaben abdecken und alle Zeit-Pfade des ungünstigsten Falls einschließen werden. Im Allgemeinen stellen die für die Durchführung von E.6 „Funktionstest auf Modulebene“ oder E.7 „Funktionstest“ benötigten Stimuli geeignete Kriterien für die Auswahl der Stimuli zur Verfügung, vorausgesetzt, dass eine ausreichende Testabdeckung während der Funktionsprüfung beansprucht werden kann. Der Schaltkreis sollte unter Bedingungen des günstigsten und des ungünstigsten Falls mit der maximal spezifizierten Taktrate geprüft werden.

Die Verifikation des zeitlichen Verhaltens kann durch die automatische Überprüfung der Konfiguration und der Haltezeit der Speicherelemente (Flip-Flops) der Zielbibliothek sowie durch die funktionale Verifikation des Schaltkreises ausgeführt werden. Die funktionale Verifikation sollte vornehmlich durch Überwachung der Ausgänge des Bausteins durchgeführt werden. Dies kann durch Vergleich der Ausgangssignale des Schaltkreises mit einem entsprechenden Referenzmodell oder dem (V)HDL-Source-Code des Schaltkreises automatisiert werden. Dieser Test ist bekannt als „Regressionstest“ und sollte gegenüber einer manuellen Überprüfung der Ausgangssignale bevorzugt werden.

ANMERKUNG Durch Anwenden dieser Maßnahme kann das zeitliche Verhalten nur der Pfade verifiziert werden, die tatsächlich während der Simulation stimuliert werden. Deshalb kann die vorgestellte Maßnahme im Allgemeinen nicht eine vollständige Analyse des zeitlichen Verhaltens des Schaltkreises zur Verfügung stellen.

E.23 Statische Laufzeitanalyse (STA)

Ziel: Unabhängige Verifikation der während der Synthese erfassten Zeitvorgaben.

Beschreibung: Die Statische Laufzeitanalyse (en: static timing analysis, STA) analysiert alle Pfade einer Netzliste (Schaltkreis), erzeugt durch das Synthese-Werkzeug unter Berücksichtigung der Back-Annotation, d. h. sowohl die durch das Synthese-Werkzeug abgeschätzten Leitungsverzögerungen als auch die Gatterverzögerungen, ohne die tatsächliche Simulation durchzuführen. Im Allgemeinen erlaubt sie deshalb eine vollständige Analyse der Zeitvorgaben des gesamten Schaltkreises. Der zu prüfende Schaltkreis sollte unter Bedingungen des günstigsten und des ungünstigsten Falls beim Betrieb mit der maximal spezifizierten Taktrate und unter Berücksichtigung des zulässigen Takt-Jitters und Arbeitszyklusversatzes analysiert werden. Die Anzahl der nicht relevanten zeitlichen Pfade kann auf ein bestimmtes Minimum beschränkt werden, indem ein geeignetes Entwurfsverfahren angewendet wird. Es wird empfohlen, das angewendete Verfahren, das leicht lesbare Ergebnisse vor Beginn des Entwurfs ermöglicht, zu untersuchen, zu analysieren und zu definieren.

ANMERKUNG Es kann angenommen werden, dass STA alle vorhandenen zeitlichen Pfade ausführlich abdeckt, wenn

a) die Zeitvorgaben genau spezifiziert werden;

b) der zu testende Schaltkreis nur solche zeitlichen Pfade enthält, die durch STA-Werkzeuge, d. h. dies ist im Allgemeinen mit vollständig synchronen Schaltkreisen der Fall, analysiert werden können.

E.24 Vergleich der Gatter-Netzliste gegen ein Referenzmodell durch Simulation

Ziel: Funktionale Äquivalenzprüfung der synthetisierten Gatter-Netzliste.

Beschreibung: Simulation der Gatter-Netzliste, erzeugt durch das Synthese-Werkzeug. Die zur Verifikation des Schaltkreises durch Simulation angewendeten Stimuli entsprechen genau den Stimuli, die während des E.6 „Funktionstests auf Modulebene“ und des E.7 „Funktionstest auf oberster Ebene“ für die Verifikation der Funktion auf Modulebene beziehungsweise auf oberster Ebene angewendet wurden. Die funktionale Verifikation sollte vornehmlich durch Überwachung der Ausgänge des Bausteins durchgeführt werden. Dies kann durch Vergleich der Ausgangssignale des Schaltkreises mit einem entsprechenden Referenzmodell oder dem (V)HDL-Source-Code des Schaltkreises automatisiert werden. Dieser Test ist bekannt als „Regressionstest“ und sollte gegenüber einer manuellen Überprüfung der Ausgangssignale bevorzugt werden.

ANMERKUNG Durch Anwenden dieser Maßnahme kann das funktionale Verhalten nur der Pfade verifiziert werden, die tatsächlich während der Simulation stimuliert werden. Deshalb kann die Testabdeckung nur so gut sein, wie während der ursprünglichen Funktionstests auf Modulebene beziehungsweise auf oberster Ebene. Es ist möglich, diese Maßnahme mit einer formalen Äquivalenzprüfung zu ergänzen. In allen Fällen sollte eine funktionale Verifikation des (V)HDL-Source-Codes mit der endgültigen durch das Synthese-Werkzeug erzeugten Netzliste durchgeführt werden.

E.25 Vergleich der Gatter-Netzliste mit dem Referenzmodell (formale Äquivalenzprüfung)

Ziel: Funktionale Äquivalenzprüfung, die von der Simulation unabhängig ist.

Beschreibung: Vergleich der Funktionalität des Schaltkreises, die durch den (V)HDL-Source-Code beschrieben ist, mit der Funktionalität des Schaltkreises der Gatter-Netzliste, erzeugt durch die Synthese. Die auf dem Prinzip der formalen Äquivalenz basierten Werkzeuge sind dazu fähig, die funktionale Äquivalenz von verschiedenen Darstellungsformen des Schaltkreises, zum Beispiel (V)HDL-Beschreibung oder Beschreibung der Netzliste, zu verifizieren. Bei Anwendung dieser Maßnahme ist eine funktionale Simulation nicht notwendig und eine unabhängige funktionale Überprüfung ist möglich. Die erfolgreiche Anwendung dieser Maßnahme kann nur gewährleistet werden, wenn das angewendete Werkzeug dazu fähig ist, vollständige Äquivalenz nachzuweisen, und alle gemeldeten Unstimmigkeiten entweder durch manuelle Inspektion oder automatisiert beurteilt werden.

ANMERKUNG Es ist von Vorteil, diese Maßnahme mit E.24 „Vergleich der Gatter-Netzliste gegen ein Referenzmodell durch Simulation“ zu kombinieren. In allen Fällen sollte eine funktionale Verifikation des (V)HDL-Source-Codes mit der endgültigen durch das Synthese-Werkzeug erzeugten Netzliste durchgeführt werden.

E.26 Überprüfung der Anforderungen und Beschränkungen des Herstellers

Ziel: Vermeidung von Ausfällen während der Produktion durch Überprüfung der Herstelleranforderungen.

Beschreibung: Eine sorgfältige Überprüfung der Anforderungen und Beschränkungen des Herstellers durch das Synthese-Werkzeug, zum Beispiel minimaler und maximaler Fan-In und Fan-Out, maximale Leitungslänge (Leitungsverzögerung), maximale Flankensteilheit der Signale, Zeitversatz und so weiter, erhöht die Zuverlässigkeit des Produktes. Neben der Bedeutung der Anforderungen für den Produktionsprozess hat ihre Nichteinhaltung einen großen Einfluss auf die Validität der für die Simulation angewendeten Modelle. Die Nichteinhaltung der Anforderungen und Einschränkungen des Herstellers kann zu fehlerhaften Simulationsergebnissen führen, die unerwünschte Funktionalitäten hervorrufen.

E.27 Dokumentation der Synthesevorgaben, Ergebnisse und Werkzeuge

Ziel: Dokumentation aller definierten Beschränkungen, die für eine optimale Synthese notwendig sind, um die endgültige Gatter-Netzliste zu erzeugen.

Beschreibung: Die Dokumentation aller Beschränkungen und Ergebnisse der Synthese ist aus folgenden Gründen unverzichtbar:

– Reproduktion der Synthese zu jeder späteren Phase;

– Erzeugung unabhängiger Syntheseergebnisse zur Verifikation.

Unverzichtbare Dokumente sind:

– Konfiguration der Synthese einschließlich der eingesetzten Werkzeuge und Synthese-Software mit der aktuellen Versionsnummer, der angewendeten Synthese-Bibliothek und den definierten Beschränkungen und Skripten;

– Synthese-Protokolldatei mit den zeitlichen Hinweisen, eingesetztes Werkzeug mit Versionsnummer und vollständige Dokumentation der Synthese;

– die erzeugte Netzliste mit den abgeschätzten zeitlichen Verzögerungen (SDF-Datei, en: Standard Delay Format).

E.28 Verwendung von betriebsbewährten Synthesewerkzeugen

Ziel: Werkzeug-gestützte Konvertierung der (V)HDL-Beschreibung eines Schaltkreises in eine Gatter-Netzliste.

Beschreibung: Werkzeug-gestützte Zuordnung des (V)HDL-Source-Codes der Schaltkreis-Funktionalität durch Verbindung der entsprechenden Gatter und Schaltkreis-Primitiven der Bibliothek des Ziel-ASICs. Die aus einer Vielfalt von möglichen Ausführungen gewählte Ausführung, die die gewünschte Funktionalität erfüllt, ist vom optimalsten Ergebnis abhängig, das durch die Synthese-Beschränkungen wie zeitliches Verhalten (Takt-Rate) und Chip-Fläche hergeleitet wird.

