

DIN EN 62628

ICS 29.020; 03.120.10; 35.080

**Leitlinien zu Softwareaspekten der Zuverlässigkeit
(IEC 62628:2012);
Deutsche Fassung EN 62628:2012**

Guidance on software aspects of dependability
(IEC 62628:2012);
German version EN 62628:2012

Lignes directrices concernant la sûreté de fonctionnement du logiciel
(CEI 62628:2012).
Version allemande EN 62628:2012

Gesamtumfang 67 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2012-09-12 angenommene Europäische Norm als DIN-Norm ist 2013-06-01.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN IEC 62628:2010-03.

Für dieses Dokument ist das nationale Arbeitsgremium K 132 „Zuverlässigkeit“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (www.dke.de) zuständig.

Die enthaltene IEC-Publikation wurde vom TC 56 „Dependability“ erarbeitet.

Das IEC-Komitee hat entschieden, dass der Inhalt dieser Publikation bis zu dem Datum (stability 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.

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.

Der Zusammenhang der zitierten Normen mit den entsprechenden Deutschen Normen ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation. Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

Leitlinien zu Softwareaspekten der Zuverlässigkeit
(IEC 62628:2012)

Guidance on software aspects of dependability
(IEC 62628:2012)

Lignes directrices concernant la sûreté de
fonctionnement du logiciel
(CEI 62628:2012)

Diese Europäische Norm wurde von CENELEC am 2012-09-12 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 CEN-CENELEC Management Centre 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 CEN-CENELEC Management Centre mitgeteilt worden ist, hat den gleichen Status wie die offiziellen Fassungen.

CENELEC-Mitglieder sind die nationalen elektrotechnischen Komitees von Belgien, Bulgarien, Dänemark, Deutschland, der ehemaligen jugoslawischen Republik Mazedonien, 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, der Türkei, Ungarn, dem Vereinigten Königreich und Zypern.

CENELEC

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

Management Centre: Avenue Marnix 17, B-1000 Brüssel

© 2012 CENELEC – Alle Rechte der Verwertung, gleich in welcher Form und in welchem Verfahren, sind weltweit den Mitgliedern von CENELEC vorbehalten.

Ref. Nr. EN 62628:2012 D

Vorwort

Der Text des Dokuments 56/1469/FDIS, zukünftige 1. Ausgabe der IEC 62628, erarbeitet vom IEC/TC 56 „Dependability“, wurde zur parallelen IEC-CENELEC-Abstimmung vorgelegt und von CENELEC als EN 62628:2012 angenommen.

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): 2013-06-12
- spätestes Datum, zu dem nationale Normen, die der EN entgegenstehen, zurückgezogen werden müssen (dow): 2015-09-12

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

Anerkennungsnotiz

Der Text der Internationalen Norm IEC 62628:2012 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:

IEC 62508	ANMERKUNG	Harmonisiert als EN 62508.
IEC 60300-1	ANMERKUNG	Harmonisiert als EN 60300-1.
IEC 60300-2	ANMERKUNG	Harmonisiert als EN 60300-2.
IEC 60300-3-3	ANMERKUNG	Harmonisiert als EN 60300-3-3.
IEC 62347	ANMERKUNG	Harmonisiert als EN 62347.
IEC 61160	ANMERKUNG	Harmonisiert als EN 61160.
IEC 61078	ANMERKUNG	Harmonisiert als EN 61078.
IEC 61025	ANMERKUNG	Harmonisiert als EN 61025.
IEC 61165	ANMERKUNG	Harmonisiert als EN 61165.
IEC 62551 ¹⁾	ANMERKUNG	Harmonisiert als EN 62551 ¹⁾ .
IEC 60812	ANMERKUNG	Harmonisiert als EN 60812.
IEC 60300-3-1	ANMERKUNG	Harmonisiert als EN 60300-3-1.
IEC 61508-3	ANMERKUNG	Harmonisiert als EN 61508-3.
IEC 62429	ANMERKUNG	Harmonisiert als EN 62429.
IEC 61014	ANMERKUNG	Harmonisiert als EN 61014.
IEC 61164	ANMERKUNG	Harmonisiert als EN 61164.
IEC 62506 ¹⁾	ANMERKUNG	Harmonisiert als EN 62506 ¹⁾ .

¹⁾ Zu veröffentlichen.

Inhalt

	Seite
Vorwort	2
Einleitung	5
1 Anwendungsbereich	6
2 Normative Verweisungen	6
3 Begriffe und Abkürzungen	6
3.1 Begriffe	6
3.2 Abkürzungen	8
4 Allgemeines zu den Softwareaspekten der Zuverlässigkeit	9
4.1 Software und Softwaresysteme	9
4.2 Softwarezuverlässigkeit und Softwareorganisationen	9
4.3 Beziehungen zwischen Software- und Hardwarezuverlässigkeit	10
4.4 Wechselwirkung zwischen Software und Hardware	11
5 Technische Realisierung der Softwarezuverlässigkeit	12
5.1 Rahmenwerk für den Systemlebenszyklus	12
5.2 Projektbezogene Zuverlässigkeitstätigkeiten	12
5.3 Softwarelebenszyklustätigkeiten	12
5.4 Merkmale der Softwarezuverlässigkeit	14
5.5 Softwareentwicklungsumgebung	15
5.6 Erarbeitung der Anforderungen an die Software und der Zuverlässigkeitsziele	15
5.7 Klassifizierung von Softwarefehlern	16
5.8 Strategie zur Realisierung der Softwarezuverlässigkeit	17
6 Methodik zur Verwirklichung der Softwarezuverlässigkeit	18
6.1 Vorgehensweisen bei der Softwareentwicklung zur Erzielung der Zuverlässigkeit	18
6.2 Metriken der Softwarezuverlässigkeit und Datenerfassung	19
6.3 Beurteilung der Softwarezuverlässigkeit	20
6.4 Verbesserung der Softwarezuverlässigkeit	30
7 Softwaresicherung	35
7.1 Überblick über die Softwaresicherung	35
7.2 Zuschneideprozess	35
7.3 Technologische Einflüsse auf die Softwaresicherung	35
7.4 Optimale Verfahren der Softwaresicherung	36
Anhang A (informativ) Kategorisierung von Software und Softwareanwendungen	38
A.1 Kategorisierung von Software	38
A.2 Softwareanwendungen	39
Anhang B (informativ) Anforderungen an Softwaresysteme und zugehörige Zuverlässigkeitstätigkeiten	41
B.1 Allgemeines	41
B.2 Anforderungen ermitteln	41
B.3 Anforderungen analysieren	41

	Seite
B.4 Architekturentwurf.....	42
B.5 Detailentwurf.....	42
B.6 Realisierung.....	43
B.7 Integration.....	43
B.8 Abnahme	44
B.9 Betrieb und Wartung.....	44
B.10 Aktualisierung und Erweiterung der Software	44
B.11 Rückzug.....	44
Anhang C (informativ) Capability Maturity Model Integration-Prozess.....	45
Anhang D (informativ) Klassifizierung von Softwarefehlerattributen	48
D.1 Allgemeines	48
D.2 Öffner (Opener Section)	48
D.3 Schließer (Closer Section).....	49
D.4 Auslösende Tätigkeit	50
Anhang E (informativ) Beispiele für aus der Datenerfassung erhaltene Softwaremetriken	53
E.1 Fehlerdatenmetriken.....	53
E.2 Produktdatenmetriken	54
E.3 Prozessdatenmetriken.....	54
Anhang F (informativ) Beispiel für kombinierte Hardware/Softwarezuverlässigkeitsfunktionen	56
Anhang G (informativ) Zusammenstellung von in Softwarefunktionsfähigkeitsmodellen verwendeten Maßgrößen	58
Anhang H (informativ) Auswahl und Anwendung von Modellen für Softwarezuverlässigkeit	59
Literaturhinweise.....	62
Anhang ZA (normativ) Normative Verweisungen auf internationale Publikationen mit ihren entsprechenden europäischen Publikationen	65
Bild 1 – Tätigkeiten im Softwarelebenszyklus	14
Bild F.1 – Blockdiagramm eines Überwachungs- und Steuerungssystems	56
Tabelle C.1 – Vergleich von Fähigkeitsgraden und Reifegraden	45
Tabelle D.1 – Klassifizierung von Softwarefehlerattributen nach der Fehlerentdeckung.....	49
Tabelle D.2 – Klassifizierung von Softwarefehlerattributen bezüglich Fehlerbeseitigung	49
Tabelle D.3 – Bei Entwurfsbewertung/Codeinspektion bewirkte Auslöser.....	50
Tabelle D.4 – Beim Einheitentest bewirkte Auslöser	51
Tabelle D.5 – Beim Funktionstest bewirkte Auslöser	51
Tabelle D.6 – Beim Systemtest bewirkte Auslöser.....	52
Tabelle H.1 – Beispiele von Modellen für Softwarefunktionsfähigkeit.....	59

Einleitung

Software wird in heutigen Produkten und Systemen in großem Stil angewendet. Beispiele sind Softwareanwendungen in speicherprogrammierbaren Steuerungen, in Rechnersystemen und in Kommunikationsnetzen. Im Laufe der Jahre sind viele Normen und Standards zu Softwaretechnik, Softwareprozesslenkung, Softwarequalität und Zuverlässigkeitssicherung entwickelt worden, aber nur in wenigen Normen und Standards wurden Softwareprobleme aus Sicht der Zuverlässigkeit behandelt.

