
Entwicklung von Echtzeitsystemen

Hubert B. Keller

Entwicklung von Echtzeitsystemen

Einführung in die Entwicklung
zuverlässiger softwarebasierter Funktionen
unter Echtzeitbedingungen

Hubert B. Keller
Institut für Automation und angewandte Informatik (IAI)
Karlsruher Institut für Technologie (KIT)
Karlsruhe, Deutschland

ISBN 978-3-658-26640-0 ISBN 978-3-658-26641-7 (eBook)
<https://doi.org/10.1007/978-3-658-26641-7>

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Springer Vieweg

© Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2019

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von allgemein beschreibenden Bezeichnungen, Marken, Unternehmensnamen etc. in diesem Werk bedeutet nicht, dass diese frei durch jedermann benutzt werden dürfen. Die Berechtigung zur Benutzung unterliegt, auch ohne gesonderten Hinweis hierzu, den Regeln des Markenrechts. Die Rechte des jeweiligen Zeicheninhabers sind zu beachten.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag, noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.

Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Vorwort

Die Automatisierung technischer Systeme ist das Rückgrat unserer modernen Gesellschaft. Softwarebasierte Funktionen realisieren dabei in allen Bereichen wesentliche Innovationen auf Basis zuverlässiger Technologien der Informationsverarbeitung unter Echtzeitbedingungen.

Die Anwendungen reichen dabei von verfahrenstechnischen Anlagen, verteilten kritischen Infrastrukturen wie Energieverteilungssystemen, der Wasserversorgung und Verkehrsleitanlagen über Home-Automation-Systeme, intelligente Staubsauger und Fuzzy-Control-gesteuerte Waschmaschinen, autonome Transportsysteme, mechatronische Systeme wie ABS- und ESP-Systeme sowie bildbasierte und interpretative Kollisions- und Fußgängererkennung im Automobil, Herzschrittmacher, Infusionspumpen und Röntgengeräte bis zu den zukünftigen vollständig autonomen mobilen Systemen aller Art.

Der Grad an Automatisierung in unserer Gesellschaft lässt sich daran bemessen, dass weit über 90 % aller Rechner in eingebetteten Anwendungen eingesetzt sind (mittlerweile wird von 98 % aller Rechner als eingebettet ausgegangen [Aca2011]). Alle Disziplinen vertrauen auf softwarebasierte Funktionen unter Echtzeitbedingungen. Der durch Software realisierte Anteil beträgt mittlerweile fast die Hälfte an der Wertschöpfung in den technischen Bereichen. Nach Isermann [Ise2008], Seite V, sind die Entwicklungen bei mechatronischen Systemen durch eine intensive Informationsverarbeitung charakterisiert: „Die funktionelle Integration wird jedoch entscheidend durch die Informationsverarbeitung und damit durch die Gestaltung der Software geprägt.“

Die Herstellung von Software ist im Allgemeinen und insbesondere bei Echtzeitsoftware, wie sie in der Automatisierung zur Anwendung kommt, noch immer fehlerbehaftet und mit hohen Kosten verbunden. Ein echter Mehrwert durch Software kann dann erreicht werden, wenn die Funktionen zuverlässig, fehlerbeherrschbar und bei offenen und vernetzten Systemen informationstechnisch sicher sind. Die zeitlichen Eigenschaften sind beweisbar zu erbringen und die Herstellung muss ökonomisch vertretbar sein. Eine erhebliche Anzahl an Fehlern wird in frühen Phasen der Entwicklung injiziert und mit hohen Kosten erst in späteren Phasen erkannt und behoben. Hier muss die Analyse als Teil des gesamten Entwicklungsprozesses konkreter angegangen werden. Dazu bedarf es entspre-

chender analyse- und teamorientierter Beschreibungsverfahren. Der Rational Unified Process (RUP) als iterativer, inkrementeller, objektorientierter und architekturzentrierter Prozess liefert mit der Unified Modeling Language (UML) eine entsprechende grafische Methode. Die UML bietet als teamorientierter Zugang ein wertvolles Hilfsmittel. Ausgehend von Anwendungsfällen (Use Cases) können für nebenläufige Strukturen, wie sie in der Automatisierung grundsätzlich auftreten, notwendige Analysen und Beschreibungen erstellt werden. Auch die Optimierung von Entwicklungsprozessen ist nach den Vorgaben des Capability Maturity Model Integrated (CMMI) nachhaltig möglich, wie Ergebnisse abgeschlossener Projekte zeigen.