E.29 Verwendung von betriebsbewährten Zielbibliotheken

ANMERKUNG Siehe auch E.4 „Betriebsbewährte Werkzeuge“.

Ziel: Vermeidung von systematischen Ausfällen verursacht durch eine fehlerhafte Zielbibliothek.

Beschreibung: Die Synthese und Simulationszielbibliothek zur Entwicklung eines ASICs werden aus einer gemeinsamen Datenbank hergeleitet und sind deshalb nicht unabhängig. Ein systematischer Ausfall wie:

– Mehrdeutigkeit zwischen dem realen und modellierten Verhalten der Schaltkreis-Elemente,

– das ungenügende Modellieren zum Beispiel der Konfiguration und Haltezeit,

ist eines der typischen Beispiele.

Deshalb sollten nur „betriebsbewährte“ Verfahren und Zielbibliotheken für den Entwurf von ASICs verwendet werden, die Sicherheitsfunktionen ausführen. Dies bedeutet:

– die Anwendung von Zielbibliotheken, die im Laufe einer erheblich langen Zeit in Projekten mit vergleichbarer Komplexität und zeitlicher Belastung verwendet worden sind.

– Verfügbarkeit der Verfahren und der zugehörigen Zielbibliotheken über einen ausreichend langen Zeitraum, so dass eine hinreichende Genauigkeit der Modellierung der Bibliothek erwartet werden kann.

E.30 Skriptbasierende Verfahren

Ziel: Reproduzierbarkeit von Ergebnissen und Automatisierung der Synthese-Zyklen.

Beschreibung: Automatisierte und skriptbasierende Steuerung der Synthese-Zyklen einschließlich der Definition der verwendeten Beschränkungen. Außer zur genauen Dokumentation einer vollständigen Beschränkung der Synthese hilft es, die Netzliste nach der Modifikation des (V)HDL-Source-Codes unter identischen Bedingungen zu reproduzieren.

E.31 Implementierung von Teststrukturen

Ziel: Entwurf von prüfbaren ASICs, der den abschließenden Produktionstest gewährleistet.

Beschreibung: Der Entwurf für Testbarkeit ermöglicht leicht prüfbare Schaltkreise mittels Implementierung von verschiedenen Teststrukturen, zum Beispiel:

– Scan-Path: Bei diesem abtastenden Verfahren werden entweder alle (vollständiger Scan-Path) oder ein Teil der Flip-Flops (teilweiser Scan-Path) in einer Einzel- oder Mehrfachkette hintereinander geschaltet und bilden eine Kette von Schieberegistern. Der Scan-Path erlaubt eine automatische Erzeugung der Testmuster der gesamten Logik eines Schaltkreises. Das Werkzeug, das die Testmuster erzeugt, wird ATPG „Automatischer Testmuster-Generator (en: automatic test pattern generator)“ genannt. Die Implementierung des Scan-Paths verbessert erheblich die Testbarkeit eines Schaltkreises und ermöglicht mehr als 98 % Tastabdeckung bei angemessenem Aufwand. Es wird deshalb empfohlen, wenn möglich, einen vollständigen Scan-Path zu implementieren.

– NAND-Tree: In einem NAND-Tree werden alle primären Eingänge eines Schaltkreises in kaskadierender Weise miteinander verbunden, um eine Kette zu bilden. Durch Anwendung eines geeigneten Testmusters („Walking-Bit“) ist es möglich, das Schaltverhalten (zeitliches Verhalten und Ansprechpegel) der Eingänge zu prüfen. Das NAND-Tree-Verfahren bietet ein einfaches Mittel für die Beschreibung der primären Eingänge. Seine Implementierung wird empfohlen, wenn das Schaltverhalten des Schaltkreises sonst nicht geprüft werden kann.

– Integrierte Selbsttests (en: build-in self test, BIST): Der Selbsttest des Schaltkreises und insbesondere der Selbsttest des eingebetteten Speichers kann sehr effizient durch Implementierung eines Onchip-Testmuster-Generators durchgeführt werden. BIST ermöglicht durch Anwendung von pseudozufälligen Testmustern und Bewertung der Signatur der implementierten Schaltkreis-Struktur eine automatische Verifikation der Schaltkreis-Struktur. Besonders für Speichertests werden BIST als zusätzliche Maßnahme empfohlen. Der Scan-Path-Test kann durch einen BIST ersetzt werden.

– Überwachung des Ruhestroms (en: quiescent current test, IDDQ-Test): Ein statischer CMOS-Schaltkreis verbraucht Strom hauptsächlich während des Schaltvorgangs. Ein absolut fehlerfreier Schaltkreis verbraucht deshalb eine unwesentlich kleine Strommenge (< 1 µA, Leckage-Strom), so lange das Testmuster statisch gehalten wird. Der IDDQ-Test ist sehr effektiv und ermöglicht mehr als 50 % Tastabdeckung bereits nach Anwendung von einigen Testmustern. Der IDDQ-Test kann auf funktionale Testmuster sowie auf durch ATPG erzeugte Testmuster angewendet werden. Diese Testmethode hat sich erwiesen, in der Praxis am nützlichsten zu sein, und ist in der Lage, Ausfälle zu erkennen, die andere Tests selten oder gar nicht erkennen können. Diese Maßnahme sollte deshalb zusätzlich zu den regelmäßigen Produktionstests angewendet werden.

– Grenzpfadabtastung (en: boundary-scan): Testarchitektur zur Verifikation der Abhängigkeiten der Komponenten auf einer gedruckten Leiterplatte gemäß dem JTAG-Standard. Dieselbe Philosophie kann auch angewendet werden, um die Abhängigkeiten von Modulen auf Onchip-Ebene zu verifizieren. Der Boundary-Scan-Test wird in erster Linie empfohlen, die Testbarkeit der gedruckten Leiterplatte zu verbessern.

E.32 Abschätzung der Testabdeckung durch Simulation

Ziel: Bestimmung der erreichten Testabdeckung durch die während des Produktionstests implementierte Testarchitektur.

Beschreibung: Die durch den Scan-Path-Test, BIST, funktionale Testmuster oder jede andere Maßnahme erreichte Testabdeckung kann durch Fehler-Simulation bestimmt werden. Während der Fehler-Simulation wird ein Testmuster auf den Schaltkreis angewendet, in dem Fehler eingefügt werden. Eine fehlerhafte Reaktion des Schaltkreises auf die angewendeten Stimuli entspricht den eingefügten Fehlern und trägt somit zur Testabdeckung bei. Die Fehler-Simulation ermöglicht die Erkennung von den Stuck-at-Fehlern „Stuck-at-1“ und „Stuck-at-0“ und die erreichte Testabdeckung stellt die Qualität der angewendeten Testmuster dar. Die

Fehler-Simulation kann im Allgemeinen sehr effektiv eingesetzt werden, um zur Logik gehörende Fehler zu erkennen, die nicht Teil des Scan-Paths sind, zum Beispiel im Falle des teilweisen Scan-Paths.

E.33 Abschätzung der Testabdeckung durch Anwendung eines ATPG-Werkzeugs

Ziel: Bestimmung der Testabdeckung, die durch während des Produktionstests eingebrachte Testmuster (Scan-Path, BIST) erwartet werden kann.

Beschreibung: Zurzeit gibt es eine Vielfalt von Verfahren, die pseudozufällige oder algorithmische Testmuster für einen mit Scan-Path ausgeführten Schaltkreis erzeugen. Das Synthese-Werkzeug wie ATPG erstellt während der Synthese eine Zusammenstellung von unerkannten Fehlern. Die Testabdeckung kann somit abgeschätzt werden und legt die untere Grenze der erreichten Testabdeckung mit dem angewendeten Testmuster fest. Es ist wichtig zu beachten, dass die Testabdeckung auf die Schaltkreis-Logik begrenzt ist, die durch den Scan-Path abgedeckt wird. Die Module wie Speicher, BIST oder Teile des Schaltkreises, die in den Scan-Path nicht integriert sind, werden in der Abschätzung der Testabdeckung nicht berücksichtigt.

E.34 Nachweis der Betriebsbewährung für verwendete Hard-Cores

Ziel: Vermeidung von systematischen Ausfällen während der Verwendung der Hard-Cores.

Beschreibung: Ein Hard-Core wird im Allgemeinen als eine Black-Box betrachtet, die die gewünschte Funktionalität darstellt und die aus der Layout-Datenbasis der Zieltechnologie zusammengesetzt ist, die die gewünschte Schaltkreis-Komponente zur Verfügung stellt. Der mögliche funktionale Ausfall kann vergleichbar zu diskreten Bauteilen wie Standard-Mikroprozessoren, Speicher usw. behandelt werden. Der Betrieb solcher Hard-Cores ohne Verifikation der ordnungsgemäßen Funktionalität ist möglich, wenn für die angewendete Zieltechnologie der verwendete Core als betriebsbewährte Komponente betrachtet werden kann. Der Rest des Schaltkreises sollte dann intensiv verifiziert werden.

E.35 Verwendung validierter Hard-Cores

ANMERKUNG Siehe auch E.6 „Funktionstest auf Modulebene“.

Ziel: Vermeidung von systematischen Ausfällen während der Verwendung der Hard-Cores.