Zuverlässigkeit ist die Fähigkeit eines Systems, geforderte Leistungen unter gegebenen Einsatzbedingungen auf Anforderung zu erfüllen. Aus der Zuverlässigkeit eines Systems kann man ableiten, ob das System vertrauenswürdig und fähig ist, den gewünschten Dienst auf Nachfrage zur Erfüllung der Anwenderbedürfnisse zu erbringen. Die zunehmenden Trends hin zu Softwareanwendungen in der Dienstleistungsbranche haben zu dem schnellen Wachstum der Internetdienste und Netzentwicklung geführt. Standardisierte Schnittstellen und Protokolle haben den Weg frei gemacht für die Bereitstellung von Softwarefunktionalität seitens Dritter über das Internet und ermöglichen so plattform-, dienstanbieter-, und domänenübergreifende Anwendungen. Software ist zur treibenden Kraft bei der Realisierung komplexer Systemoperationen geworden, und sie ermöglicht es, elektronische Geschäftsvorgänge im Sinne einer nahtlosen Integration der Unternehmensführungsprozesse zu realisieren. Der Gestaltung der Software kommt jetzt bei der Datenverarbeitung, der Überwachung und dem Schutz sicherheitsrelevanter Güter und bei den Kommunikationsbeziehungen in Netzdiensten die Schlüsselrolle zu. Dieser Paradigmenwechsel hat die globale Geschäftswelt dazu gebracht, weitestgehend einer Situation zu vertrauen, die in ganz besonderem Maße auf Softwaresystemen beruht und auf die sie wiederum ihre geschäftlichen Vorgänge aufbaut. Für das Erreichen einer erfolgreichen Systemleistung und einer unversehrten Datenintegrität spielt die Softwarezuverlässigkeit eine dominante Rolle.

In dieser Norm werden die gegenwärtig in der Industrie üblichen Vorgehensweisen vorgestellt, und es wird eine Methodik beschrieben, wie eine hohe Softwarezuverlässigkeit leichter erreicht werden kann. Der Einfluss der Unternehmensführung auf die Software im Hinblick auf die Zuverlässigkeit wird aufgezeigt, und es werden passende technische Prozesse beschrieben, wie Softwarezuverlässigkeit in Systemen technisch realisiert werden kann. Die Evolution der Softwaretechnik und die schnelle Adaptation von Softwareanwendungen in der industriellen Praxis haben das Bedürfnis für eine praxisorientierte Normung der Softwarezuverlässigkeit im Hinblick auf weltweite Geschäftsbeziehungen geschaffen. Mittels einer strukturierten Vorgehensweise wird zur Anwendung dieser Norm angeleitet.

In dieser Norm werden die allgemeinen Voraussetzungen zur Erzielung einer zuverlässigen Software und die dafür notwendigen Softwareprozesse dargestellt. Auf der Grundlage dieser Voraussetzungen und dieser Prozesse können mit Blick auf die Zuverlässigkeit die meisten Softwareprodukte entwickelt und Softwaresysteme realisiert werden. Für kritische Aufgaben und bei sicherheitsrelevanten Anwendungen sind zusätzliche Anforderungen vonnöten. Industriespezifische Belange der Softwarequalifizierung mit Blick auf die Konformität von Zuverlässigkeit und Qualität werden in dieser Norm nicht behandelt.

Diese Norm kann auch als Anleitung zur zuverlässigen Gestaltung und Entwicklung von Firmware dienen. Sie behandelt jedoch weder die Gesichtspunkte der Implementierung von Firmware, die Software enthält, noch die Implementierung von in Hardwarechips eingebetteter Firmware zur Realisierung ganz bestimmter, festgelegter Funktionen. Beispiele hierfür sind anwendungsspezifische integrierte Schaltungen (ASIC) und durch Mikroprozessor gesteuerte Geräte. Diese Produkte werden häufig als Teil der physischen Hardwaremerkmale entwickelt und in die Hardware integriert, um deren Abmessungen und Gewicht zu minimieren, um so Echtzeitanwendungen zu erleichtern, wie dies beispielsweise in Mobiltelefonen der Fall ist. Obwohl die in dieser Norm beschriebenen allgemeinen Zuverlässigkeitsgrundsätze und Vorgehensweisen beim Firmwareentwurf und der Firmwareanwendung als Leitlinien verwendet werden können, so sind doch für deren Konstruktion, Teileanfertigung und für die Implementierung von in Produkte eingebetteter Software spezielle Anforderungen vonnöten. Die physikalischen Vorgänge beim Ausfall anwendungsspezifischer Schaltungen sind von anderer Natur, als dies beim Ausfall von Softwaresystemen der Fall ist.

Diese Internationale Norm ist nicht zu Zwecken der Zertifizierung oder Konformitätsbewertung vorgesehen.

1 Anwendungsbereich

Diese Norm behandelt die Auswirkung von Software auf die Zuverlässigkeit, und sie leitet dazu an, wie eine zuverlässige Software abhängig von Führungsgrundsätzen, Entwicklungsprozessen und Anwendungsumgebungen erreicht werden kann. Es wird ein allgemeines Rahmenwerk der Anforderungen an die Software im Hinblick auf Zuverlässigkeit ausgearbeitet, es wird ein Zuverlässigkeitsprozess für die Software in Lebenszyklusprozessen vorgestellt, es werden Beurteilungskriterien und eine Methodik für den Entwurf und die Implementierung der Software im Hinblick auf Zuverlässigkeit präsentiert und praktische Vorgehensweisen aufgezeigt, wie die Leistungsfähigkeit bewertet und Zuverlässigkeitskenngrößen in Softwaresystemen gemessen werden können.

Diese Norm ist geeignet für Entwickler und Lieferanten von Softwaresystemen, Systemintegratoren, Betreiber und Wartungspersonal sowie für Anwender von Softwaresystemen, die mit praktischen Vorgehensweisen und der technischen Umsetzung befasst sind, um zuverlässige Softwareprodukte und Systeme zu erreichen.

2 Normative Verweisungen

Die folgenden Dokumente, die in diesem Dokument teilweise oder als Ganzes zitiert werden, 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 60050-191, *International Electrotechnical Vocabulary – Chapter 191: Dependability and quality of service*

IEC 60300-3-15, *Dependability management – Part 3-15: Application guide – Engineering of system dependability*

3 Begriffe und Abkürzungen

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC 60050-191 und die folgenden Begriffe.

3.1 Begriffe

3.1.1

Software

Programme, Verfahren, Regeln, Dokumentation und Daten eines Informationen verarbeitenden Systems

Anmerkung 1 zum Begriff: Software ist eine intellektuelle Schöpfung, die unabhängig davon ist, auf welchem Medium sie gespeichert ist.

Anmerkung 2 zum Begriff: Zur Programmausführung, zur Speicherung von Daten und für deren Übertragung benötigt Software Hardwaregeräte.

Anmerkung 3 zum Begriff: Softwarearten sind u. a. Firmware, Systemsoftware und Anwendungssoftware.

Anmerkung 4 zum Begriff: Zur Dokumentation gehören: Anforderungsspezifikationen, Entwurfsspezifikationen, Quelltextausdrucke, Kommentare im Quelltext, Hilfetext und Hinweise, die über die Rechner/Mensch-Schnittstelle angezeigt werden sollen, Installationsanweisungen, Betriebsanweisungen, Anwenderhandbücher und Handbücher zur Unterstützung der Softwarewartung.

3.1.2

Firmware

in einem Festwertspeicher enthaltene Software, die nicht zur Modifizierung vorgesehen ist

BEISPIEL BIOS (en: Basic input/output system) eines Personalcomputers.

Anmerkung 1 zum Begriff: Zur Modifizierung der Software muss das sie enthaltende Gerät entweder ausgetauscht oder neu programmiert werden.

3.1.3

eingebettete Software

Software in einem System, deren Hauptzweck nicht rechenbetont ist

BEISPIEL Software zur Motorsteuerung oder zur Steuerung des Bremssystems eines Kraftfahrzeuges.

3.1.4

Softwareeinheit

Softwaremodul

Softwareelement, das für sich alleine in Rechnerprogrammcode kompiliert werden kann, um eine Aufgabe auszuführen oder eine Tätigkeit zu starten, um das gewünschte Ergebnis einer oder mehrerer Softwarefunktion(en) zu erzielen

Anmerkung 1 zum Begriff: Die Begriffe Modul und Einheit werden häufig je nach dem Textzusammenhang in austauschbarer Weise verwendet oder als Unterelemente voneinander auf unterschiedliche Weisen definiert. Die Beziehung dieser Begriffe untereinander ist noch nicht genormt.

Anmerkung 2 zum Begriff: In einer idealen Situation kann eine Softwareeinheit so entworfen und programmiert werden, dass diese exakt eine spezifizierte Funktion ausführt. Bei manchen Anwendungen kann es erforderlich sein, zwei oder mehr Softwareeinheiten zu kombinieren, um so zusammen die spezifizierte Funktion zu erreichen. In solchen Fällen werden die Softwareeinheiten wie eine einzige Softwarefunktion getestet.