Die Automatisierung von Anlagen und Maschinen sowie kritischen Infrastrukturen etc. bedeutet das Ausführen von softwarebasierten Funktionen unter Echtzeitbedingungen. Die Ausführung hat deterministisch und beweisbar zu erfolgen. Sicherheitskritische Anwendungen müssen die entsprechenden Zielfunktionen verlässlich und rechtzeitig erbringen. Hierzu sind verschiedene Arten der rechtzeitigen und gleichzeitigen Behandlung von realen Ereignissen möglich. Das prozessorientierte Konzept mit virtuellen Recheneinheiten (CPU – Central Processing Unit) und der prioritätenbasierten Abbildung auf reale CPUs mit preemptivem Scheduling (verdrängende Ausführung) erlaubt die Trennung der Ausführung und der Verwaltung von Funktionen bzw. Prozessen. Die Trennung von Ausführung und Verwaltung mit der gegenseitigen Isolierung der Prozessausführung ist ein zentraler Aspekt für die konstruktive Sicherstellung der Zuverlässigkeit von Echtzeitsoftware.

Sowohl bei Model Driven Architecture (MDA) zur automatischen Codegenerierung als auch bei der manuellen Programmierung steht eine Programmiersprache als Ziel den Modellbeschreibungen gegenüber. Die Programmiersprache muss adäquate Sprachelemente zur Abbildung nebenläufiger Prozesse und alle weiteren Modellierungsaspekte, wie typstrenge Datenstrukturen, Module mit „information hiding“ (Verbergen der Implementierung) als Architekturbausteine sowie sichere Kontrollstrukturen besitzen. Zusätzlich ist eine Laufzeitumgebung notwendig, die Fehler im Ablauf erkennt und es erlaubt, sinnvoll darauf zu reagieren. Die Programmiersprache Ada erfüllt hierzu alle wesentlichen Anforderungen.

Safety-Anforderungen sind bei offenen und vernetzten Systemen und Anlagen des Maschinenbaus untrennbar mit Security-Anforderungen verknüpft. Hierzu müssen Architekturen und Konzepte definiert werden, um sicherheitskritische Risiken zu beherrschen. Sender und Empfänger von Daten sind zu identifizieren, Transportwege zu sichern und auch systemfremde Software beherrschbar einzubinden. Viele Probleme aus dem Bereich der Informationssicherheit (Cyber-Angriffe) können in der Implementierung durch eine Sprache wie Ada vermieden werden. Dies ist durch den Vergleich der Indizierungskonzepte von C/C++ und Ada ersichtlich. Die Manipulation und Veränderung von Daten in Programmanweisungen durch Überschreiten von Indexgrenzen oder beim Aufrufkontext („call frame“) von Prozeduren wird durch das Typkonzept von Ada und die Prüfung zur Laufzeit verhindert. Hier ist ein Umdenken in der Ausbildung im Bereich der Programmierung und des Softwareengineerings erforderlich, damit „Buffer Overflow“

oder „handcrafted commands and parameter“ als Angriffsvektoren im Cyber Security nicht mehr existent sind.

Automatisierung ist umfassend zu verstehen. Die Entwicklung softwarebasierter Funktionen unter Echtzeitbedingungen in der Automatisierung wird hier auf eine systematische Basis gestellt. Es werden die notwendigen Anforderungen sowie deren Erfüllbarkeit analysiert und eine integrative Darstellung und Bewertung der notwendigen Randbedingungen und Methoden zur Realisierung von zuverlässigen softwarebasierten Funktionen unter Echtzeitbedingungen zur Automatisierung vorgenommen.

Arbeiten des VDI (Verein Deutscher Ingenieure e.V.) Präsidiumsarbeitskreises „Technische Sicherheit“, in denen der Autor maßgeblich involviert war, haben gezeigt, dass der Begriff und das Konzept der „funktionalen Sicherheit“ (Safety) in den verschiedenen Anwendungsbereichen sehr unterschiedlich ausgelegt und umgesetzt wird. Der Arbeitskreis hat über viele Jahre das große Gemeinsame herausgearbeitet und eine disziplinübergreifende Sichtweise, Systematik und Methodik abgeleitet und definiert. Das Ergebnis ist mittlerweile als englischsprachiges Buch im Springer-Verlag publiziert.

Auch eine integrative Sicht auf Security-Anforderungen in der Automatisierung wurde unter Mitwirkung des Autors im Rahmen des Lenkungsreises Security des ZVEI erarbeitet. Das Ergebnis wurde als Namur-Empfehlung NE 153 publiziert (Namur – Interessengemeinschaft Automatisierungstechnik der Prozessindustrie) und deckt von der Anforderungsanalyse bis zur Wartung im Betrieb alle Aspekte von Security in der Automatisierung ab.