Beschreibung: Die Validierung des Cores sollte wegen der Komplexität des Cores und den während der Entwurfsphase auf Grundlage des (V)HDL-Source-Codes angenommenen Beschränkungen von den Herstellern durchgeführt werden. Die Validierung gilt nur für die Konfiguration und die Zieltechnologie der angewendeten Komponente.

E.36 Online-Tests der Hard-Cores

ANMERKUNG Siehe auch E.13 „Testabdeckung der Verifikationsszenarien (Testbenches)“.

Ziel: Vermeidung von systematischen Ausfällen während der Verwendung der Hard-Cores.

Beschreibung: Verifikation der ordnungsgemäßen Funktion und Implementierung der verwendeten Hard-Cores durch Anwendung von Online-Tests. Bei Anwendung dieser Maßnahme ist ein effizientes Testkonzept notwendig und die Beurteilung des angewendeten Konzepts sollte dokumentiert werden.

E.37 Design-Rule-Check (DRC)

Ziel: Verifikation der Entwurfsregeln des Herstellers.

Beschreibung: Verifikation des erzeugten Layouts hinsichtlich der Entwurfsregeln des Herstellers, zum Beispiel minimale Leitungslängen, maximale Leitungslängen und verschiedene Regeln bezüglich der Anordnung der Layout-Strukturen. Ein vollständiger und ordnungsgemäßer Durchlauf des DRC sollte im Detail dokumentiert werden.

E.38 Überprüfung des Layouts Versus Schematic (LVS)

Ziel: Unabhängige Verifikation des Layouts.

Beschreibung: LVS entnimmt die Funktionalität des Schaltkreises aus der Layout-Datenbasis und vergleicht die entnommenen Elemente des Schaltkreises einschließlich der Abhängigkeiten mit der Eingangs-Netzliste. Dies gewährleistet die Äquivalenz des Layouts des Schaltkreises mit der Netzliste, die die Funktionalität des Schaltkreises festlegt. Ein vollständiger und ordnungsgemäßer Durchlauf des LVS sollte im Detail dokumentiert werden.

E.39 Zusätzliche Reserven (> 20 %) für Prozesstechnologien mit weniger als 3 Jahren Anwendung

Ziel: Gewährleistung der Robustheit der implementierten Funktionalität des Schaltkreises sogar unter starken Fertigungs- und Parameter-Schwankungen.

Beschreibung: Das wirkliche Verhalten des Schaltkreises wird durch eine Anzahl sich überschneidender physikalischer Effekte besonders bei kleinen Strukturen (zum Beispiel unterhalb 0,5 µm) bestimmt. Wegen des Mangels an Detail-Kenntnissen und notwendigen Vereinfachungen kann ein genaues Modell der Elemente des Schaltkreises in der Tat nicht abgeleitet werden. Mit abnehmenden geometrischen Strukturen spielen Leitungsverzögerungen immer mehr eine dominierende Rolle. Signalverzögerungen entlang den Leitungen und Übersprechkapazitäten zwischen den Leitungen wachsen überproportional. Signalverzögerungen sind im Vergleich zu Gatter-Verzögerungen nicht länger vernachlässigbar. Geschätzte Leitungsverzögerungen geben ein zunehmendes Risiko mit abnehmenden geometrischen Strukturen wieder.

Es wird deshalb empfohlen, ausreichende Reserven (> 20 %) bezüglich minimaler und maximaler zeitlicher Beschränkungen bei Verwendung von Prozessen, die seit weniger als 3 Jahren verwendet werden, für die entworfenen Schaltkreise vorzusehen, um die ordnungsgemäße Funktion des Schaltkreises bei stark schwankenden Parametern während der Fertigung oder wegen des Mangels von präzisen Modellen zu gewährleisten.

E.40 Burn-In-Test

Ziel: Gewährleistung der Robustheit des hergestellten Chips. Aussondern frühzeitiger Ausfälle. Freiliegende Produkte auf Halbleitersubstraten müssen ihre Robustheit nicht durch den Burn-In, aber z. B. durch Belastungstests auf Wafer-Ebene, beweisen.

Beschreibung: Der Burn-In-Test sollte bei der höchsten zulässigen Betriebstemperatur (im Allgemeinen 125 °C) durchgeführt werden. Die Testdauer ist von dem zu erreichenden SIL-Wert oder von besonderen Burn-In-Empfehlungen, zum Beispiel vom ASIC-Hersteller, abhängig. Burn-In kann verwendet werden zum:

– Aussondern frühzeitiger Ausfälle (Beginn der Badewannen-Kurve mit abnehmender Ausfallrate);
– Nachweis, dass frühzeitige Ausfälle bereits während der Herstellung und Prüfung ausgesondert werden (d. h. dass Geräte, die die Serienfertigung verlassen haben, bereits im Bereich der konstanten Ausfallrate der Badewannenkurve liegen).

E.41 Verwendung von betriebsbewährten Schaltkreisfamilien

Ziel: Gewährleistung der Zuverlässigkeit des hergestellten Chips.

Beschreibung: Der Hersteller eines Sicherheitsentwurfs sollte ausreichende Erfahrung mit der Anwendung der verwendeten programmierbaren Technologie des Schaltkreises und bezüglich sich weiter entwickelnder Werkzeuge haben.

E.42 Betriebsbewährter Fertigungsprozess

Ziel: Gewährleistung der Zuverlässigkeit des hergestellten Chips.

Beschreibung: Ein betriebsbewährter Fertigungsprozess ist durch eine ausreichende Folge von Fertigungserfahrungen gekennzeichnet.

E.43 Qualitätskontrolle des Fertigungsprozesses

Qualitätsmaßnahmen und Überwachungsmechanismen während des Fertigungsprozesses der Schaltkreise sichern eine kontinuierliche Fertigungsüberwachung. Zum Beispiel optische oder elektrische Überwachung der Teststrukturen, Temperatur-Feuchtigkeits-Tests oder Temperatur-Tests (siehe IEC 60068-2-1, IEC 60068-2-2 usw.).

E.44 Produktions-Qualitätsprüfung des Schaltkreises

Die Qualität des Schaltkreises wird durch Ausführung ausgewählter Stress-Tests nachgewiesen, zum Beispiel Temperatur-Feuchtigkeits-Tests oder Temperatur-Tests (siehe IEC 60068-2-1, IEC 60068-2-2 usw.). Der Hersteller des Schaltkreises wird entsprechende Nachweise liefern.

E.45 Funktionale Qualitätsprüfung des Schaltkreises

Alle Schaltkreise werden funktional geprüft. Der Hersteller des Schaltkreises wird entsprechende Nachweise liefern.

E.46 Qualitätsnormen

Der ASIC-Hersteller sollte für ein ausreichendes Qualitätsmanagement sorgen, das zum Beispiel im Qualitäts- und Zuverlässigkeitshandbuch dokumentiert ist: ISO 9000-Zertifizierung oder Qualitätsbeurteilung des Standardlieferanten (en: standard supplier quality assessment, SSQA).

Anhang F
(informativ)

Definitionen der Eigenschaften der Software-Lebenszyklusphasen

Tabelle F.1 – Spezifikation der Anforderungen an die Sicherheit der Software

(siehe IEC 61508-3, 7.2 und IEC 61508-3, Tabelle C.1)

	Eigenschaft	Definition
1.1	Vollständigkeit in Bezug auf die durch die Software zu berücksichtigenden sicherheitsbezogenen Erfordernisse	Die Spezifikation der Anforderungen an die Sicherheit der Software berücksichtigt alle sicherheitsbezogenen Erfordernisse und Einschränkungen, die aus den früheren Phasen des Sicherheitslebenszyklus resultieren und der Software zugewiesen wurden. Sicherheitsbezogene Erfordernisse und Einschränkungen werden normalerweise in den Eingaben zur Spezifikationstätigkeit der Anforderungen an die Sicherheit der Software angegeben. Dies kann die Spezifikation davon einschließen, was die Software nicht ausführen darf oder vermeiden muss.
1.2	Korrektheit in Bezug auf die durch die Software zu berücksichtigenden sicherheitsbezogenen Erfordernisse	Die Spezifikation der Anforderungen an die Sicherheit der Software stellt eine geeignete Antwort auf die sicherheitsbezogenen Erfordernisse und Einschränkungen, die der Software zugewiesen wurden, bereit. Das Ziel ist, sicherzustellen, dass das, was spezifiziert wird, wirklich Sicherheit in allen notwendigen Bedingungen garantieren wird.
1.3	Freiheit von internen Spezifikationsfehlern, einschließlich Freiheit von Mehrdeutigkeiten	Interne Vollständigkeit und Konsistenz der Spezifikation der Anforderungen an die Sicherheit der Software: Bereitstellung aller notwendigen Informationen für alle Funktionen und Situationen, die von ihren Aussagen abgeleitet werden können; Formulierung keiner widersprechenden oder inkonsistenten Aussagen. Im Gegensatz zur Vollständigkeit und zur Konsistenz in Bezug auf sicherheitsbezogene Erfordernisse können interne Vollständigkeit und Konsistenz nur basierend auf der Spezifikation der Anforderungen an die Sicherheit der Software beurteilt werden.
1.4	Verständlichkeit der Anforderungen an die Sicherheit	Die Spezifikation der Anforderungen an die Sicherheit der Software ist für alle, die sie lesen müssen, ohne übermäßige Bemühung vollauf verständlich, selbst wenn sie nicht früh in das Projekt miteinbezogen worden sind, vorausgesetzt, dass sie das erforderliche Wissen haben. Ein wichtiges Ziel ist, die Verifikation und eventuell Modifikationen zu ermöglichen.
1.5	Freiheit von nachteiliger Beeinflussung durch Nichtsicherheitsfunktionen von durch die Software zu berücksichtigenden sicherheitsbezogenen Erfordernissen	Die Spezifikation der Anforderungen an die Sicherheit der Software vermeidet Anforderungen, die für die Sicherheit der EUC-Einrichtung nicht notwendig sind. Das Ziel ist, eine unnötige Komplexität im Entwurf und in der Ausführung der Software zu vermeiden, um das Risiko von Fehlern und von Funktionen, die für die Sicherheit nicht wichtig sind, aber diejenigen Funktionen, die für die Sicherheit wichtig sind, beeinflussen oder gefährden, zu vermindern.
1.6	Befähigung zur Bereitstellung einer Basis zur Verifikation und Validierung	Die Spezifikation der Anforderungen an die Sicherheit der Software führt zu Tests und Prüfungen, die einen Nachweis liefern, dass die Software die Spezifikation der Anforderungen an die Sicherheit der Software erfüllt.