3.1.5

Softwarekonfigurationseinheit

Softwareeinheit, die als eine einzelne Einheit im Konfigurationsmanagementprozess konfiguriert und behandelt wird

Anmerkung 1 zum Begriff: Eine Softwarekonfigurationseinheit kann aus einer oder mehreren Softwareeinheiten zur Durchführung einer Softwarefunktion bestehen.

3.1.6

Softwarefunktion

durch ein Softwaremodul oder eine Softwareeinheit wie spezifiziert oder gemäß vorgegebenen Anforderungen ausgeführte elementare Operation

3.1.7

Softwaresystem

definierter Satz von Softwareeinheiten, die sich nach ihrer Integration zusammen so verhalten, dass sie vorgegebene Anforderungen erfüllen

BEISPIELE Anwendungssoftware (Software für Abrechnungs- und Informationsverarbeitung); Programmiersoftware (Software zur Leistungsanalyse und rechnergestützte Werkzeuge) und Systemsoftware (Software zur Steuerung und Überwachung von Rechnerhardware wie etwa Betriebssysteme).

3.1.8

Softwarezuverlässigkeit

Fähigkeit einer Softwareeinheit, sich nach ihrer Integration in den Systembetrieb auf Anforderung wie gefordert zu verhalten

3.1.9

Softwarefehler

Bug

Zustand einer Softwareeinheit, in dem sie unfähig sein kann, sich wie gefordert zu verhalten

Anmerkung 1 zum Begriff: Softwarefehler sind entweder Spezifikationsfehler, Entwurfsfehler, Programmierfehler, durch den Compiler eingeschleuste Fehler oder während der Softwarewartung eingeschleuste Fehler.

Anmerkung 2 zum Begriff: Ein Softwarefehler ist solange „schlafend“, bis er durch einen speziellen Auslöser (Trigger) aktiviert wird, und er kehrt üblicherweise in den Schlafzustand zurück, wenn der Trigger entfernt wird.

Anmerkung 3 zum Begriff: Im Zusammenhang mit dieser Norm ist ein Bug ein Sonderfall eines Softwarefehlers, der auch unter der Bezeichnung latenter Softwarefehler bekannt ist.

3.1.10

Softwareausfall

Ausfall, der das Sichtbarwerden oder Auftauchen (Manifestation) eines Softwarefehlers ist

Anmerkung 1 zum Begriff: Ein einzelner Softwarefehler wird solange wieder auftauchen und sich als Ausfall manifestieren, bis der Fehler beseitigt ist.

3.1.11

Code

Zeichen- oder Bit-Muster, dem eine spezielle Bedeutung zugewiesen ist, um ein Rechnerprogramm in einer Programmiersprache auszudrücken

Anmerkung 1 zum Begriff: Quelltext (Quellcode) sind kodierte Anweisungen und Datendefinitionen, die in einer Form ausgedrückt sind, die zur Eingabe in Assembler, Compiler oder andere Übersetzerprogramme geeignet sind.

Anmerkung 2 zum Begriff: Kodierung ist der Prozess der Transformierung von Logik und Daten aus Entwurfsspezifikationen oder Beschreibungen in eine Programmiersprache.

Anmerkung 3 zum Begriff: Eine Programmiersprache ist eine Sprache, durch die Rechnerprogramme ausgeführt werden können.

3.1.12

(Rechner-)Programm

Satz kodierter Anweisungen, der ausgeführt wird, um festgelegte logische und mathematische Operationen an Daten vorzunehmen

Anmerkung 1 zum Begriff: Programmierung ist die generelle Tätigkeit bei der Softwareentwicklung, bei der der Programmierer oder Rechneranwender einen festgelegten Satz von Anweisungen vorgibt, die der Rechner ausführen muss.

Anmerkung 2 zum Begriff: Ein Programm besteht aus einer Kombination kodierter Anweisungen und Datendefinitionen, die es der Rechnerhardware ermöglicht, rechenbetonte oder steuerungsbetonte Funktionen auszuführen.

3.2 Abkürzungen

ASIC	anwendungsspezifische integrierte Schaltung (en: Application specific integrated circuit)
CASE	rechnergestützte Softwareentwicklung (en: Computer-aided software engineering)
CMM	Reifegradmodell (en: Capability maturity model)
CMMI	Capability maturity model integration
COTS	handelsüblich (en: Commercial-off-the-shelf)
FMEA	Fehlzustandsart- und –auswirkungsanalyse (en: Failure mode and effects analysis)
FTA	Fehlzustandsbaumanalyse (en: Fault tree analysis)
IP	Internetprotokoll
IT	Informationstechnik

KSLOC	Tausend-Zeilen-Quelltext (en: Kilo (thousand) source lines of code)
ODC	Modell zur Fehlerklassifizierung (en: Orthogonal defect classification)
RBD	Zuverlässigkeitsblockdiagramm (en: Reliability block diagram)
USB	Universal serial bus

4 Allgemeines zu den Softwareaspekten der Zuverlässigkeit

4.1 Software und Softwaresysteme

Software ist eine virtuelle Betrachtungseinheit. Im Zusammenhang mit dieser Norm bezieht sich Software auf Prozeduren, Programme, Codes, Daten und Anweisungen zur Systemsteuerung und Informationsverarbeitung. Ein Softwaresystem besteht aus einer integrierten Zusammenstellung von Softwareeinheiten wie Rechnerprogrammen, Prozeduren und ausführbaren Codes, die in eine physische Hostrechnerhardware eingegliedert sind und zusammen den Systembetrieb ermöglichen und die geforderten Funktionen erbringen. Die Hierarchie des Softwaresystems kann als ein Gebilde aus Systemarchitektur und Subsystem-Softwareprogrammen und Softwareeinheiten der unteren Ebenen angesehen werden. Eine Softwareeinheit kann getestet werden, wie dies beim Entwerfen eines Programmes spezifiziert wurde. Zur Bildung einer Softwarefunktion können in manchen Fällen zwei oder mehr Softwareeinheiten erforderlich sein. Das System umfasst sowohl Hardware- als auch Softwareelemente, die zur Erbringung nützlicher Funktionen zusammenwirken und so gemeinsam die geforderten Dienste leisten.

In einem kombinierten Hardware/Software-System tragen die Softwareelemente des Systems auf zwei wichtige Weisen zum Gesamtsystem bei: a) Die Betriebssoftware läuft ohne Unterbrechung und unterstützt so die Hardwareelemente beim Systembetrieb; und b) Anwendungssoftware läuft auf Anforderung des Nutzers, um spezielle Dienste für den Nutzer zu erbringen. Bei der Zuverlässigkeitsanalyse der Subsystemsoftware müssen die sich aus der Softwareanwendung ergebenden Zeitfaktoren im Systembetriebsprofil berücksichtigt werden sowie auch solche Softwareelemente, die für einen Dauerbetrieb des Systems benötigt werden. Das System muss für die Aufteilung und zur Beurteilung der Zuverlässigkeit softwarebasierter Systeme modelliert werden.

Die Beachtung der menschlichen Aspekte der Zuverlässigkeit [1]¹⁾ spielt eine Schlüsselrolle auf dem Weg hin zu einem erfolgreichen Softwareentwurf und einer erfolgreichen Implementierung. Die Mensch-Maschine-Schnittstelle und die Betriebsumgebung beeinflussen das Ergebnis des Zusammenwirkens der Software mit der Hardware und wirken sich auf die Zuverlässigkeit des Systemverhaltens aus. Daraus ergibt sich die strategische Notwendigkeit eines Softwarezuverlässigkeitsentwurfes und einer angemessenen Produktpflege im Softwarelebenszyklusprozess [2].

4.2 Softwarezuverlässigkeit und Softwareorganisationen

Softwarezuverlässigkeit wird erzielt durch einen sauberen Entwurf und eine adäquate Eingliederung in den Systembetrieb. Diese Norm beschreibt, wie bestehende Zuverlässigkeitstechniken und bekanntermaßen erfolgreiche industrielle Vorgehensweisen ermittelt und zum Softwarezuverlässigkeitsentwurf und zur Softwareimplementierung verwendet werden können. Die Zuverlässigkeitsmanagementsysteme [3], [4] beschreiben, wie und wo relevante Zuverlässigkeitsmaßnahmen wirkungsvoll im Lebenszyklusprozess implementiert werden können. Das Erreichen der Softwarezuverlässigkeit wird beeinflusst durch

- Unternehmenspolitik und technische Führung,
- Entwurfs- und Implementierungsprozesse,
- projektspezifische Bedürfnisse und Anwendungsumgebungen.

¹⁾ Verweisungen in rechteckigen Klammern beziehen sich auf die Literaturhinweise.

Softwareorganisationen sind organisierte und geführte Gruppen, die über Mitarbeiter und Einrichtungen verfügen mit Verantwortlichkeiten, Zuständigkeiten und Beziehungen, zu denen Software als Teil ihrer Routinetätigkeiten gehört. Sie gibt es in staatlichen Stellen, in öffentlichen und privaten Gesellschaften, in Firmen, Vereinigungen und Institutionen. Softwareorganisationen sind nach den jeweiligen Geschäftsbedürfnissen und Anwendungsumgebungen für verschiedene Kombinationen von Entwicklung, Betrieb und Dienstleistungen strukturiert.