Der Autor war in der Entwicklung von Prozessrechnersystemen mit redundanter Funktionalität für hoch sicherheitskritische Anwendungen bei der Siemens AG tätig. Danach entwickelte er am Kernforschungszentrum Karlsruhe umfangreiche verteilte Echtzeitsysteme mit der Sprache Ada. Diese war nach softwaretechnischen Kriterien ausgewählt worden. Die Verwendung von Tasks mit entsprechenden Kommunikationsschnittstellen in Ada ist eine Methode, um die Komplexität im Ablaufverhalten zu beherrschen. Weitere Systeme zur Automatisierung komplexer industrieller Prozesse haben auch sprachfremde Software eingebunden. Hier hat sich dann die Eigenschaft des Ada-Laufzeitsystems gezeigt, Fehler synchron zur Programmausführung grundsätzlich zu erkennen, zu melden und eine Reaktion darauf zu ermöglichen. Der weitere Weg war der Einsatz von Model-Driven-Architecture-Ansätzen für hochzuverlässige und gleichzeitig zu 70 % automatisch generierte Systeme. Dabei erfolgte die Integration von Sicherheitsprotokollen für die Kommunikation verteilter Systemkomponenten. Im Rahmen von Kooperationen mit der Industrie ist z. B. das INSPECT-pro-control-System mit sehr hoher Zuverlässigkeit, geringsten Fehlerraten und erheblich geringerem Budget als vergleichbare Entwicklungen realisiert worden. Das INSPECT-pro-control-System ist ein generisches Werkzeug, das mittlerweile weltweit und nahezu fehlerfrei in Anlagen der Industrie unter Echtzeitbedingungen eingesetzt wird. Anwendungsbereiche sind z. B. thermische Abfallbehandlung, Zinkrecycling, Zementindustrie, verteilte Sensorsysteme und Sonderabfallbehandlung.

Die vielfältigen Anforderungen der wissenschaftlichen Tätigkeit im Bereich der hochinterdisziplinären Technologieentwicklung am Karlsruher Institut für Technologie

(KIT) und dessen Vorläufer (Kernforschungszentrum und später Forschungszentrum) und einer gleichzeitigen erfolgreichen industriellen Umsetzung sowie die Vermittlung des gesamten Methodenspektrums in der Lehre waren und sind eine hochinteressante Herausforderung. Das angestrebte Ziel war immer, innovative, leistungsfähige und von der Industrie akzeptierte Systeme zur Automatisierung komplexer industrieller Prozesse unter Echtzeitbedingungen zu realisieren. Das ingenieurwissenschaftliche Vorgehen verbindet hierbei die Methoden des Ingenieurwesens und den hohen Zuverlässigkeitsgrad von Systemen mit den Methoden der Informatik. Mein Dank geht sowohl an meine Mitarbeiter für die fruchtbare und hervorragende Zusammenarbeit als auch an die übergeordneten Führungsebenen für die kreativen und konstruktiven Freiräume. Dadurch konnten schon früh wissenschaftlich innovative Entwicklungen der Ingenieurwissenschaften und der Informatik sich gegenseitig befruchtend prototypisch angegangen und nachfolgend industriell erfolgreich umgesetzt werden.

Diese vielfältigen und herausfordernden Forschungs- und Entwicklungsthemen haben dabei zeitlich durchweg auch in den privaten Bereich hineingewirkt. Daher danke ich ganz besonders meiner Familie für das Verständnis für die Arbeit eines immer intensiv beschäftigten Wissenschaftlers.

Karlsruhe, 2019

Hubert B. Keller

Literatur

- [Aca2011] Cyber-Physical Systems. Acatech Position. Deutsche Akademie der Wissenschaften, Berlin (2011), Seite 5.
- [Ise2008] Isermann, R.: Mechatronische Systeme – Grundlagen. Springer, Heidelberg (2008). ISBN: 978-3-540-32512-3