Tabelle F.2 – Softwareentwurf und Softwareentwicklung: Entwurf der Softwarearchitektur

(siehe IEC 61508-3, 7.4.3 und IEC 61508-3, Tabelle C.2)

	Eigenschaft	Definition
2.1	Vollständigkeit in Bezug auf die Spezifikation der Anforderungen an die Sicherheit der Software	Der Entwurf der Softwarearchitektur berücksichtigt alle sicherheitsbezogenen Erfordernisse und Einschränkungen, die durch die Spezifikation der Anforderungen an die Sicherheit der Software erhoben werden.
2.2	Korrektheit in Bezug auf die Spezifikation der Anforderungen an die Sicherheit der Software	Der Entwurf der Softwarearchitektur stellt eine geeignete Antwort auf die festgelegten Anforderungen an die Sicherheit der Software bereit.
2.3	Freiheit von internen Entwurfsfehlern	Der Entwurf der Softwarearchitektur und die Dokumentation des Entwurfs sind frei von Fehlern, die unabhängig von irgendeiner festgelegten Anforderung an die Sicherheit der Software identifiziert werden können. Beispiele: Verklemmungen (en: deadlock), Zugang zu nicht autorisierten Betriebsmitteln, Mangel an Betriebsmitteln, interne Unvollständigkeit (d. h. nicht alle Situationen, die vom Entwurf selbst hergeleitet werden, können berücksichtigt werden).
2.4	Einfachheit und Verständlichkeit Vorhersagbarkeit des Verhaltens	Der Entwurf der Softwarearchitektur erlaubt in allen festgelegten Situationen eine korrekte und genaue Vorhersage der Arbeitsweise der Software. Insbesondere umfassen diese Situationen abweichende Situationen und Versagenssituationen. Vorhersagbarkeit deutet insbesondere an, dass die Arbeitsweise nicht von Elementen abhängt, die nicht vom Entwickler oder Anwender beeinflusst werden können.
2.5	Verifizierbarer und testbarer Entwurf	Der Entwurf der Softwarearchitektur und die Dokumentation des Entwurfs erlauben und ermöglichen die Erstellung des glaubhaften Nachweises, dass alle festgelegten Anforderungen an die Sicherheit der Software durch den Entwurf korrekt berücksichtigt worden sind und dass der Entwurf frei von internen Fehlern ist. Verifizierbarkeit kann abgeleitete Eigenschaften wie Einfachheit, Modularität, Klarheit, Testbarkeit, Nachweisbarkeit usw. andeuten, abhängig von den verwendeten Verfahren zur Verifikation.
2.6	Fehlertoleranz	Der Entwurf der Softwarearchitektur gibt Gewissheit, dass die Software ein sicheres Verhalten bei Anwesenheit von Fehlern haben wird (interne Fehler, Versagen des Bedieners oder Fehler der externen Systeme). Ein defensiver Entwurf kann aktiv oder passiv sein. Aktive defensive Entwürfe können Eigenschaften wie Erkennung, Aufzeichnung und Eingrenzung von Fehlern, abgestufte Funktionseinschränkung und Beseitigung von irgendwelchen unerwünschten Nebenwirkungen vor der Wiederaufnahme des Normalbetriebs beinhalten. Passive defensive Entwürfe beinhalten Eigenschaften, welche die Unzugänglichkeit zu bestimmten Arten von Fehlern oder bestimmten Bedingungen (Unmengen von Eingaben, bestimmte Daten und Zeiten) garantieren, ohne dass die Software irgendeine bestimmte Handlung durchführt.
2.7	Abwehr von Versagensfällen infolge gemeinsamer Ursache durch externe Vorfälle	Der Entwurf der Softwarearchitektur ermöglicht die Erkennung von Versagensarten infolge gemeinsamer Ursache und wirkungsvollen Vorkehrungen gegen das Versagen.

Tabelle F.3 – Softwareentwurf und Softwareentwicklung: unterstützende Werkzeuge und Programmiersprache

(siehe IEC 61508-3, 7.4.4 und IEC 61508-3, Tabelle C.3)

	Eigenschaft	Definition
3.1	Unterstützung der Erstellung von Software mit den erforderlichen Softwareeigenschaften	Mittel zur Fehlererkennung oder zur Beseitigung von fehleranfälligen Konstrukten.
3.2	Klarheit über den Betrieb und Funktionalität eines Werkzeugs	Die Bereitstellung der umfangreichen Abdeckung und Rückmeldung zu allen Aspekten des Betriebs des Werkzeugs.
3.3	Korrektheit und Reproduzierbarkeit der Ausgabe	Die Konsistenz und Genauigkeit des Werkzeugs für jede vorgenommene Eingabe.

Tabelle F.4 – Softwareentwurf und Softwareentwicklung: detaillierter Entwurf

(siehe IEC 61508-3, 7.4.5 und IEC 61508-3, 7.4.6 und IEC 61508-3, Tabelle C.4)

	Eigenschaft	Definition
4.1	Vollständigkeit in Bezug auf die Spezifikation der Anforderungen an die Sicherheit der Software	Methoden eines ausführlichen Softwareentwurfs und einer ausführlichen Softwareerstellung werden angewendet, die sicherstellen, dass die resultierende Software alle sicherheitsbezogene Erfordernisse und Einschränkungen, die der Software zugewiesen sind, berücksichtigt.
4.2	Korrektheit in Bezug auf die Spezifikation der Anforderungen an die Sicherheit der Software	Es sind bestimmte Nachweise vorhanden, um zu beanspruchen, dass die Anforderungen an die Sicherheit, die der Software zugewiesen sind, durch die entwickelte Software erfüllt worden sind.
4.3	Freiheit von internen Entwurfsfehlern	Die entwickelte Software ist frei von internen Fehlern. Beispiele: Verklemmungen (en: deadlock), Zugang zu nicht autorisierten Betriebsmitteln, Mangel an Betriebsmitteln.
4.4	Einfachheit und Verständlichkeit Vorhersagbarkeit des Verhaltens	Das Verhalten der entwickelten Software ist durch sachliches und überzeugendes Testen und Analyse vorhersagbar.
4.5	Verifizierbarer und testbarer Entwurf	Die entwickelte Software ist verifizierbar und testbar.
4.6	Fehlertoleranz/Fehlererkennung	Verfahren und Entwürfe geben Gewissheit, dass die entwickelte Software sich bei Anwesenheit von Fehlern sicher verhalten wird.
4.7	Freiheit von Versagensfällen infolge gemeinsamer Ursache	Verfahren und Entwürfe erkennen Versagensarten infolge gemeinsamer Ursache und stellen wirkungsvolle Vorkehrungen gegen Softwareversagen bereit.

Tabelle F.5 – Softwareentwurf und Softwareentwicklung: Test der Softwaremodule und Integration

(siehe IEC 61508-3, 7.4.7 und IEC 61508-3, 7.4.8 und IEC 61508-3, Tabelle C.5)

	Eigenschaft	Definition
5.1	Vollständigkeit des Testens und der Integration in Bezug auf die Spezifikation des Entwurfs	Das Testen der Software überprüft das Softwareverhalten gründlich genug, um sicherzustellen, dass alle Anforderungen der Spezifikation des Softwareentwurfs berücksichtigt worden sind.
5.2	Korrektheit des Testens und der Integration in Bezug auf die Spezifikation des Entwurfs (erfolgreicher Abschluss)	Das Testen der Module ist abgeschlossen. Es sind bestimmte Nachweise vorhanden, um zu beanspruchen, dass die Anforderungen an die Sicherheit erfüllt worden sind.
5.3	Reproduzierbarkeit	Bei Wiederholung der einzelnen Beurteilungen, die als Teil des Testens der Module und der Integration ausgeführt werden, werden gleich bleibende Resultate erzielt.
5.4	Genau definierte Konfiguration des Testens	Das Testen der Module und die Integration wurden mit den beanspruchten Ergebnissen auf die korrekte Version der Elemente und der Software angewendet. Sie ermöglichen, die Ergebnisse mit der bestimmten Konfiguration der Software, so wie sie erstellt wurde, zu verbinden.