Typische Softwareorganisationen sind u. a. solche,

- a) die Software als ihr primäres Produkt entwickeln,
- b) die Hardwareprodukte mit eingebetteter Software entwickeln,
- c) die ihren Kunden Softwareunterstützungsdienste bereitstellen,
- d) die Softwareretzwerke und Systeme betreiben und warten.

Anhang A enthält eine Kategorisierung von Software und Softwareanwendungen, wie sie von typischen Softwareorganisationen bereitgestellt werden.

4.3 Beziehungen zwischen Software- und Hardwarezuverlässigkeit

Aus Sicht der Zuverlässigkeit sind das Softwareverhalten und die Leistungsmerkmale anders als die bei Hardware beobachteten Merkmale. Softwarecodes werden von Menschen geschaffen, diese Arbeiten unterliegen menschlichen Fehlhandlungen, welche durch die Entwicklungsumgebung und die Organisationskultur beeinflusst sind. Die meisten Ausfalldaten von Hardware und Geräten sind aus tatsächlichen Anwendungen wohl bekannt und dokumentiert, wogegen die Natur von Softwarefehlern und ihre Rückverfolgbarkeit von Ursache und Wirkung im Systembetrieb immer noch nicht leicht zu bestimmen sind. In den meisten Fällen können zu Systemausfällen führende Softwarefehler immer noch nicht schlüssig dupliziert werden. Vollzogene Korrekturmaßnahmen nach durch Softwarefehler eingetretenen Systemausfällen garantieren keine vollständige Beseitigung der eigentlichen Ursachen des Softwareproblems.

Ein Bug führt nach seiner Auslösung zu einem Softwareausfall (Ereignis) und stellt einen Softwarefehler (Zustand) dar. Alle Softwarefehler, die dazu führen, dass die Software ihre beabsichtigten Funktionen nicht mehr erbringen kann, werden vom Softwareanwender bemerkt. Fehler und Bugs führen dazu, dass die Software sich nicht mehr wie beabsichtigt verhält. Eine Bugs enthaltende Software könnte immer noch ihre beabsichtigte Funktion, die für den Anwender nicht bemerkbar ist, erfolgreich ausführen. Bugs könnten Ausfälle bewirken, sie könnten aber auch zu störenden Nebenwirkungen führen, die keine bestimmte Funktion betreffen. Ein Softwarefehler kann einen Systemausfall bewirken, der systematische Ausfallsymptome zeigt.

Softwaresysteme und Hardwareprodukte (Geräte) haben aber auch viele Ähnlichkeiten. Ihre jeweiligen Entwurfs- und Entwicklungsstufen werden entsprechend einem Plan gelenkt und überwacht, gleiches gilt für die nachfolgenden Stufen Integration, Test (Prüfung) und Produktion. Die Entdeckung von Ausfällen und latenten Fehlern erfolgt durch eine rigorose Analyse, durch einen Test- und Prüfprozess mit umfänglicher Test- oder Fehlerabdeckung. Der Grad des Prüfumfanges wird bestimmt durch die Taxierung des entdeckten Fehleranteils in Prozent oder die Wahrscheinlichkeit der Fehlerentdeckung. Obwohl die Managementtechniken ähnlich sind, gibt es auch Unterschiede [5], [6]. Nachfolgend einige Beispiele hierzu:

- Software besitzt keine körperlichen Eigenschaften, Hardware hat diese jedoch. Software verschleißt nicht. Ausfälle, die Softwarefehlern zuordenbar sind, treten ohne vorherige Warnung auf, und es gibt oft keinen Hinweis, dass sie eingetreten sind. Hardware erfährt oft einen graduellen Verschleiß und verschlechtert sich möglicherweise langsam, bis ein ausgefallener Zustand erreicht wird.
- Änderungen an der Software sind flexibel und viel weniger zeitaufwändig oder teuer als Änderungen am Hardwareentwurf. Änderungen am Hardwareentwurf erfordern eine ganze Reihe zeitaufwändiger Anpassungen bei Schlüsselgeräten, Materialbeschaffung, Fertigung, Zusammenbau und Dokumentation. Andererseits könnte jedoch das Regressionstesten von großen und komplexen Softwareprogrammen durch Zeit- und Kostengrenzen beschränkt werden.
- Das Prüfen der Hardware (Testen und Verifizieren) stellt sich einfacher dar, da es durch begrenztes Prüfen aufgrund der Kenntnisse über die Gerätephysik möglich ist, das Gerät zu analysieren und sein Verhalten vorherzusagen. Softwaretesten kann aber auch durch Regressionstesten und Analysieren vereinfacht werden, wenn es darum geht, aufgrund einer erkannten Ausfallursache vollzogene kleinere

Änderungen zu verifizieren. Jedoch könnten kleinere Änderungen, die eine probabilistische Ausfallursache der Software beheben sollen, wie etwa kritische Wettläufe (race conditions), zu sehr ausgeklügelten Test- und Verifizierungszyklen führen, damit die adäquate Korrektur des Problems nachgewiesen werden kann.

- Instandhaltung (also Wartung und Reparatur) würde die Hardware in ihren betriebsfähigen Zustand zurückversetzen, und dies im Allgemeinen ohne Änderungen am Entwurf beziehungsweise der Konstruktion. Softwarereparatur und Wartung würden dagegen Änderungen am Entwurf bedingen und neue Wartungspakete (Service Packs) oder Softwarereleases nach sich ziehen, um dadurch Softwarefehler zu korrigieren oder zu beheben.

4.4 Wechselwirkung zwischen Software und Hardware

Im Systembetrieb findet eine Wechselwirkung zwischen Software und Hardware statt. Die Schnittstelle zwischen der Hardware und dem Betriebssystem ist für die Zuverlässigkeit von Belang. Die dort zu beachtenden Punkte werden üblicherweise durch Techniken zur Erkennung und Korrektur von Fehlaussagen gelöst sowie durch eine Ausnahmebehandlung der Hardware und des Betriebssystems zur Abschwächung der Wirkung körperlicher Fehler und durch die Beseitigung fehlerhafter Informationen und durch die Korrektur einer irrtümlichen zeitlichen Abstimmung der Wechselwirkung. Das Aufkommen von Mehrkernprozessoren hat ein redundantes gleichzeitiges Abarbeiten mehrerer Ausführungsstränge (Multithreading) ermöglicht, um so die Zuverlässigkeit des Systemverhaltens zu erhöhen. Dieses versetzt den Anwender, Programmierer oder Systemarchitekten in die Lage, die den Mehrkernprozessoren innewohnende Redundanz so zu beeinflussen und auszunützen, dass die Erkennung von Fehlaussagen und Irrtümern und die sich daran anschließende Erholung verbessert werden. Dies bietet auch die Gelegenheit zur Erholung von weichen Fehlaussagen und Irrtümern oder transienten Fehlaussagen und Irrtümern, die sich entweder auf die Hardware oder auf die Software oder beide auswirken. Die Ausnützung der erhöhten Komplexität bei der Mehrkernredundanz sollte bei solchen Anwendungen beachtet werden.

In jedem Leitsystem steuert das System einige physikalische Prozesse aktueller Hardwaregeräte wie beispielsweise Sensoren und Aktuatoren, die im Systembetrieb ausfallen können. Viele dieser Geräte enthalten eingebettete Software, die dem Systementwickler oder -architekten nicht zugänglich sind. Beispiele sind intelligente Sensoren, die Fehlaussagen erkennen können, die Redundanz enthalten und die gewisse Fehlerbehebungsmechanismen enthalten, die durch die eingebettete Software gesteuert werden. Es ist wichtig, die Softwaresteuerungsalgorithmen zu überprüfen. Dadurch soll sichergestellt werden, dass die Steuerungsalgorithmen angesichts schlechter Sensordaten oder fehlender Sensorwerte belastbar und dehnfähig reagieren und dass sie eine ausgebliebene (ausgefallene) Betätigung erkennen können und in der Lage sind, solche Situationen zu kompensieren oder wieder in einen ausfallsicheren Zustand zurückzukehren. Die Rückmeldung von Sensoren ist essentiell zur Bestätigung von erfolgreicher Betätigung. Der Rückmeldemechanismus sollte eine gewisse unabhängige Überprüfung der Wirkung der veranlassten Betätigung enthalten. Das Verhalten des Steuerungssystems, Annahmen und Ausfallarten sollten beim Entwurf des Softwaresteuerungssystems beachtet werden.

Falls das System vorsätzlich Cyber-Angriffen ausgesetzt ist, könnten Hardwarefehler absichtlich und bösartig injiziert werden mit dem Ziel, die Softwarealgorithmen zu hintertreiben oder sie zu durchkreuzen. Beispielsweise kann man Hardwarefehler in ein kryptographisches System injizieren, um den Schlüssel herauszufinden, oder man kann einen Virus in ein USB-Gerät einschleusen, das dazu dient, eine Abstimmungsmaschine zu initialisieren. Die Wechselwirkung zwischen Software und Hardware könnte dem Systembetrieb ernsthafte Probleme bereiten und so die Zuverlässigkeit des Systemverhaltens nachteilig beeinflussen.