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Entwicklungsstand	3
1.3	Ziele, Gliederung und Basisliteratur	17
	Literatur	21
2	Herstellungsprozesse für Software	27
2.1	Allgemeine Vorgehensmodelle	30
2.2	Primitivmodell	30
2.3	Wasserfallmodell	31
2.4	Rapid Prototyping	32
2.5	Evolutionäre Softwareentwicklung	32
2.6	Spiralmodell	33
2.7	Iteratives Modell mit parallelen Phasen	34
2.8	Extremes Programmieren (Extreme Programming, XP)	34
2.9	Cleanroom Software Engineering	38
2.10	V-Modell und V-Modell XT als Prozessmodelle	39
2.11	Rational Unified Process (RUP)	42
2.12	Softwarequalität und Prozessreife	47
2.13	ISO 9000	48
2.14	Capability Maturity Model Integrated (CMMI)	48
2.15	Bewertung der Herstellungsprozesse für softwarebasierte Funktionen	48
	Literatur	51
3	Analyse und Bewertung von Konzepten zur Ereignisbehandlung unter Echtzeitbedingungen	53
3.1	Grundlegende Eigenschaften von Echtzeitsystemen	53
3.2	Ereignisbehandlung in Echtzeitsystemen	57
3.3	Analyse der Lösungsansätze zur Ereignisbehandlung unter Echtzeitbedingungen	65

3.3.1	Polling als Konzept	65
3.3.2	Zeitgesteuertes Konzept	66
3.3.3	Interruptbasiertes Konzept	67
3.3.4	Cyclic-Executive-Konzept oder SPS-Programmierung	70
3.3.5	Co-Routinen-Konzept	72
3.4	Bewertung der Konzepte zur Ereignisbehandlung unter Echtzeitbedingungen	73
	Literatur	74
4	Prozesskonzept als Basis für Echtzeit und Zuverlässigkeit	77
4.1	Historie des Prozesskonzeptes	77
4.1.1	Prozesskonzept – Historie Großrechner	77
4.1.2	Prozesskonzept – Historie Prozessrechner	78
4.2	Virtuelle CPU – vCPU	79
4.2.1	Prozesszustände und Warteschlangen	82
4.2.2	Kooperation und Nebenläufigkeit	85
4.2.3	Datenkonsistenz bei kooperierenden Prozessen	91
4.2.4	Bewertung der Verfahren zur Zugriffssteuerung	104
4.3	Verwaltung von Prozessen	105
4.4	Scheduling	111
4.4.1	Rate Monotonic Scheduling (RMA/RMS)	113
4.4.2	Earliest Deadline First Scheduling (EDF)	116
4.4.3	Least Laxity First Scheduling (LLF)	117
4.4.4	Response Time Analysis	118
4.4.5	Worst-Case-Execution-Time-Analyse – WCETA	120
4.4.6	Analyse der Schedulingverfahren	120
4.5	Scheduling mit Zugriffsprotokollen	130
4.5.1	Priority Inheritance Protocol (PIP)	132
4.5.2	Priority Ceiling Protocol (PCP)	134
4.6	Bewertung der Schedulingverfahren und Synchronisationsprotokolle	140
	Literatur	142
5	Programmierung von Echtzeitsystemen mit hoher Zuverlässigkeit	145
5.1	Einführung	145
5.2	Einführung in die Sprache Ada	151
5.3	Grundlegende Zuverlässigkeit in der Typsicherheit und im Algorithmus	154
5.3.1	Beispiele von Typdefinitionen	155
5.3.2	Sequenzielle Kontrollstrukturen	159
5.3.3	Modularisierung als Architektureigenschaft	162

5.3.4	Monitorkonzept und Scheduling bei Ada	165
5.3.5	Tasking und Kommunikation	166
5.3.6	Exception Handling	170
5.4	Studentisches Beispiel zur Umsetzung von UML-Modellen nach Ada mit Model Driven Architecture	171
5.5	Formale Spezifikation mit Ada 2012	178
5.6	Bewertung der Sprachanforderungen für zuverlässige Echtzeitsysteme	181
	Literatur	183
6	Integrative Betrachtung von technischer Sicherheit und Informationssicherheit	185
6.1	Technische Sicherheit	186
6.1.1	Risikobegriff	188
6.1.2	Sicherheitsnormen am Beispiel der IEC 61508	190
6.2	Informationssicherheit (IT-Security)	195
6.2.1	Begriffe	197
6.2.2	Cyber-Security-Schwachstellen und Angriffsmöglichkeiten	199
6.2.3	Kritikalität von Cyber-Security-Schwachstellen	203
6.2.4	Methoden zur Absicherung gegen Cyber-Security-Angriffe	208
6.3	Konzepte zur integrativen Betrachtung von Safety und Security zur Sicherstellung der Zuverlässigkeit in der Automatisierung	213
6.3.1	Klassifikation und Bewertung der Cyber-Sicherheitskritikalität	219
6.3.2	Ursachen von Implementierungsschwachstellen	224
6.3.3	Resümee	226
	Literatur	227
7	Beispielhafte Umsetzung der Entwicklungsmethodik	231
7.1	Ein gassensitives Sensorsystem mit sicherheitskritischem Profil	231
7.1.1	Beschreibung des Sensorsystems	231
7.1.2	Softwareentwicklung mit Ada und dem Ravenscar-Profil	233
7.1.3	Zusammenfassung	237
7.2	Das INSPECT-pro-control-System zur Optimierung thermischer Prozesse	238
7.2.1	Inspect-Architektur	240
7.2.2	Der Model-Driven-Architecture-Ansatz	242
7.2.3	UML-Modell	243
7.3	Zusammenfassung	247
	Literatur	248