Tabelle F.6 – Integration der programmierbaren Elektronik (Hardware und Software)

(siehe IEC 61508-3, 7.5 und IEC 61508-3, Tabelle C.6)

	Eigenschaft	Definition
6.1	Vollständigkeit der Integration in Bezug auf die Spezifikation des Entwurfs	Die Integration stellt die geeignete Tiefe und Abdeckung der Systemelemente bereit, um zu zeigen, dass sie die vorgesehenen Funktionen ausführen können und nicht vorgesehene Funktionen unter allen vorhersehbaren Betriebsbedingungen und bei Systemausfall nicht ausführen. Dies umfasst die angewendeten Prinzipien zur Verifikation, den zum Ziel gesetzten Level des Entwurfs und die Aspekte der Integration (zum Beispiel Verifikation der Vollständigkeit der Wechselwirkungen zwischen den Modulen).
6.2	Korrektheit der Integration in Bezug auf die Spezifikation des Entwurfs (erfolgreicher Abschluss)	Die Integration basiert auf korrekte Annahmen. Z. B. Korrektheit der erwarteten Ergebnisse, der Bedingungen der betrachteten Verwendung, der Repräsentativität der Testumgebung. Die Integration ist abgeschlossen. Es sind bestimmte Nachweise vorhanden, um zu beanspruchen, dass die Anforderungen an die Sicherheit erfüllt worden sind.
6.3	Reproduzierbarkeit	Bei Wiederholung der einzelnen Beurteilungen, die als Teil der Integration ausgeführt werden, werden gleichbleibende Resultate erzielt.
6.4	Genau definierte Konfiguration der Integration	Die Integration gibt eine angemessene Gewissheit, dass sie, wie dokumentiert, wirksam auf die korrekte Version der Elemente und der Software mit den beanspruchten Ergebnissen angewendet wurde. Sie ermöglicht, die Ergebnisse mit der bestimmten Konfiguration der Software, so wie sie erstellt wurde, zu verbinden.

Tabelle F.7 – Softwareaspekte bezüglich der Validierung der Systemsicherheit

(siehe IEC 61508-3, 7.7 und IEC 61508-3, Tabelle C.7)

	Eigenschaft	Definition
7.1	Vollständigkeit der Validierung in Bezug auf die Spezifikation des Softwareentwurfs	Die Validierung der Software berücksichtigt alle Anforderungen der Spezifikation des Softwareentwurfs.
7.2	Korrektheit der Validierung in Bezug auf die Spezifikation des Softwareentwurfs (erfolgreicher Abschluss)	Die Validierung der Software ist abgeschlossen. Es sind bestimmte Nachweise vorhanden, um zu beanspruchen, dass die Anforderungen an die Sicherheit erfüllt worden sind.
7.3	Reproduzierbarkeit	Bei Wiederholung der einzelnen Beurteilungen, die als Teil der Validierung der Software ausgeführt werden, werden gleichbleibende Resultate erzielt.
7.4	Genau definierte Konfiguration der Validierung	Klare und präzise Definition – des Systems, – der Anforderungen, – der Umgebung.

Tabelle F.8 – Softwaremodifikation

(siehe IEC 61508-3, 7.8 und IEC 61508-3, Tabelle C.8)

	Eigenschaft	Definition
8.1	Vollständigkeit der Modifikation in Bezug auf ihre Anforderungen	Die Modifikation wurde von autorisiertem Personal mit einem angemessenen Verständnis von ihren funktionalen, sicherheitsbezogenen, technischen und betrieblichen Auswirkungen ordnungsgemäß freigegeben.
8.2	Korrektheit der Modifikation in Bezug auf ihre Anforderungen	Die Modifikation erreicht ihre festgelegten Ziele.
8.3	Freiheit vom Einbringen interner Entwurfsfehler	Die Modifikation bringt keine neuen systematischen Fehler ein. Beispiele: Division durch Null, Indizes oder Zeiger außerhalb der Grenzen, Verwendung nicht initialisierter Variablen.
8.4	Vermeidung ungewollten Verhaltens	Die Modifikation bringt kein Verhalten ein, das entsprechend den in der Spezifikation der Anforderungen an die Sicherheit der Software festgelegten Einschränkungen vermieden werden muss.
8.5	Verifizierbarer und testbarer Entwurf	Der Entwurf der Software ist so, dass die Auswirkung der Modifikation geeignet ist, gründlich überprüft zu werden.
8.6	Regressionstest und Deckungsgrad der Verifikation	Der Entwurf der Software ist so, dass wirksame und gründliche Regressionstests möglich sind, um zu zeigen, dass die Software nach der Modifikation weiterhin der Spezifikation der Anforderungen an die Sicherheit der Software entspricht.

Tabelle F.9 – Softwareverifikation

(siehe IEC 61508-3, 7.9 und IEC 61508-3, Tabelle C.9)

	Eigenschaft	Definition
9.1	Vollständigkeit der Verifikation in Bezug auf die vorherige Phase	Die Verifikation ist geeignet festzustellen, dass die Software allen relevanten Anforderungen der Spezifikation der Anforderungen an die Sicherheit der Software entspricht.
9.2	Korrektheit der Verifikation in Bezug auf die vorherige Phase (erfolgreicher Abschluss)	Die Verifikation ist abgeschlossen. Es sind bestimmte Nachweise vorhanden, um zu beanspruchen, dass die Anforderungen an die Sicherheit erfüllt worden sind.
9.3	Reproduzierbarkeit	Bei Wiederholung der einzelnen Beurteilungen, die als Teil der Verifikation ausgeführt werden, werden gleichbleibende Resultate erzielt.
9.4	Genau definierte Konfiguration der Verifikation	Die Verifikation wurde mit den beanspruchten Ergebnissen auf die korrekte Version der Elemente und der Software angewendet. Sie ermöglicht, die Ergebnisse mit der bestimmten Konfiguration der Software, so wie sie erstellt wurde, zu verbinden.

Tabelle F.10 – Beurteilung der funktionalen Sicherheit

(siehe IEC 61508-3, Abschnitt 8 und IEC 61508-3, Tabelle C.10)

	Eigenschaft	Definition
10.1	Vollständigkeit der Beurteilung der funktionalen Sicherheit in Bezug auf diese Norm	Die Beurteilung der funktionalen Sicherheit der Software ergibt eine klare Aussage zum Umfang der erreichten Übereinstimmung, der getroffenen Entscheidungen, der abhelfenden Handlungen und empfohlenen Zeitskalen, der erzielten Aussagen und der Empfehlungen, die sich zur Abnahme ergeben, der qualifizierten Abnahme oder Ablehnung und zu jeder Zeit Einschränkungen, die auf diesen Empfehlungen beruhen.
10.2	Korrektheit der Beurteilung der funktionalen Sicherheit der Software in Bezug auf die Spezifikation des Entwurfs (erfolgreicher Abschluss)	Die Beurteilung der funktionalen Sicherheit der Software ist abgeschlossen. Es sind bestimmte Nachweise vorhanden, um zu beanspruchen, dass die Anforderungen an die Sicherheit erfüllt worden sind.
10.3	Nachweisbare Auflösung aller erkannter Probleme	Es gibt eine klare Aussage, in welchem Umfang die Probleme, die sich während der Beurteilung der funktionalen Sicherheit der Software ergaben, berücksichtigt wurden.
10.4	Fähigkeit zur Modifikation der Beurteilung der funktionalen Sicherheit der Software nach Änderungen ohne die Notwendigkeit einer umfangreichen Nachbearbeitung der Beurteilung	Die Beurteilung der funktionalen Sicherheit der Software ist geeignet, nachbearbeitet zu werden, um Teilen der Beurteilung der funktionalen Sicherheit der Software zu erlauben, nach Softwareänderungen neu beurteilt zu werden und revidierte Aussagen zu treffen, ohne die Notwendigkeit einer umfangreichen Nachbearbeitung der vollständigen Beurteilung der funktionalen Sicherheit der Software.

Tabelle F.10 *(fortgesetzt)*

	Eigenschaft	Definition
10.5	Reproduzierbarkeit	Die Beurteilung der funktionalen Sicherheit wird nach einem konsistenten, geplanten und offenen Prozess zu bestimmten Einzelpersonen und Dokumenten durchgeführt. Diese erlaubt die Prüfung der Grundlage der Beurteilung und der getroffenen Entscheidungen für alle diejenigen, die von ihren Entscheidungen betroffen sind, einschließlich Lieferanten, Anwender, Instandhalter und Behörden. Die Beurteilung der funktionalen Sicherheit erlaubt unabhängigen kompetenten Personen, die einzelnen Beurteilungen zu wiederholen, die als Teil der Beurteilung durchgeführt wurden.
10.6	Rechtzeitigkeit	Die Beurteilung der funktionalen Sicherheit wird mit einer zu den Phasen des Sicherheitslebenszyklus der Software passenden Häufigkeit ausgeführt, zumindest bevor die ermittelten Gefährdungen gegenwärtig sind. Sie stellt rechtzeitig einen Mängelbericht bereit. Die Ergebnisse der Tests, Inspektionen, Analysen usw. sind tatsächlich verfügbar, wenn sie als Eingabe für eine Entscheidung im Rahmen der Beurteilung erforderlich sind.
10.7	Genau definierte Konfiguration	Die Beurteilung der funktionalen Sicherheit der Software erlaubt, die Ergebnisse mit der bestimmten Konfiguration des Systems zu verbinden, das durch die Ergebnisse der Beurteilung der funktionalen Sicherheit bestätigt werden soll.

Anhang G
(informativ)

Anleitung zur Entwicklung von sicherheitsbezogener objektorientierter Software

Alle Empfehlungen dieser Norm für den Entwurf von Software gelten für objektorientierte Software. Weil der objektorientierte Ansatz Informationen unterschiedlich zum prozeduralen oder funktionalen Ansatz darstellt, enthält die folgende Auflistung die Empfehlungen, welche einer besonderen Betrachtung bedürfen:

– Verständnis der Klassenhierarchien und Bestimmung der Softwarefunktion(en), die bei Aufruf einer gegebenen Methode ausgeführt wird (werden) (einschließlich, wenn verwendend, einer bestehenden Klassenbibliothek);

– strukturbezogene Prüfungen (IEC 61508-3, Tabelle B.2 und IEC 61508-7, C.5.8).