Interoperabilitätsprobleme in Verbindung mit der Wechselwirkung zwischen Software und Hardware könnten ebenfalls vorhanden sein, falls die Software in unpassender Weise in einer andersartigen Umgebung oder für eine andersartige Anwendung wiederverwendet wird.

Die Lösung von Zuverlässigkeitsproblemen bezüglich der Wechselwirkung zwischen Software und Hardware kann durch ein besseres Verständnis der Art und Weise erfolgen, wie das neue technische System arbeitet, sowie durch eine sorgfältige Bewertung der Zuverlässigkeit, bei der die Auswirkungen von Hardwareausfällen auf das Softwaresystem vollständig geprüft und zu ihrer Beurteilung herangezogen werden.

5 Technische Realisierung der Softwarezuverlässigkeit

5.1 Rahmenwerk für den Systemlebenszyklus

Es sollte ein Rahmenwerk für den Systemlebenszyklus eingerichtet werden, der die Entwicklung des Systems und die Realisierung des Produktes begleitet. Das Rahmenwerk hilft bei der Definition der Systemlebenszyklen und bei der Durchführung der Systemlebenszyklusprozesse. IEC 60300-3-15 beschreibt die technische Realisierung der Systemzuverlässigkeit und die Umsetzung der Lebenszyklen, die sich an den in ISO/IEC 15288 [7] beschriebenen technischen Prozessen orientiert. Dies gilt für jedwedes System, unabhängig davon, ob dieses aus Hardware, Software oder beidem zusammengesetzt ist.

5.2 Projektbezogene Zuverlässigkeitstätigkeiten

Die Tätigkeiten zur technischen Realisierung der Software während der im Systemlebenszyklus beschriebenen Entwurfszyklen und in der Nutzungsphase sollten auf die Hardwaregegenstände abgestimmt geplant, koordiniert und gelenkt werden. Zu den technischen Tätigkeiten in der Nutzungsphase gehören üblicherweise Entwurfsänderungen, die durch hohe Ausfallraten in der Kundenanwendung veranlasst werden oder die sich durch obsoletere Hardware ergeben, weil keine passenden Ersatzteile mehr für die Aufrechterhaltung des Betriebs erhältlich sind. So wie sich die Hardware im Ablauf des Produktlebenszyklus ändert, sollte auch die Software gleichermaßen geändert werden. Änderungen an der Software sind notwendig, da der Systementwurf eine vorwärts und rückwärts gerichtete Kompatibilität zwischen den verschiedenen Versionen und Konfigurationen des Systementwurfs erfordert.

Zuverlässigkeitstätigkeiten sollten im Hinblick auf erfolgreiche Systementwicklung, Herstellung, Implementierung, Betrieb und Instandhaltung in die jeweiligen Projektpläne integriert und Bestandteil der Aufgaben zur Realisierung des Systems werden. Die Anleitung zur technischen Realisierung der Zuverlässigkeit in Systemen nach IEC 60300-3-15 gilt für diese Norm mit. Die folgenden Tätigkeiten werden im Hinblick auf das Erreichen der Softwarezuverlässigkeit während der Implementierung des Softwareprojektes empfohlen:

- a) die Ziele und Anforderungen an die Software während des Softwarelebenszyklus (siehe 5.3) und hinsichtlich der Anwendungsumgebung (siehe Abschnitt A.2) ermitteln;
- b) die für das Softwareprojekt zutreffenden Merkmale der Softwarezuverlässigkeit (siehe 5.4) ermitteln;
- c) die Eignung der Zuverlässigkeitsmanagementprozesse und der verfügbaren Ressourcen für die Unterstützung der Softwareprojektentwicklung und Implementierung (siehe 5.5) überprüfen;
- d) die Anforderungen an die Software und die Zuverlässigkeitsziele (siehe 5.6, Anhang B) festlegen;
- e) Softwarefehler (siehe 5.7) klassifizieren und zutreffende Softwaremetriken (siehe 6.2, Anhang E) für die Umsetzung der Softwarezuverlässigkeitsstrategie (siehe 5.8) ermitteln;
- f) Erfolg versprechende Zuverlässigkeitsmethoden auf den Softwareentwurf und die Softwarerealisierung (siehe 6.1, 6.3) anwenden;
- g) Zuverlässigkeitsverbesserung, wo nötig, unter Beachtung verschiedener Einschränkungen im Projektzuschnitt (siehe 6.4, 7.2) initiieren;
- h) den Entwicklungs- und Implementierungsprozess zur Kontrolle und Rückmeldung mit dem Ziel der Aufrechterhaltung der Softwarebetriebsfähigkeit überwachen und die Zuverlässigkeit im Systembetrieb (siehe Abschnitt 7) sichern.

5.3 Softwarelebenszyklustätigkeiten

Der Softwarelebenszyklus umfasst die folgenden Tätigkeiten:

- *Anforderungen festlegen:* Aus Systemsicht werden die Anforderungen an kombinierte Hardware- und Softwareelemente unter Berücksichtigung der Bedürfnisse der Anwender und der Einschränkungen der Systemanwendungen ermittelt.
- *Anforderungen analysieren:* Mögliche Entwurfsalternativen werden ermittelt, und die Systemanforderungen für die Erfüllung der betrieblichen Anwendungen werden in eine technische Vorgabe für die Entwicklung der Hardware- und Softwaresubsysteme und des Systems selbst umgesetzt.

- *Architekturentwurf*: In dieser Stufe wird eine Lösung erarbeitet, wie die Systemanforderungen durch Aufteilung der Systemelemente in Bausteine für den Bau der Subsysteme erfüllt werden können, um so ein Fundament für die Zerlegung und Aufteilung der Softwaresubsysteme anzulegen und relevante Softwarefunktionen ermitteln zu können, mit denen die geforderten Softwaremerkmale dann erfüllt werden können.
- *Detailentwurf*: In dieser Stufe wird ein Entwurf für jede in der Systemarchitektur ermittelte Funktion erarbeitet und entwickelt; die benötigten Softwareeinheiten und Schnittstellen für die Funktion werden erschaffen, die auf die Software, die Hardware oder beide aufgeteilt werden können. Die so auf die Software aufgeteilten Funktionen werden ausreichend detailliert definiert, um darauf aufbauend kodieren und testen zu können. Die Softwarefunktion kann man als ein Softwaresubsystem bezeichnen, und sie kann zu Zwecken der Entwurfslenkung als ein Softwarekonfigurationsteil gekennzeichnet werden.
- *Realisierung*: In dieser Stufe werden die ausführbaren Softwareeinheiten produziert; diese müssen Verifizierungskriterien und Entwurfsanforderungen erfüllen. Dies schließt Tätigkeiten in unteren Ebenen ein wie:
 - *Kodierung* der Softwareeinheiten;
 - *Einheitentest* zur Verifizierung, ob die Softwareeinheit die Entwurfsvorgaben erfüllt;
 - *Subsystemtest* zur Verifizierung, ob die Funktionen des Softwareprogramms die Entwurfsvorgaben erfüllen.
- *Integration*: In dieser Stufe werden die Softwareeinheiten und die Subsysteme gemäß dem Architekturentwurf zusammengebaut (assembliert), und das gesamte Softwaresystem wird in das Hosthardwaresystem zum Testen installiert.
- *Abnahme*: In dieser Stufe werden die Systemfähigkeiten hergestellt und die Softwareanwendungen daraufhin validiert, ob sie die geforderten Dienste im spezifizierten Systembetrieb in der beabsichtigten Zielumgebung erbringen; Softwareabnahmetests und Prüfungen schließen Tätigkeiten in unteren Ebenen ein wie:
 - *Zuverlässigkeitswachstumsprüfungen* zur Erhöhung der Funktionsfähigkeit des Softwaresystems; das Testen erfolgt nach vollständiger Integration des Softwaresystems und dessen Ausführung bei simulierten, die Zielumgebung nachbildenden Betriebs- und Einsatzbedingungen;
 - *Qualifikationsprüfungen* zur Validierung der Abnahme des Softwaresystems zur Freigabe an den Kunden.
- *Softwarebetrieb und Wartung*: In dieser Stufe trägt die Software zum Betrieb des Systems bei, hält die betrieblichen Fähigkeiten des Systems aufrecht und stellt nach Aufforderung die jeweiligen betrieblichen Dienste und Leistungen bereit.
- *Softwareaktualisierung/Erweiterung* verbessert die Softwareleistung durch zusätzliche Merkmale.
- *Entsorgung der Software* beendet die betreffende Softwaredienstleistung.

Im Anhang B sind typische Anforderungen an das Softwaresystem und die Zuverlässigkeitstätigkeiten in den entsprechenden Stufen des Softwarelebenszyklus dargestellt.

In Bild 1 sind die für den Softwarelebenszyklus wichtigen Zuverlässigkeitstätigkeiten dargestellt, die in das Projekt aufgenommen werden sollten.