8	Empfehlungen für die Entwicklung von zuverlässiger	
	Automatisierungssoftware	251
	Literatur	259
9	Zusammenfassung	261
	Literatur	265
	Anhang	267
	Literatur	283
	Stichwortverzeichnis	285

Abbildungsverzeichnis

Abb. 1.1	Verfahrenstechnische Anlage [Zin2006]	2
Abb. 1.2	Entwicklung ABS-Systeme [Med2008]	3
Abb. 1.3	Entwicklung im Infotainmentsystem von Automobilen [ZVEI2016]	4
Abb. 1.4	Umfang der softwarebasierten Funktionalität im Automobil [Con2005]	5
Abb. 1.5	Kontinuumsprinzip	7
Abb. 1.6	Wertschöpfungsanteil von Software [Ros2003]	9
Abb. 1.7	Injektionszeitpunkt von Fehler und Kosten der Behebung [Cros2005]	10
Abb. 1.8	Fehlervermeidung durch den PSP im CMMI [Gir2012]	13
Abb. 1.9	Zyklomatische Komplexität	14
Abb. 1.10	Fehlerrisiken gegenüber Anweisungen und zyklomatischer Komplexität [Ram2004]	14
Abb. 1.11	Sicherheitsziele von außen und Implementierungsschwachstellen	15
Abb. 2.1	Life Cycle in der Softwareentwicklung [Pom1984]	28
Abb. 2.2	Qualitätseigenschaften von Software (DIN66272)	29
Abb. 2.3	Klassisches Wasserfallmodell [Sum2007]	31
Abb. 2.4	Evolutionäres Modell [Sum2007]	32
Abb. 2.5	Spiralmodell [Boe1988]	33
Abb. 2.6	XP-Praktiken [Bec2000]	35
Abb. 2.7	V-Modell [VM1997]	39
Abb. 2.8	Bausteine des V-Modells [VM1997]	40
Abb. 2.9	Produkt und Rolle [VXT2004]	42
Abb. 2.10	Entwicklung von UML [UML1999]	43
Abb. 2.11	Elemente von UML [UML2002]	44
Abb. 2.12	Aktivitätsdiagramm nebenläufiger Funktionalität	45
Abb. 2.13	Prozessdefinition im RUP [Rup1998]	46
Abb. 2.14	Stufen im CMM-Modell [SEI2002]	49
Abb. 2.15	Vergleich der Vorgehens-/Prozessmodelle und RUP als zentraler Ansatz	50
Abb. 3.1	ABS-Beispiel	54
Abb. 3.2	Zyklus	54