Tabellen G.1 und G.2 liefern eine informative Anleitung zur Anwendung von objektorientierter Software, um die allgemeinere normative Anleitung zu ergänzen, die in IEC 61508-3, Tabellen A.2 und A.4 zur Verfügung gestellt wird.

Tabelle G.1 – Objektorientierte Softwarearchitektur

	Empfehlung	Details	SIL1	SIL2	SIL3	SIL4
G1.1	Nachvollziehbarkeit des Konzepts des Anwendungsbereiches zu den Klassen der Architektur	ANMERKUNG 1	+	++	++	++
G1.2	Verwendung von geeigneten Rahmen, allgemein verwendeten Kombinationen der Klassen und Entwurfsmuster ANMERKUNG Bei Verwendung bestehender Rahmen und Entwurfsmuster gelten die Anforderungen an vorentwickelter Software für diese Rahmen und Muster.	ANMERKUNG 2	+	++	++	++

ANMERKUNG 1 Nachvollziehbarkeit vom Anwendungsbereich zur Klassenarchitektur ist weniger wichtig.

ANMERKUNG 2 BEISPIEL 1: Für einen Teil des vorgesehenen sicherheitsbezogenen Projektes könnte ein Rahmen von einem nicht sicherheitsbezogenen Projekt bestehen, der eine ähnliche Aufgabe erfolgreich gelöst hat und der den Projektteilnehmern wohlbekannt ist. Dann wird die Verwendung dieses Rahmens empfohlen.

BEISPIEL 2: Es kann der Fall eintreten, dass verschiedene Algorithmen erforderlich sind, um eng gekoppelte Teilaufgaben des sicherheitsbezogenen Projektes zu lösen. Das Strategie-Muster kann ausgewählt werden, um auf die Algorithmen zuzugreifen.

BEISPIEL 3: Ein Teil des sicherheitsbezogenen Projektes kann daraus bestehen, geeignete Warnungen an interne und externe Projektbeteiligte auszugeben. Das Beobachter-Muster kann ausgewählt werden, um diese Warnungen zu organisieren. Die Anforderung gilt nicht für Bibliotheken.

ANMERKUNG 3 Es gibt im Allgemeinen eine abstrakte Basisklasse, die Zugriff auf die abgeleiteten konkreten Klassen zur Verfügung stellt.

Tabelle G.2 – Detaillierter objektorientierter Entwurf

	Empfehlung	SIL1	SIL2	SIL3	SIL4
G2.1	Klassen sollten nur eine Zielsetzung haben	+	+	++	++
G2.2	Vererbung wird nur verwendet, wenn die abgeleitete Klasse eine Verbesserung ihrer Basisklasse ist	++	++	++	++
G2.3	Vererbungstiefe wird durch Programmierrichtlinien eingeschränkt	+	+	++	++
G2.4	Überladen von Operationen (Methoden) unter strenger Kontrolle	+	++	++	++
G2.5	Mehrfachvererbung wird nur bei Schnittstellen-Klassen eingesetzt	++	++	++	++
G2.6	Vererbung von unbekannten Klassen			--	--
G2.7	Verifikation, dass die wiederverwendeten objektorientierten Bibliotheken den Empfehlungen dieser Tabelle entsprechen	++	++	++	++

ANMERKUNG 1 Mit anderen Worten: Eine Klasse ist dadurch gekennzeichnet, dass sie eine Verantwortung hat, d. h. sie verwaltet eng gekoppelte Daten und die Operationen mit diesen Daten.

ANMERKUNG 2 Sorgfalt ist erforderlich, um umlaufende Abhängigkeiten zwischen den Objekten zu vermeiden.

Die folgenden Begriffe, die oben verwendet werden, werden hier informell definiert.

Tabelle G.3 – Einige detaillierte objektorientierte Begriffe

Begriff	Informelle Definition
Basisklasse	Klasse, die abgeleitete Klassen hat. Eine Basisklasse wird mitunter Oberklasse oder Elternklasse genannt.
abgeleitete Klasse	Klasse (Anordnung von Eigenschaften und Operationen), die Eigenschaften und/oder Operationen von einer anderen Klasse (Basisklasse) erbt. Eine abgeleitete Klasse wird mitunter Subklasse oder Kindklasse genannt.
Rahmen	Struktur eines Programms, in vielen Fällen bereits entwickelt, um für spezielle Anwendungen ausgefüllt zu werden.
Überladen	Ersatz einer Operation (Methode, Unterprogramm) durch eine andere Operation (Methode, Unterprogramm) mit derselben Signatur und Vererbungs-Hierarchie zur Laufzeit. Merkmal von objektorientierten Sprachen oder Programmen. Realisiert Polymorphie.
Signatur einer Operation	Bezeichnung einer Operation (Unterprogramm, Methode) zusammen mit ihren Parametern (Argumenten) und deren Typen, gelegentlich auch deren Rückgabetypen. Zwei Signaturen sind identisch, wenn sie identische Parameter-Namen, Parameter-Anzahl und Parameter-Typen haben. In einigen Sprachen müssen auch die Rückgabetypen identisch sein.

Stichwortverzeichnis

A

Abgestufte Funktionseinschränkungen ... C.3.8
Abschätzung der Testabdeckung durch Anwendung eines ATPG-Werkzeugs ... E.33
Abschätzung der Testabdeckung durch Simulation ... E.32
Analogsignal-Überwachung ... A.2.7
Analyse von Ausfällen infolge gemeinsamer Ursache ... C.6.3
Anforderungen an die Leistungsfähigkeit ... C.5.19
Animation der Spezifikation und des Entwurfs ... C.5.26
Anregung und Antwort ... B.2.4.5
Antivalente Signalübertragung ... A.11.4
Anwendung validierter Soft-Cores ... E.20
Äquivalenzklassen und Test von Partitionen des Eingangsbereichs ... C.5.7
Assertion-Programmierung (en: failure assertion programming) ... C.3.3
Ausfallanalyse ... B.6.6
Ausfallarten- und Auswirkungsanalyse (FMEA) ... B.6.6.1
Auslösung der Sicherheitsabschaltung über eine thermische Sicherung ... A.10.3
Automatische Softwaregenerierung ... C.4.6

B

Beachtung von Programmierrichtlinien ... E.14
Beachtung von Richtlinien und Normen ... B.3.1
Belastungstest (en: Avalanche/stress testing) ... C.5.21
Benutzerfreundlichkeit ... B.4.2
Berechnung von kommunizierenden Systemen (en: Calculus of Communicating Systems, CCS) ... C.2.4.2
Betrieb nur durch erfahrene Bediener ... B.4.5
Betriebs- und Instandhaltungsanweisungen ... B.4.1
Betriebsbewährt ... C.2.10.1
Betriebsbewährte Werkzeuge und Compiler ... C.4.4
Betriebsbewährte Werkzeuge ... E.4
Betriebsbewährter Fertigungsprozess ... E.42
Beurteilung eines allgemeinen Verifikationsnachweises ... C.2.10.2
Black-Box-Test ... B.5.2
Blockwiederholung (zum Beispiel doppeltes ROM mit Hardware- oder Softwarevergleich) ... A.4.5
Burn-In-Test ... E.40

C

Checklisten ... B.2.5
Codeinspektion ... E.18
Codesicherung ... A.6.2
Codierte Verarbeitung (ein Kanal) ... A.3.4

D

Datenaufzeichnung und Analyse C.5.2
Datenflussanalyse C.5.10
Datenflussdiagramme C.2.2
Datenwege (interne Kommunikation) A.7
Defensive Programmierung C.2.5
Defensive Programmierung E.16
Design-Rule-Check (DRC) E.37
Diversitäre Hardware B.1.4
Diversitäre Programmierung C.3.5
Diversitäre Überwachungseinrichtung C.3.4
Dokumentation der Synthesevorgaben, Ergebnisse und Werkzeuge E.27
Dokumentation von Simulationsergebnissen E.17
Dokumentation B.1.2
Doppeltes RAM mit Hardware- oder Softwarevergleich und Schreib-/Lesetest A.5.7
Durchführung von Testfällen aus der Fehlererwartung („Fehler erraten“) C.5.5
Durchführung von Testfällen nach einer Grenzwertanalyse C.5.4
Durchführung von Testfällen nach Fehlereinpflanzung C.5.6
Dynamische Analyse und Test B.6.5
Dynamische Prinzipien A.2.2
Dynamische Rekonfiguration C.3.10

E

E/A-Einheiten und Schnittstellen (externe Kommunikation) A.6
Echtzeit-Yourdon C.2.1.4
Ein-Bit-Hardwareredundanz A.7.1
Ein-Bit-Redundanz (zum Beispiel RAM-Überwachung mit einem Parity-Bit) A.5.5
Einflussanalyse C.5.23
Eingabebestätigung B.4.9
Eingabevergleich/-entscheidung A.6.5
Eingeschränkte Betriebsmöglichkeiten B.4.4
Eingeschränkte Verwendung asynchroner Konstrukte E.9
Eingeschränkte Verwendung von Interrupts C.2.6.5
Eingeschränkte Verwendung von Rekursion C.2.6.7
Eingeschränkte Verwendung von Zeigern (en: pointers) C.2.6.6
Elektrik A.1
Elektronik A.2
Endliche Zustandsmaschinen, Zustandsübergangsdiagramme B.2.3.2
Energieabschaltung mit Sicherheitsabschaltung A.8.3
Entity-Relationship-Attribut-Datenmodelle B.2.4.4
Entscheidungs-/Wahrheitstabellen C.6.1