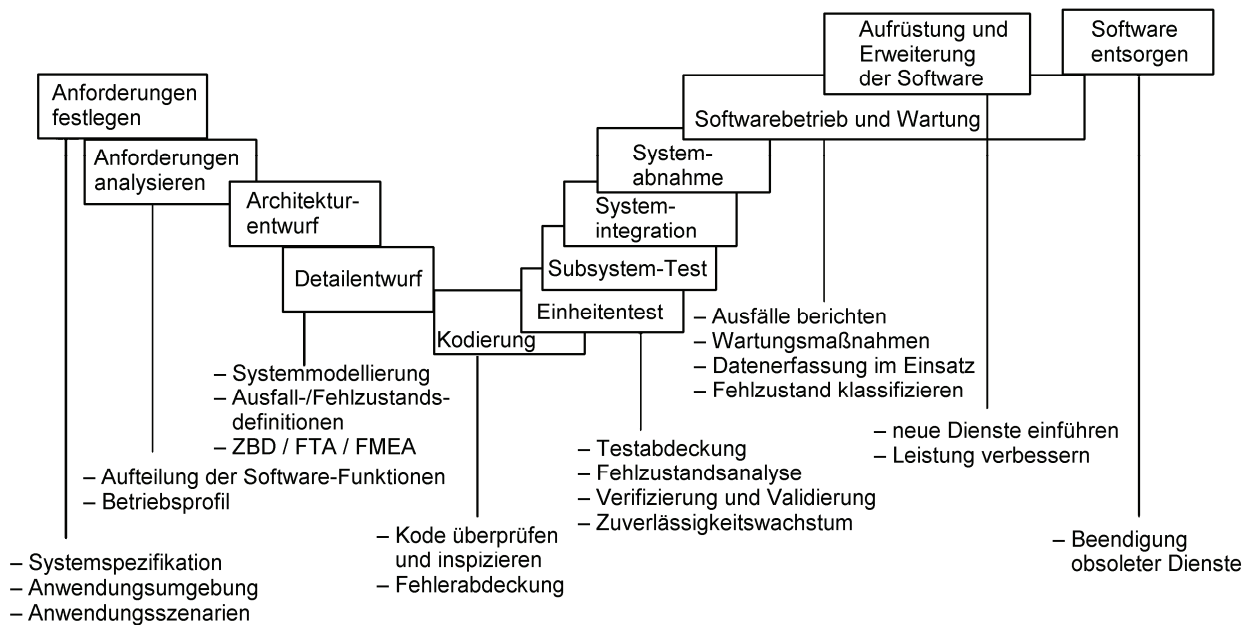


Bild 1 – Tätigkeiten im Softwarelebenszyklus

5.4 Merkmale der Softwarezuverlässigkeit

Merkmale der Softwarezuverlässigkeit sind solche speziell auf die Zuverlässigkeit bezogene Leistungsmerkmale und Leistungskenngrößen, die dem Softwaresystem durch Entwurf und Konstruktion inhärent sind. Beim Zuverlässigkeitsmanagement sollten die besonderen Merkmale der Softwarezuverlässigkeit beachtet und diese bei Systementwurf und Konstruktion berücksichtigt werden, damit gemeinsame Ziele für die Systemzuverlässigkeit (Hardware und Software) gesetzt und diese dann auch erreicht werden können.

Zu den wichtigsten Merkmalen der Softwarezuverlässigkeit gehören:

- **Verfügbarkeit:** bereit sein zur Ausführung der Software;
- **Funktionsfähigkeit:** die Softwaredienstleistung kontinuierlich erbringen;
- **Wartbarkeit:** leichte Änderbarkeit, Aufrüstung und Erweiterung der Software;
- **Wiederherstellbarkeit:** nach einem Ausfall (Absturz) sollte die Software von alleine oder durch externe Maßnahmen wieder lauffähig sein;
- **Integrität (Unversehrtheit):** Korrektheit der von der Software erzeugten Daten und Informationen.

Weitere Merkmale der Softwarezuverlässigkeit sind anwendungsspezifisch. Dazu gehören neben anderen:

- **Sicherheit (en: security):** Schutz vor böartigem Eindringen in die Softwareanwendung;
- **Sicherheit (en: safety):** Verhinderung von Schäden bei Anwendung von Software;
- **Funktionstüchtigkeit:** robuster, fehlertoleranter und unterbrechungsfreier Betrieb;
- **Wiederverwendbarkeit:** Einsatz einer existierenden Software in anderen Anwendungen;
- **Wartbarkeit:** Bereitstellung und Management von Mitteln zur Durchführung von Softwarewartungsaufgaben;
- **Portabilität:** plattformübergreifende Anwendung der Software.

Diese Merkmale der Softwarezuverlässigkeit bilden die Grundlage für den Softwaresystementwurf und die Softwareanwendung.

5.5 Softwareentwicklungsumgebung

Die Zielsetzung des Zuverlässigkeitsmanagements ist die Schaffung einer gut ausgewogenen und kreativen Entwicklungsumgebung im Rahmen der Projektbudgetmittel, des Zeitplans und der Liefertermine. Organisationen im Zusammenhang mit Softwareentwicklung und Softwaredienstleistungen sind auf die Anwendung durch die Nutzer orientiert. Softwareprojekte müssen zugeschnitten werden, damit die Zuweisung der verfügbaren Ressourcen gestaltet und geeignete Entwurfsoptionen für eine erfolgreiche Implementierung gesucht und gefunden werden können. Die Wahl der anwendbaren Prozesse zur technischen Realisierung der Zuverlässigkeit in ein gegebenes Softwaresystem erfolgt durch den Prozess des Projektzuschnitts für ein erfolgreiches Zuverlässigkeitsmanagement. Die empfohlenen Zuschneideprozesse sind in 7.2 beschrieben. Dabei sollten mögliche Gelegenheiten für die Vergabe der Entwicklungsarbeiten, für eine Wiederverwendung der Software und die Verwendung handelsüblicher Softwareprodukte für die Systemintegration ausgelotet werden.

Die Softwareentwicklungsumgebung ist auf einen organisierten Prozess angewiesen, der eine gute Entwurfspraxis hinsichtlich einer fehlerlosen Codegenerierung fördert, der Fehler bei der Festlegung der Anforderungen weitgehend verringert und der die Software vor deren Freigabe validiert. Beim Softwaremanagement wird häufig auf der Grundlage eines Fähigkeit-Reife-Modells für Infrastrukturentwicklung [8] vorgegangen. Dieses ist der formalen Implementierung der *Capability Maturity Model Integration* [9] nach Anhang C für Softwareprozessmanagement ähnlich. Softwareentwicklung ist ein technischer Prozess, der eingeführten Disziplinen bei der technischen Erarbeitung der Software und erprobten Anwendungsleitlinien folgt. Die Softwareentwicklungsumgebung und Praxisgrundregeln sollten in die Führungsgrundsätze der Organisation mit aufgenommen und dementsprechend Auftrag und Ziele für das Erreichen der Zuverlässigkeit formuliert werden.

Bei der Softwareentwicklung werden häufig rechnergestützte Entwicklungswerkzeuge (CASE) eingesetzt. Ein wirkungsvolles automatisiertes System bietet die rechnerische Genauigkeit, ermöglicht die Rückverfolgbarkeit von Daten und stellt ein Konfigurationsmanagement und Mittel für das Erfassen der erforderlichen Eingabegrößen in die automatisierten Modelle zur Verfügung. Die meisten Systeme zum Erfassen und zum Berichten von im Einsatz erfolgten Ausfällen, deren Analyse und Korrekturmaßnahmen werden aus den gleichen Gründen automatisiert. Vorhandene Erfahrungsdaten bezüglich der Softwareprodukte und Softwaredienstleistungen sind ein unentbehrliches und wertvolles Gut.

5.6 Erarbeitung der Anforderungen an die Software und der Zuverlässigkeitsziele

Anforderungen an die Software sollten für alle Stufen des Softwarelebenszyklus erarbeitet werden. Zur Implementierung in den jeweiligen Stufen sollten geeignete Zuverlässigkeitstätigkeiten ermittelt werden. Es ist wichtig, die Ausführung der einzelnen Zuverlässigkeitstätigkeiten zeitlich richtig einzuplanen. Zuverlässigkeitsanwendungen sind zeitabhängig und haben erhebliche Auswirkung auf die Lebenszykluskosten des Systems [10]. Ein geeigneter Projektzuschnitt ist für Entwurfsabwägungen und die Überwindung von Hemmnissen wesentlich. Die Anforderungen an die Software und die Zuverlässigkeitsziele sollten Teil der gesamten Softwareproduktspezifikationen sein. Die Strategie für die Implementierung der Softwarezuverlässigkeit wird in 5.8 beschrieben. Die Methodik für die technische Realisierung der Zuverlässigkeit in Softwaremodulen und zum Entwurf der Softwaresystemarchitektur wird in Abschnitt 6 behandelt. Der Prozess des Zuschneidens ist in 7.2 beschrieben.

Die Zuverlässigkeitstätigkeiten bezüglich der Anforderungen an die Software sind anwendungsspezifisch. Sie ergeben sich aus dem Softwareentwurf und den Implementierungsbedürfnissen, die für die Erfüllung der geforderten Systemfunktionen und Leistungen benötigt werden. Systematische Vorgehensweisen bei der Implementierung der jeweiligen Zuverlässigkeitstätigkeiten in allen Stufen des Softwarelebenszyklus sichern das Erreichen der Zuverlässigkeitsziele. Aus wichtigen Zuverlässigkeitsmerkmalen und den zugehörigen quantitativen Maßgrößen werden spezifische Zuverlässigkeitsziele abgeleitet. Anforderungen an die Softwarezuverlässigkeit können projektweise formuliert werden, dazu können die in Anhang B enthaltenen Hinweise verwendet werden.