Abb. 3.3	Airbagsystem	55
Abb. 3.4	Reaktionsintervall	55
Abb. 3.5	Struktur Echtzeitverarbeitung	58
Abb. 3.6	Vernetzung im Automobil [Med2008]	59
Abb. 3.7	Zeit-Aktivitäts-Diagramm	61
Abb. 3.8	Periodische Prozesse mit Zykluszeit T_P und Rechenzeit T_C	62
Abb. 3.9	Skala zeitlicher Bereiche	63
Abb. 3.10	Ereigniserzeugung	63
Abb. 3.11	Ereignisse E und zugehörige Parameter	64
Abb. 3.12	Abbildung der Ausführung auf kleinstem Zyklus	66
Abb. 3.13	Verletzung der Zeitanforderung	68
Abb. 3.14	Ausführung nach steigenden Wiederholungsraten	69
Abb. 3.15	Funktionszerlegung und Abbildung auf Minimalzyklus	71
Abb. 3.16	Co-Routinen-Konzept	73
Abb. 4.1	Virtuelle und reale CPU	80
Abb. 4.2	Instanzen und Aufrufkontexte	81
Abb. 4.3	Zustände und Warteschlangenverwaltung von Prozessen	83
Abb. 4.4	Kontextwechsel	84
Abb. 4.5	Einordnung nach Prioritäten	84
Abb. 4.6	Wartezustände (Ereignis)	85
Abb. 4.7	Datenbasierte Kommunikation	86
Abb. 4.8	Nachrichten- und datenbasierte Kommunikation	86
Abb. 4.9	Datenfluss	88
Abb. 4.10	Mögliche Berechnungspfade	89
Abb. 4.11	Abbildung der vCPU auf die reale CPU	89
Abb. 4.12	Schwer- und leichtgewichtige Prozesse [Sta2003]	90
Abb. 4.13	Konkurrierender Zugriff auf gemeinsame Daten	92
Abb. 4.14	Unterbrechbare Befehlsfolge	93
Abb. 4.15	Kritischer Abschnitt („critical section“)	93
Abb. 4.16	Beispielszenario	94
Abb. 4.17	Ablauf im Algorithmus von Dekker (vgl. [Heho1989])	95
Abb. 4.18	Algorithmus nach Peterson [Pet1981]	95
Abb. 4.19	Atomarer Test [Sta2003]	97
Abb. 4.20	P- und V-Operationen (vgl. [Heho1989])	98
Abb. 4.21	Problem mit Semaphore zur Reihenfolgesynchronisation	98
Abb. 4.22	Zugriffsteuerung und Reihenfolgesynchronisation [Ben2009]	99
Abb. 4.23	Überkreuzzugriffe	99
Abb. 4.24	Lokalität statt Verteilung	100
Abb. 4.25	Read- und Write-Operationen [Ben2009]	101
Abb. 4.26	Monitorkonzept	101
Abb. 4.27	Synchronisation mit Signalen [Ben2009]	102
Abb. 4.28	Rendezvous-Konzept von Ada	103
Abb. 4.29	Rendezvous-Beispiel	104

Abb. 4.30	Bewachte Eingänge	104
Abb. 4.31	Verwaltungsfunktion	106
Abb. 4.32	Windows-2000-Architektur [Win2000]	109
Abb. 4.33	Microkernel-Architektur [Sta2003]	109
Abb. 4.34	Prozesszustände	110
Abb. 4.35	Ereigniswarteschlangen	111
Abb. 4.36	Prozesszustände	112
Abb. 4.37	Zeitgrößen eins Rechenprozesses	112
Abb. 4.38	Phasenlage zum Zeitpunkt Null bei Rate Monotonic Analysis (RMA)	114
Abb. 4.39	Phasenlagen und Ausführbarkeit	114
Abb. 4.40	Rate Monotonic Analysis mit Erweiterungen	115
Abb. 4.41	Rate Monotonic Scheduling (RMS) am Beispiel	115
Abb. 4.42	Earliest Deadline First (EDF) am Beispiel	116
Abb. 4.43	Least Laxity First (LLF) am Beispiel	117
Abb. 4.44	Wechselwirkung zweier Prozesse	118
Abb. 4.45	Prozessbeispiel	121
Abb. 4.46	Zeichnerische Lösung von RMS	123
Abb. 4.47	Zeitgrößen für EDF (Restantwortzeit RAZ)	124
Abb. 4.48	Least-Laxity-First-Beispiel	130
Abb. 4.49	Zugriff auf gemeinsame Daten	131
Abb. 4.50	Prioritätsinversion	131
Abb. 4.51	Unbounded Blocking	132
Abb. 4.52	Prioritätsvererbung	133
Abb. 4.53	Deadlock	133
Abb. 4.54	Ceiling-Priority-System	134
Abb. 4.55	Ressourcenzugriffe	135
Abb. 4.56	Ceiling-Prioritäten der Ressourcen	135
Abb. 4.57	Zusammenspiel im Beispiel bei PCP	136
Abb. 4.58	Beispiel mit Testnachweis	137
Abb. 4.59	Beispiel nach Optimierung	138
Abb. 4.60	Beispiel Taskset und Ressourcenverbund	139
Abb. 5.1	Syntaktischer Fehler in der IF-Anweisung	147
Abb. 5.2	Syntaktisch fehlerhafter logischer Vergleich	148
Abb. 5.3	Syntaktische Struktur der If-Anweisung von Ada	148
Abb. 5.4	Konvertierung in C	148
Abb. 5.5	Gleitpunktzahl als Schleifenvariable	148
Abb. 5.6	Inkonsistente Repräsentation von Aufzählungswerten	149
Abb. 5.7	Regel zur Erhöhung der Lesbarkeit [Loc2005]	150
Abb. 5.8	Syntaktische Struktur der IF-Anweisung in Ada	150
Abb. 5.9	Ergebnis statische Analyse [Gan2012]	151
Abb. 5.10	Einbettung von Ada	153
Abb. 5.11	Typsystem in Ada [Ada2012]	161
Abb. 5.12	Auslagerung zur Komplexitätsbeherrschung	163