Entwurf für Testbarkeit ... E.11
Entwurfs- und Programmierrichtlinien ... C.2.6
Entwurfsbeschreibung in (V)HDL ... E.1
Entwurfsüberprüfung ... C.5.16
Ereignisbaumanalyse (en: event tree analysis, ETA) ... B.6.6.3
Erhöhung der Störfestigkeit ... A.11.3
Erkennung von Ausfällen durch Überwachung während des Betriebs ... A.1.1
Erweiterte Funktionstests ... B.6.8

F

Fehlerbaumanalyse (en: fault tree analysis, FTA) ... B.6.6.5
Fehlerbaummodelle ... B.6.6.9
Fehlererkennende und korrigierende Codes ... C.3.2
Fehlererkennung und Diagnose ... C.3.1
Felderfahrung ... B.5.4
Formale Inspektion ... C.5.14
Formale Methoden ... C.2.4, B.2.2
Formaler Beweis (Verifikation) ... C.5.12
Funktionale Qualitätsprüfung des Schaltkreises ... E.45
Funktionstest auf Modulebene ... E.6
Funktionstest auf oberster Ebene ... E.7
Funktionstest eingebettet in Systemumgebung ... E.8
Funktionstest unter Umgebungsbedingungen ... B.6.1
Funktionstest ... B.5.1

G

Geeignete Programmiersprache ... C.4.5
Gegenseitiger Vergleich durch Software ... A.3.5
Geheimnisprinzip/Kapselung ... C.2.8
Gestaffelte Meldung von Thermosensoren und bedingter Alarm ... A.10.4

H

Hardware mit automatischen Tests ... A.2.6

I

Implementierung von Teststrukturen ... E.31
Informationsredundanz ... A.7.6
Inspektion (Überprüfungen und Analysen) ... B.3.7
Inspektion der Spezifikation ... B.2.6
Inspektion durch Verwendung von Testmustern ... A.7.4
Instandhaltungsfreundlichkeit ... B.4.3

J

Jackson Systementwicklung (en: Jackson System Development, JSD) ... C.2.1.3

K

Kombination von zeitlicher und logischer Überwachung des Programmablaufs A.9.4
Kommunikation und Massenspeicher A.11
Kontrollflussanalyse C.5.9
Kontrollierte Anforderungsbeschreibung (en: Controlled Requirements Expression, CORE) C.2.1.2
Kreuzweise Überwachung mehrfacher Aktoren A.13.2
Künstliche Intelligenz – Fehlerkorrektur C.3.9

L

Logik höherer Ordnung (en: High Order Logic, HOL) C.2.4.4
Logische Überwachung des Programmablaufs A.9.3
LOTOS 2.4.5
Lüfterkontrolle A.10.2
Lüftung und Beheizung A.10

M

Markov-Modelle B.6.6.6
Maßnahmen gegen die Einwirkung der physikalischen Umgebung A.14
Mehr-Bit-Hardwareredundanz A.7.2
Mehrheitsentscheider A.1.4
Mehrkanalige parallele Ausgabe A.6.3
Methodenunspezifische Werkzeuge B.2.4.2
Modellbasiertes Testen (Testfallgenerierung) C.5.27
Modellierung der Leistungsfähigkeit C.5.20
Modellorientierte Verfahren mit hierarchischer Analyse B.2.4.3
Modellprüfung C.5.12.1
Modifizierte Prüfsumme A.4.2
Modularer Ansatz C.2.9
Modularisierung E.12, B.3.4
Monte-Carlo-Simulation B.6.6.8

N

Nachrichtenverlaufstabellen C.2.14
Nachvollziehbarkeit C.2.11
Nachweis der Betriebsbewährung für verwendete Hard-Cores E.34
Numerische Offline-Analyse C.2.13

O

OBJ C.2.4.6
Online-Test während der Erstellung von dynamischen Variablen oder dynamischen Objekten, Keine dynamische Variablen oder dynamische Objekte C.2.6.4, C.2.6.3
Online-Tests der Hard-Cores E.36

P

Produktions-Qualitätsprüfung des Schaltkreises E.44

Programmierrichtlinien C.2.6.2
Projektmanagement B.1.1
Prototypenerstellung/Animation C.5.17

Q

Qualitätskontrolle des Fertigungsprozesses E.43
Qualitätsnormen E.46

R

RAM-Test „Abraham“ A.5.4
RAM-Test „Checkerboard“ oder „March“ A.5.1
RAM-Test „Galpat“ oder „transparenter Galpat“ A.5.3
RAM-Test „Walkpath“ A.5.2
RAM-Überwachung mit einem modifizierten Hammingcode oder Erkennung von Datenfehlern mit fehlererkennenden und -korrigierenden Codes (en: error-detection-correction codes, EDC) A.5.6
Räumliche Trennung mehrfacher Leitungen A.11.2
Reaktionszeiten und Speicherbeschränkungen C.5.22
Rechnerunterstützte Entwurfswerkzeuge B.3.5
Rechnerunterstützte Spezifikationswerkzeuge B.2.4
Referenzsensor A.12.1
Regeneration durch Wiederholung C.3.7
Rückwärtsregeneration C.3.6
Ruhestromprinzip A.1.5

S

Schaltplaneingabe E.2
Schnittstellenprüfung C.5.3
Schutz gegen Irrtümer des Bedieners B.4.6
Schutz vor Modifikation B.4.8
Selbsttest durch Software: begrenzte Anzahl von Mustern (ein Kanal) A.3.1
Selbsttest durch Software: Walking Bit (ein Kanal) A.3.2
Selbsttest unterstützt durch Hardware (ein Kanal) A.3.3
Semi-formale Methoden B.2.3
Sensoren A.12
Sequentielle Kommunikationsprozesse (en: Communicating Sequential Processes, CSP) C.2.4.3
Sicherheit und Leistungsfähigkeit in Echtzeit: zeitgesteuerte Architektur C.3.11
Signatur mit doppelter Wortbreite (16 Bit) A.4.4
Signatur mit einfacher Wortbreite (8 Bit) A.4.3
Simulation der Gatter-Netzliste zur Überprüfung der Zeitvorgaben E.22
Simulation des Prozesses C.5.18
Simulation B.3.6
Skriptbasierende Verfahren E.30
Softwarefunktionsfehleranalyse B.6.6.4

Software FMEA C.6.2
Software-Gefährdungs- und Betreibbarkeitsuntersuchung (en: software hazard and operability study, CHAZOP, FMEA) C.6.2
Software-Gefährdungs- und Betreibbarkeitsuntersuchung (en: software hazard and operability study, CHAZOP, FMEA) C.6.2
Softwarekomplexitätskontrolle C.5.13
Software-Konfigurationsmanagement C.5.24
Spannungsüberwachung (sekundärseitig) A.8.2
Spannungsversorgung A.8
Sprachenteilmenge C.4.2
Strukturabhängige Tests C.5.8
Standardtestschnittstelle (Access Port) und Boundary-Scan-Architektur A.2.3
Statische Analyse B.6.4
Statische Laufzeitanalyse (STA) E.23
Statistisches Testen B.5.3
Statistisches Testen C.5.1
Stellglieder (Aktoren) A.13
Streng typisierte Programmiersprache C.4.1
Strukturdiagramme C.2.3
Strukturierte Beschreibungsmethodik E.3
Strukturierte Programmierung C.2.7
Strukturierte schematische Methoden C.2.1
Strukturierte Spezifikation B.2.1
Strukturierter Entwurf B.3.2
Symbolische Ausführung C.5.11
Synchronisation von primären Eingängen und Kontrolle von Metastabilitäten E.10

T

Temperatursensor A.10.1
Temporäre Logik C.2.4.7
Test der Störfestigkeit gegen Stoßspannungen B.6.2
Test durch Fehlereinbau B.6.10
Test unter Grenzbedingungen B.6.9
Testabdeckung der Verifikationsszenarien (Testbenches) E.13
Testmanagement und Automatisierungswerkzeuge C.4.7
Testmuster A.6.1
Tests durch redundante Hardware A.2.1
Tools and translators
Trennung der Sicherheitsfunktionen des E/E/PE-Systems von Nichtsicherheitsfunktionen B.1.3
Trennung elektrischer Energieleitungen von Informationsleitungen A.11.1

U

Überprüfung der Anforderungen und Beschränkungen des Herstellers E.26

Überprüfung des Layouts Versus Schematic (LVS) E.38
Überspannungsschutz mit Sicherheitsabschaltung A.8.1
Übertragungsredundanz A.7.5
Überwachte Ausgaben A.6.4
Überwachte Redundanz A.2.5
Überwachung von Relaiskontakten A.1.2
Überwachung A.13.1
UML C.3.12
Unterlastung A.2.8
Unveränderliche Speicherbereiche A.4
Ursache-Wirkungsdiagramme B.6.6.2