Die Einflüsse auf kombinierte Hardware/Softwaresysteme sind in den Systemzuverlässigkeitsspezifikationen beschrieben [11]. Die nachfolgenden Einflussfaktoren, die sich während der Softwareentwicklung auf das Erreichen der Zuverlässigkeit auswirken, sollten beachtet werden:

- die Entwicklungskultur der Organisation, ihr Fähigkeits-Reifeprozess und ihre Erfahrung mit Softwareentwurf, Entwicklung und Implementierung (siehe Anhang C);
- Verständnis für die Anwendungsumgebungen, für die Anwenderbedürfnisse und sich ändernde dynamische Marktentwicklungen hin zu neuen Plattformen oder hin zur Entwicklung und praktischen Implementierung von Leistungsmerkmalen;
- Dokumentationsprozesse wie Ausfallberichterstattung, Datenerfassung, Softwarekonfigurationsmanagement zur Überwachung der Softwareversionen und Pflege aufgezeichneter Erfahrungsberichte;
- Anwendung von Softwareentwurfsregeln zur Fehlervermeidung durch Überwachung und Lenkung des Entwurfsprozesses zur Optimierung des Softwareverhaltens angesichts komplexer Software, Programme und Funktionen;
- wirkungsvoller Einsatz geeigneter Softwaremethoden und Werkzeuge wie etwa strukturierter Entwurf, Fehlertoleranz, Entwicklungsbewertung [12] und Softwarefehlermanagement zur Förderung des Zuverlässigkeitswachstums;
- Wahl geeigneter höherer Programmiersprachen, die besser für eine gezielte und strukturierte Softwareentwicklung geeignet sind;
- anerkannte Vorgaben zur Qualifizierung und Messung von Systemzuverlässigkeitskenngrößen.

5.7 Klassifizierung von Softwarefehlern

Softwarefehler können klassifiziert werden als Spezifikationsfehler, Entwurfsfehler, Programmierfehler, durch den Compiler verursachte Fehler oder Fehler, die bei der Softwarewartung induziert wurden.

Die Klassifizierung von Softwarefehlern ist ein Mittel zur Erfassung und Gruppierung relevanter Informationen über Softwarefehler. Der Klassifizierungsprozess hilft den Softwareentwicklern, unübliche Fehlermuster zu entdecken und Korrekturen einzuleiten. Der Zweck dieser Bemühungen ist es, das wiederholte Auftreten ähnlicher Fehlerklassen zu eliminieren.

Die Methode der *Orthogonal Defect Classification* (ODC) [13] ist eine bei der Softwareentwicklung verwendete Methode zur Analyse von Softwarefehlerdaten^{N1)}. Die ODC-Methode behandelt die kausalen Wirkungen von Qualitätsproblemen im Softwareentwurf und im Code in einer prozeduralen Sprachumgebung. Ein Defekt ist die Nichterfüllung einer Anforderung hinsichtlich der beabsichtigten oder spezifizierten Verwendung der Software. In diesem Kontext zeigt ein Fehler aufgrund der Unfähigkeit der Software, ihre geforderte Funktionen auszuführen, in dem ODC-Schema die Merkmale von Defektattributen. Defektattribute sind die Signatur eines Defekts, der relevante Information bezüglich eines Softwarefehlers enthält. Bei der ODC-Methode werden die Softwarefehlerinformation der Defektattribute zu Analyse- und Modellierungszwecken erfasst. Die Analyse der ODC-Daten ergibt eine wertvolle Diagnosemethode zur Beurteilung der Reife des Softwareprodukts in verschiedenen Stufen des Softwarelebenszyklus. Die ODC-Methode kann ebenfalls dazu verwendet werden, den Prozess durch eine Analyse der Arten der Trigger (Auslöser) zu beurteilen, um spezifische technische Bedürfnisse erkennen zu können, damit die fehlenden Trigger stimuliert werden. Die kausale Analyse der Fehlerdaten (Defektdaten) stellt einen Weg zur Verringerung von Softwarefehlern und zur Zuverlässigkeitsverbesserung dar.

Die ODC-Defektattribute werden wie folgt klassifiziert: *Activity* (Aktivität), *Trigger* (Auslöser), *Target* (Ziel), *Defect Type* (Defekttyp), *Defect Qualifier* (Defektmerkmal), *Source* (Quelle), *Impact* (Auswirkung) und *Age* (Alter). Sie werden üblicherweise während der Softwareentwicklung erfasst und analysiert und dienen der Entwurfsverbesserung. Die Information über Defekte ist bei zwei besonderen Zeitpunkten verfügbar. Wenn ein Fehler gefunden wird, sind die Umstände der Offenlegung des Fehlers und der wahrscheinlichen Auswirkung auf den Anwender im Allgemeinen bekannt. Wenn ein Fehler geschlossen wird, nachdem der Fehler behoben wurde, sind die genaue Art und der Umfang der Fehlerbehebung (Fix) bekannt. In den ODC-Kategorien wird die Semantik eines Fehlers (Defekt) aus diesen beiden Perspektiven erfasst. Durch das Definieren der *Aktivitäten* im Entwicklungsprozess und deren Abbildung auf die ODC-Auslöser liefert die ODC-Methode der Softwareentwicklungsorganisation wertvolle Einsichten und maßgeschneiderte Informationen über Fehler (Defekte).

^{N1)} Nationale Fußnote: Der hier gebräuchliche englische Begriff „defect“ wird in dieser Norm mit „Fehler“ und auch mit „Defekt“ übersetzt.

Der Satz von Defektattributen wird zusammengefasst a) wenn ein Fehler gefunden wird, der als *opener section (Öffner)* bekannt ist, und b), wenn ein Fehler behoben wird, der als *closer section (Schließer)* bekannt ist. Die ODC-Methode ist in reifen Softwareorganisationen am nützlichsten, in denen zur Verbesserung des Softwareproduktes üblicherweise umfangreiche Daten erfasst und analysiert werden.

Anhang D enthält eine Übersicht der Klassifizierung von Softwarefehlerattributen.

5.8 Strategie zur Realisierung der Softwarezuverlässigkeit

5.8.1 Vermeidung von Softwarefehlern

Softwarecodes werden generiert, um ein Softwareprodukt zu erzeugen. Ein während des Softwareentwurfs oder der Kodierung geschehener Irrtum könnte später im Betrieb nach seiner Auslösung als ein Softwarefehler auftauchen, der zu einem Systemausfall führen könnte. Da Softwarefehler die Hauptursache für Softwareausfälle sind, sollte verhindert werden, dass Softwarefehler während der Entwicklung in den Entwurf eingeschleust werden. Dazu eignen sich Codeüberprüfung sowie die Beseitigung von Restfehlern, die beim Testen nicht erkannt worden waren; dies ist eine übliche Vorgehensweise zur Verringerung von Fehlerproblemen im Softwarelebenszyklus. Zur empfohlenen Strategie der Softwarefehlervermeidung gehören Fehlerverhütung und Fehlerbeseitigung.

a) Fehlerverhütung

- Vorgaben für die Fehlerverhütung in den Softwareentwicklungsdisziplinen anlegen;
- die Überprüfung von Anforderungsspezifikationen veranlassen;
- frühzeitiges Zusammenspiel mit den Anwendern veranlassen und Verfeinerung der Softwareanforderungen durchführen;
- formale Methoden einführen, wo anwendbar und praktikabel;
- systematische Techniken für die Wiederverwendung von Software implementieren und deren Anwendung sichern.

b) Fehlerbeseitigung

- erkenne und beseitige das Vorhandensein von Softwarefehlern durch Testen;
- formale Inspektion durchführen, um Fehler zu finden, diese korrigieren und die Korrekturen überprüfen;
- während des Betriebs der Software korrigierende und perfektionierende Wartungsmaßnahmen durchführen.

5.8.2 Beherrschung von Softwarefehlern

Softwarefehler sind schwierig zu erkennen, und zu deren Beseitigung werden unterschiedliche Mittel eingesetzt, so etwa rigoroses Testen der Software und Softwareinspektion. Das ausgiebige Testen der Software wird häufig aus Sicht des Projektmanagements begrenzt, weil dies als zu lange und zu teuer angesehen wird. Basiert die Zuverlässigkeitssicherung alleine auf dem Testen, so garantiert dies keine vollständige Beseitigung von Softwarefehlern. Zur Beherrschung von Softwarefehlern verwendet man Fehlertoleranz- und Vorhersagemethoden und versucht so, die Manifestierung von nach der Einsatzfreigabe der Software etwa noch vorhandenen latenten Fehlern oder Bugs zu verringern. Zur empfohlenen Strategie der Beherrschung von Softwarefehlern gehören Fehlertoleranz und Fehler-/Ausfallvorhersage.

a) Fehlertoleranz

- eine Methodik zur Fehlerbegrenzung, Fehlererkennung und die Erholung nach Fehlern erstellen;
- Vielfalt im Softwareentwurf und Rückfallschemata implementieren;
- Multiversions-Programmiertechniken einführen;
- sich selbst prüfende Programmiertechniken implementieren.