Abb. 5.13	Benutzt-Beziehung	163
Abb. 5.14	Modul mit privatem Typ und verborgener Implementierung	164
Abb. 5.15	Hierarchisch-modulare Architektur in Ada	165
Abb. 5.16	Beispiel eines „protected type“ [Ben2009]	166
Abb. 5.17	Zeitlich begrenztes Warten auf einen Aufruf	168
Abb. 5.18	Selektive Aufrufannahme mit zeitlich begrenztem Warten	169
Abb. 5.19	Beispiel Use Case	172
Abb. 5.20	Sequenzdiagramm Tee	173
Abb. 5.21	Klassendiagramm	173
Abb. 5.22	Stereotype Task	174
Abb. 5.23	Vererbungshierarchie	174
Abb. 5.24	Zustandsautomat	175
Abb. 5.25	Transformationsvorgang	176
Abb. 5.26	Typdefinition für Ereignisannahme	176
Abb. 5.27	Benutzerdefinierte Programmbereiche	177
Abb. 5.28	Task-Zustände [Mat2013]	177
Abb. 5.29	Task-Entries	177
Abb. 5.30	Ablaufverhalten der Task	178
Abb. 5.31	Spezifikation einer statischen Typinvariante	179
Abb. 5.32	Spezifikation einer dynamischen Typinvariante	180
Abb. 5.33	Spezifikation von Default-Werten	180
Abb. 5.34	Spezifikation eines Stack-Typs	180
Abb. 5.35	Spezifikation zur Überprüfung der Wirkung von Push mit „Old“	180
Abb. 6.1	Safety und IT-Security als zweiseitige Sicherheitsanforderung	186
Abb. 6.2	Risikobegriff [VDI2010]	189
Abb. 6.3	Redundanz und Funktionsdegradierung	189
Abb. 6.4	Safety Life Cycle [IEC2010]	191
Abb. 6.5	Safety Integrity Levels (nach [IEC1998])	192
Abb. 6.6	Vorgehen und Entscheidungslogik zur Erzeugung von Sicherheit (Safety), vgl. [Kel2018])	194
Abb. 6.7	Herausforderungen nach dem ZVEI [ZVEI2006]	196
Abb. 6.8	Funktionsebenen in der Security (vgl. auch [BSI2012])	200
Abb. 6.9	Anzahl der Produkte in industriellen Bereichen mit Schwachstellen im Jahr 2017 („Number of vulnerable products used in different industries“, [Kas2018])	202
Abb. 6.10	Angriffspunkte bei Windkraftanlagen nach [Rei2018]	203
Abb. 6.11	Verteilung der festgestellten Typen von Schwachstellen bei ICS-Systemen im Jahr 2017 („Most common vulnerability types“, [Kas2018])	204
Abb. 6.12	Zeitliche Entwicklung der Schwachstellen „Execute Code“, „Buffer Overflow“ und „Memory Corruption“ bei ICS-Systemen [CVE2018]	205
Abb. 6.13	Zeitliche Entwicklung der entdeckten Schwachstellen bei ICS-Systemen [CVE2018]	205