V

Validierung durch Regressionstest C.5.25
Validierung von Soft-Cores E.21
VDM, VDM++ – Wiener Entwicklungsmethode (en: Vienna Development Method) C.2.4.8
Verallgemeinertes stochastisches Petri-Netz (en: generalised stochastic petri net, GSPN) B.6.6.10
Veränderliche Speicherbereiche A.5
Veränderliche Speicherbereiche A.5
Verarbeitungseinheiten (CPUs) A.3
Vergleich der Gatter-Netzliste gegen ein Referenzmodell durch Simulation E.24
Vergleich der Gatter-Netzliste mit dem Referenzmodell (formale Äquivalenzprüfung) E.25
Vergleich des Source-Codes mit dem ausführbaren Code C.4.4.1
Vergleicher A.1.3
Verwendung bewährter/verifizierter Softwareelemente C.2.10
Verwendung eines Codecheckers E.15
Verwendung validierter Hard-Cores E.35
Verwendung von betriebsbewährten Schaltkreisfamilien E.41
Verwendung von betriebsbewährten Synthesewerkzeugen E.28
Verwendung von betriebsbewährten Zielbibliotheken E.29
Verwendung von bewährten Bauteilen B.3.3
(V)HDL-Simulation E.5
(V)HDL-Simulation E.5
Vollständige Hardwareredundanz A.7.3

W

Walk-through (Software) C.5.15
Walk-through E.19, B.3.8
Watchdog mit separater Zeitbasis ohne Zeitfenster A.9.1
Watchdog mit separater Zeitbasis und Zeitfenster A.9.2
Worst-Case-Analyse B.6.7

Wortsicherungsverfahren mit Mehr-Bit-Redundanz (zum Beispiel ROM-Überwachung mit einem modifizierten Hammingcode) A.4.1

Z

Z C.2.4.9
Zeitbehaftete Petri-Netze B.2.3.3
Zeitliche Überwachung mit Test während des Betriebs A.9.5
Zeitliche und logische Programmlaufüberwachung A.9
Zertifizierte Werkzeuge und Compiler C.4.3
Zusätzliche Reserven (> 20 %) für Prozesstechnologien mit weniger als 3 Jahren Anwendung E.39
Zuschaltung von Umluftkühlung und Statusanzeige A.10.5
Zustandsloser Softwareentwurf (oder Entwurf eingeschränkter Zustände) C.2.12
Zuverlässigkeitsblockdiagramm (en: reliability block diagrams, RBD) B.6.6.7
Zuverlässigkeitsblockdiagramm C.6.4
Zwangsöffnender Schalter A.12.2

Literaturhinweise

[1] IEC 60068-1:1988, *Environmental testing – Part 1: General and guidance*

ANMERKUNG Harmonisiert als EN 60068-1:1994 (nicht modifiziert).

[2] IEC 60529:1989, *Degrees of protection provided by enclosures (IP Code)*

ANMERKUNG Harmonisiert als EN 60529:1991 (nicht modifiziert).

[3] IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

ANMERKUNG Harmonisiert als EN 60812:2006 (nicht modifiziert).

[4] IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

ANMERKUNG Harmonisiert als EN 60880:2009 (nicht modifiziert).

[5] IEC 61000-4-1:2006, *Electromagnetic compatibility (EMC) – Part 4-1: Testing and measurement techniques – Overview of IEC 61000-4 series*

ANMERKUNG Harmonisiert als EN 61000-4-1:2007 (nicht modifiziert).

[6] IEC 61000-4-5:2005, *Electromagnetic compatibility (EMC) – Part 4-5: Testing and measurement techniques – Surge immunity test*

ANMERKUNG Harmonisiert als EN 61000-4-5:2006 (nicht modifiziert).

[7] IEC/TR 61000-5-2:1997, *Electromagnetic compatibility (EMC) – Part 5: Installation and mitigation guidelines – Section 2: Earthing and cabling*

[8] IEC 61025:2006, *Fault tree analysis (FTA)*

ANMERKUNG Harmonisiert als EN 61025:2007 (nicht modifiziert).

[9] IEC 61069-5:1994, *Industrial-process measurement and control – Evaluation of system properties for the purpose of system assessment – Part 5: Assessment of system dependability*

ANMERKUNG Harmonisiert als EN 61069-5:1995 (nicht modifiziert).

[10] IEC 61078:2006, *Analysis techniques for dependability – Reliability block diagram and boolean methods*

ANMERKUNG Harmonisiert als EN 61078:2006 (nicht modifiziert).

[11] IEC 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

ANMERKUNG Harmonisiert als EN 61131-3:2003 (nicht modifiziert).

[12] IEC 61160:2005, *Design review*

ANMERKUNG Harmonisiert als EN 61160:2005 (nicht modifiziert).

[13] IEC 61163-1:2006, *Reliability stress screening – Part 1: Repairable assemblies manufactured in lots*

ANMERKUNG Harmonisiert als EN 61163-1:2006 (nicht modifiziert).

[14] IEC 61164:2004, *Reliability growth – Statistical test and estimation methods*

ANMERKUNG Harmonisiert als EN 61164:2004 (nicht modifiziert).

[15] IEC 61165:2006, *Application of Markov techniques*

ANMERKUNG Harmonisiert als EN 61165:2006 (nicht modifiziert).

[16] IEC 61326-3-1:2008, *Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 3-1: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) – General industrial applications*

ANMERKUNG Harmonisiert als EN 61326-3-1:2008 (nicht modifiziert).

[17] IEC 61326-3-2:2008, *Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 3-2: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) – Industrial applications with specified electromagnetic environment*

ANMERKUNG Harmonisiert als EN 61326-3-2:2008 (nicht modifiziert).

[18] IEC 81346-1:2009, *Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations – Part 1: Basic rules*

ANMERKUNG Harmonisiert als EN 81346-1:2009 (nicht modifiziert).

[19] IEC 61506:1997, *Industrial-process measurement and control – Documentation of application software*

[20] IEC/TR 61508-0:2005, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 0: Functional safety and IEC 61508*

[21] IEC 61511 (alle Teile), *Functional safety – Safety instrumented systems for the process industry sector*

ANMERKUNG Harmonized in der Reihe EN 61511 (nicht modifiziert).

[22] IEC 62061:2005, *Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems*

ANMERKUNG Harmonisiert als EN 62061:2005 (nicht modifiziert).

[23] IEC 62308:2006, *Equipment reliability – Reliability assessment methods*

ANMERKUNG Harmonisiert als EN 62308:2006 (nicht modifiziert).

[24] ISO/IEC 1539-1:2004, *Information technology – Programming languages – Fortran – Part 1: Base language*

[25] ISO 5807:1985, *Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts*

[26] ISO/IEC 7185:1990, *Information technology – Programming languages – Pascal*

[27] ISO/IEC 8631:1989, *Information technology – Program constructs and conventions for their representation*

[28] ISO/IEC 8652:1995, *Information technology – Programming languages – Ada*

[29] ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*

[30] ISO/IEC 9899:1999, *Programming languages – C*

[31] ISO/IEC 10206:1991, *Information technology – Programming languages – Extended Pascal*

[32] ISO/IEC 10514-1:1996, *Information technology – Programming languages – Part 1: Modula-2, Base Language*

[33] ISO/IEC 10514-3:1998, *Information technology – Programming languages – Part 3: Object Oriented Modula-2*

[34] ISO/IEC 13817-1:1996, *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*

[35] ISO/IEC 14882:2003, *Programming languages – C++*

[36] ISO/IEC/TR 15942:2000, *Information technology – Programming languages – Guide for the use of the Ada programming language in high integrity systems*

[37] IEC 61800-5-2, *Electromagnetic compatibility (EMC) – Part 5: Installation and mitigation guidelines – Section 2: Earthing and cabling*

ANMERKUNG Harmonisiert als EN 61800-5-2.

[38] IEC 60601 (alle Teile), *Medical electrical equipment*

ANMERKUNG Harmonized in der Reihe EN 60601 (teilweise modifiziert).

[39] IEC 60068-2-1, *Environmental testing – Part 2-1: Tests – Test A: Cold*

ANMERKUNG Harmonisiert als EN 60068-2-1.

[40] IEC 60068-2-2, *Environmental testing – Part 2-2: Tests – Test B: Dry heat*

ANMERKUNG Harmonisiert als EN 60068-2-2.

[41] ISO 9000, *Quality management systems – Fundamentals and vocabulary*

ANMERKUNG Harmonisiert als EN ISO 9000.

[42] IEC 61508-1:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements*

ANMERKUNG Harmonisiert als EN 61508-1:2010 (nicht modifiziert).

[43] IEC 61508-2:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems*

ANMERKUNG Harmonisiert als EN 61508-2:2010 (nicht modifiziert).

[44] IEC 61508-3:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements*

ANMERKUNG Harmonisiert als EN 61508-3:2010 (nicht modifiziert).

[45] IEC 61508-6:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3*

ANMERKUNG Harmonisiert als EN 61508-6:2010 (nicht modifiziert).

Anhang ZA
(normativ)

Normative Verweisungen auf internationale Publikationen mit ihren entsprechenden europäischen Publikationen

Die folgenden zitierten Dokumente sind für die Anwendung dieses Dokuments erforderlich. Bei datierten Verweisungen gilt nur die in Bezug genommene Ausgabe. Bei undatierten Verweisungen gilt die letzte Ausgabe des in Bezug genommenen Dokuments (einschließlich aller Änderungen).

ANMERKUNG Wenn internationale Publikationen durch gemeinsame Abänderungen geändert wurden, durch (mod) angegeben, gelten die entsprechenden EN/HD.

Publikation	Jahr	Titel	EN/HD	Jahr
IEC 61508-4	2010	Functional safety of electrical/electronic/ programmable electronic safety-related systems – Part 4: Definitions and abbreviations	EN 61508-4	2010