- b) Fehler-/Ausfallvorhersage
 - die Beziehungen zwischen Fehler und Ausfall in der Betriebsumgebung ermitteln;
 - ein System zur Erfassung relevanter Daten einführen;
 - auf Zuverlässigkeitswachstum, wo angebracht, testen;
 - geeignete Zuverlässigkeitsmodelle zur Fehler-/Ausfallabschätzung entwickeln und implementieren;
 - Vorhersagetechniken für vorausschauende Freigabe (Release) von Softwareversionen verfeinern.

6 Methodik zur Verwirklichung der Softwarezuverlässigkeit

6.1 Vorgehensweisen bei der Softwareentwicklung zur Erzielung der Zuverlässigkeit

Die Fähigkeitsreife (en: capability maturity) bei der Softwareentwicklung spiegelt die Fähigkeit einer Organisation wider, Software mit Beständigkeit zu entwickeln und zuverlässige Produkte für beabsichtigte Anwendungen zu liefern. Die folgenden Techniken zur Vermeidung und zur Überwachung und Beherrschung von Fehlern werden, soweit anwendbar, zur Aufnahme in die Softwareentwicklung empfohlen:

- a) Methoden standardisieren für den Architekturentwurf auf hoher Ebene, Detailentwurf, Kodierung und Testen und Dokumentation zur leichteren Kommunikation und Fehlervermeidung;
- b) Modulare Entwürfe entwickeln für Softwareeinheiten und Subsysteme mit gut definierten Softwarefunktionen und Schnittstellen durch die Schaffung einfacher, getrennter und unabhängiger Softwareeinheiten zur Erleichterung von Wechselwirkungen im Entwurf, für die Wartung, die Verfolgbarkeit von Fehlaussagen und Irrtümern und die Abschwächung von Fehlern und Beseitigung von Bugs;
- c) Entwurfsmuster verwenden, die allgemein wiederverwendbare Lösungen auf Basis gut getesteter Software sind, als Vorlagen zur Lösung von Softwareentwurfsproblemen zur Beschleunigung des Entwicklungsprozesses;
- d) soweit zutreffend, formale Entwurfsmethoden einrichten zur Lenkung und Dokumentation von Softwareentwurf und Entwicklungsprozess;
- e) verwende Softwarezuverlässigkeitstechniken [14] zur Beurteilung der Softwarezuverlässigkeit und deren Verbesserung [15];
- f) Software wiederverwenden aus Softwarebibliotheken von gut ausgetesteten Softwareeinheiten und Subsystemen für ähnliche Anwendungen und Betriebsprofile zur Verringerung von Entwicklungskosten und Zeit und zur Minimierung von in den Entwurf neu eingeführten Fehlern;
- g) entwickle Regressionstestmethoden zur Sicherstellung der Funktionalität existierender Software bei neu eingeführter Funktionalität oder wenn eine Fehlerbeseitigung durchgeführt worden ist;
- h) Testen von Softwareeinheiten und Subsystemen zur Verifizierung von Funktionen in niederen Entwurfs-ebenen und zur Validierung des Systemverhaltens bezüglich der Integration des Architekturentwurfs auf hoher Ebene mit dem Ziel der progressiven Beseitigung von Bugs, um eine Fehlerausbreitung zu verhindern;
- i) Inspektionen und Überprüfungen durchführen zu den Anforderungen an den Softwareentwurf, von Softwarecodes, Benutzerhandbüchern, Schulungsunterlagen und Testdokumenten mit dem Ziel, so viele (unbeabsichtigte) wie möglich Fehlgriffe zu beseitigen; dabei sollten, soweit praktikabel, verschiedene Überprüfungsteams zum Vergleich der Ergebnisse eingesetzt werden;
- j) Änderungsüberwachung zur Verringerung des Auftretens von Fehlern wie beispielsweise Versions- und Änderungsüberwachungsprozesse im Softwarekonfigurationsmanagement;
- k) Ursache-Wirkung-Probleme analysieren und geeignete Korrekturmaßnahmen zur nachhaltigen Softwareverbesserung implementieren;
- l) ein Datenerfassungssystem für die Erfassung und Speicherung von Softwarefehlern und früheren Leistungsdaten in einer Wissensbasis einrichten.

6.2 Metriken der Softwarezuverlässigkeit und Datenerfassung

Softwarezuverlässigkeitsmaßgrößen, auch Metriken genannt, sind die Zuverlässigkeitskenngrößen eines Softwaresystems. Durch die Messung der Metriken erhält man eine quantitative Skale und Methode zur Bestimmung des Wertes einer spezifischen Kenngröße im Zusammenhang mit dem Softwaresystem. Diese in der Industrie üblichen Metriken erhält man entweder durch direkte Messung oder durch Rückschlüsse. Sie werden zur Messung der Leistung eines Softwaresystems verwendet. Die folgenden Softwaremetriken sind de facto Industriestandards. Sie sollten soweit geeignet zur Beurteilung der Softwarezuverlässigkeit herangezogen werden.

- a) *Verfügbarkeit* ist ein Maß für die Klardauer über der Gesamtdauer des Systembetriebs.
- b) *Ausfallhäufigkeit* ist ein Maß für die Anzahl der Ausfälle über der Dauer des Systembetriebs.
- c) *Dauer bis zum Ausfall* ist ein Maß für die ausfallfreie Zeit.
- d) *Dauer bis zur Wiederherstellung* ist ein Maß für die Dauer einer Wiederherstellung eines Systems aus einem ausgefallenen Zustand heraus zurück zum normalen Betriebszustand.
- e) *Fehlerdichte* ist ein Maß für die Anzahl der Fehler je tausend Zeilen Quelltext (KSLOC) oder per Funktionspunkt und dient der Beurteilung der Systemzuverlässigkeit.
- f) *Funktionspunkt* ist ein Maß für den funktionalen Umfang der Anwendungssoftware zur Planung eines Softwareprojektes anhand der „function point analysis“-Methode [16].
- g) *Codeüberdeckung* (en: *Code coverage*) ist ein Maß für den Grad, zu dem der Quellcode und die logischen Zweige eines Softwareprogramms systematisch getestet worden sind; *Codeüberdeckung* ist ein Indikator für die Gründlichkeit der Softwaretests, er wird zur Darstellung des *Fehlerüberdeckungsgrades* (en: *fault coverage*) verwendet, der den Anteil der bei der versuchsweisen Ausführung (Test) des Codes erkannten Fehler angibt.
- h) *Grad der Beseitigung von Fehlern* ist ein Maß für die Anzahl der Fehler, die in einem Softwareprodukt erkannt und beseitigt wurden, bezogen auf eine definierte Zeitspanne oder die Dauer einer Softwareausführung; der *Grad der Beseitigung von Fehlern* wird beim Zuverlässigkeitswachstum zur Feststellung des Wachstumstrends verwendet.
- i) *Restfehler in der Software* sind ein Maß für die geschätzte Anzahl der in dem Softwareprodukt nach dem Testen auf Beseitigung von Softwarefehlern noch verbliebenen Bugs.
- j) *Zeit bis zur Freigabe der Software* ist ein Maß für die geschätzte Zeit bis zur Freigabe des Softwareprodukts bzw. für den Freigabezeitplan. Bei dieser Schätzung bedient man sich etablierter Kriterien bezüglich eines akzeptablen Niveaus noch verbliebener Bugs im Softwareprodukt, deren Angabe für das Softwareprojektmanagement dienlich ist.
- k) *Softwarekomplexität* ist ein Maß für den Schwierigkeitsgrad des Entwurfs und der Implementierung einer Softwarefunktion oder eines Softwaresystems; weitere Komplexitätsmetriken auf der Grundlage des Komplexitätskonzepts sind u. a. Programmkomplexität, funktionale Komplexität, operationelle Komplexität; auf die Komplexität bezogene Metriken dienen als Eingabe in die Zuverlässigkeitsbeurteilung und die Vorhersagemodelle.

Es gibt zahlreiche Maßgrößen für verschiedene Anwendungen während des Softwarelebenszyklus. Softwaremetriken können zur leichten Datenerfassung in drei allgemeine Kategorien gruppiert werden.

- 1) *Fehlerdatenmetriken* erfassen die zu Softwareproblemen berichteten Daten, zur Messung der Auswirkung der Fehlzustände sowie zur Messung der Effizienz des Berichtssystems zur Verbesserung der Softwarewartung.
- 2) *Produktdatenmetriken* erfassen Informationen über das Softwareprodukt durch Kategorisierung nach Größe (Umfang), Funktionalität, Komplexität, Einsatzort und anderen Kenngrößen. Damit dienen diese Erfahrungsdaten als Eingabedaten für neue Produktentwicklungen. Diese Metriken liefern Erfahrungsberichte über das Verhalten und Daten und weitere Informationen zu verschiedenen Softwareproduktgruppen.
- 3) *Prozessdatenmetriken* erfassen Informationen über den Wiederherstellungsprozess der Software und die Bedingungen zum Zeitpunkt des Erkennens und der Beseitigung eines Fehlers, und sie dienen als Eingaben in Zuverlässigkeitsvorhersagemodelle.