Abb. 6.14	Schwachstellen bei SCADA-Systemen [NST2010]	206
Abb. 6.15	Beispielhaftes IT-Security-Konzept bei SCADA-Systemen [Sie2018]	207
Abb. 6.16	Ebenen in der semantischen Absicherung gegen Cyber-Angriffe im Smart Energy Grid (vgl. [Hag2016])	208
Abb. 6.17	Diversität zur Manipulationsentdeckung [Ghe2011]	214
Abb. 6.18	Zertifizierte und offene Software	215
Abb. 6.19	Rückwirkungsfreie Schnittstelle über Stellvertreter	216
Abb. 6.20	Best Practices für Safety/Security [Ibr2004]	217
Abb. 6.21	Mathematische Definition einer Funktion mit definierter Eingabe- und Ausgabemenge	224
Abb. 6.22	Formale Definition einer Funktion entsprechend definierter Eingabemenge	224
Abb. 6.23	Nichtausschließliche Funktionsrealisierung mit zusätzlichen Freiheitsgraden	224
Abb. 6.24	Zeitliche Entwicklung der höchsten Kritikalität (grün) von Schwachstellen [NIST2018]	225
Abb. 6.25	Zeitliche Entwicklung von Schwachstellen im Detail [NIST2018]	226
Abb. 7.1	Beispielhafte LZP und Temperaturverlauf	232
Abb. 7.2	Vergleich gemessene/theoretische CTP eines binären Toluol-Ethanol-Gemisches	233
Abb. 7.3	Vergleich gemessene/theoretische CTP eines Nicht-Toluol-Ethanol-Gemisches	233
Abb. 7.4	Sensor als Hubsystem	234
Abb. 7.5	Tasks des Sensorsystems mit Zeitangaben	235
Abb. 7.6	Hubmagnetensteuerung	237
Abb. 7.7	Tasks des Sensorsystems mit Zeitangaben	238
Abb. 7.8	Sensorsystem mit Hubsensor und Rechnerplatine	238
Abb. 7.9	INSPECT-pro-control-System [Ker2001]	239
Abb. 7.10	Inspect-Komponenten	240
Abb. 7.11	Inspect-Benutzerschnittstellen	242
Abb. 7.12	Generierung von PSM aus PIM durch Transformation	243
Abb. 7.13	Profil für den MDA-Ansatz beim INSPECT-pro-control-System [Ker2001]	244
Abb. 7.14	Modellierter Zustandsautomat [Ker2001]	246
Abb. 7.15	Generiertes Schema für eine nebenläufige Task [Ker2001]	246
Abb. 7.16	Generierter Quelltext für Annahme von Entry-Aufrufen [Ker2001]	247

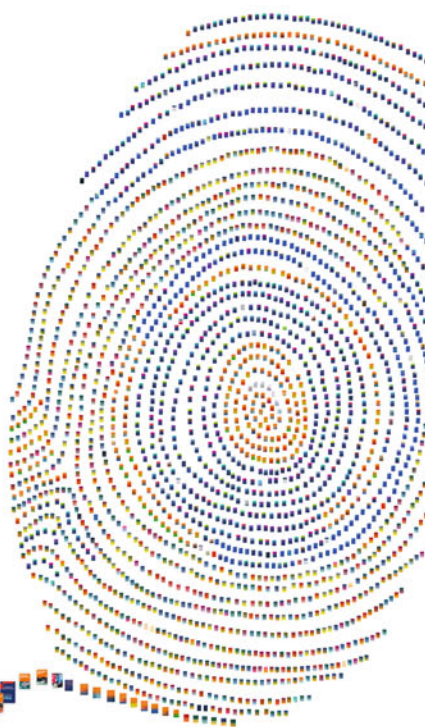
Tabellenverzeichnis

Tab. 1.1	Vergleich der Softwareumfänge und Fehlerraten in technischen Systemen (nach Tab. A.1)	4
Tab. 1.2	Erfolg bei Softwareprojekten [Stan2013]	6
Tab. 1.3	Kostenanteil von Elektronik [FAST2005]	10
Tab. 1.4	Typische Fehlerverteilung im Entwicklungsprozess [Linh2013]	12
Tab. 3.1	Funktionsbeispiel	67
Tab. 4.1	Beispielprozesse für LLF	128
Tab. 4.2	Laxity-Berechnung und LLF-Ausführung	129
Tab. 4.3	Vergleich EDF und RMA/RMS	142
Tab. 6.1	Expositionseinordnung	222
Tab. 6.2	Klassenfestlegung	223
Tab. 7.1	Tasks und geforderte Eigenschaften	236

Lizenz zum Wissen.

Sichern Sie sich umfassendes Technikwissen mit Sofortzugriff auf tausende Fachbücher und Fachzeitschriften aus den Bereichen: Automobiltechnik, Maschinenbau, Energie + Umwelt, E-Technik, Informatik + IT und Bauwesen.




Exklusiv für Leser von Springer-Fachbüchern: Testen Sie Springer für Professionals 30 Tage unverbindlich. Nutzen Sie dazu im Bestellverlauf Ihren persönlichen Aktionscode **C0005406** auf www.springerprofessional.de/buchaktion/



Jetzt
30 Tage
testen!

Springer für Professionals.

Digitale Fachbibliothek. Themen-Scout. Knowledge-Manager.

-  Zugriff auf tausende von Fachbüchern und Fachzeitschriften
-  Selektion, Komprimierung und Verknüpfung relevanter Themen durch Fachredaktionen
-  Tools zur persönlichen Wissensorganisation und Vernetzung

www.entschieden-intelligenter.de

Springer für Professionals

 **Springer**