

DIN IEC 62628

ICS 03.120.10; 29.020

Einsprüche bis 2010-05-31

Entwurf**Leitlinien zu Softwareaspekten der Zuverlässigkeit
(IEC 56/1349A/CD:2009)**

Guidance on Software Aspects of Dependability (IEC 56/1349A/CD:2009)

Anwendungswarnvermerk

Dieser Norm-Entwurf mit Erscheinungsdatum 2010-03-08 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

Weil die beabsichtigte Norm von der vorliegenden Fassung abweichen kann, ist die Anwendung dieses Entwurfes besonders zu vereinbaren.

Stellungnahmen werden erbeten

- vorzugsweise als Datei per E-Mail an dke@din.de in Form einer Tabelle. Die Vorlage dieser Tabelle kann im Internet unter www.dke.de/stellungnahme abgerufen werden;
- oder in Papierform an die DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (Hausanschrift: Stresemannallee 15, 60596 Frankfurt am Main).

Die Empfänger dieses Norm-Entwurfs werden gebeten, mit ihren Kommentaren jegliche relevante Patentrechte, die sie kennen, mitzuteilen und unterstützende Dokumentationen zur Verfügung zu stellen.

Gesamtumfang 107 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Inhalt

	Seite
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.....	8
4.1 Software und Softwaresysteme.....	8
4.2 Hinweise zur Anwendung dieser Norm	9
5 Managementeinfluss auf die Softwarezuverlässigkeit	10
5.1 Systemlebenszyklus- und Softwarelebenszyklus-Prozesse	10
5.2 Merkmale der Softwarezuverlässigkeit.....	12
5.3 Softwareentwicklungsumgebung.....	13
5.4 Erarbeitung der Anforderungen an die Software und der Zuverlässigkeitsziele	13
5.5 Fehlaussagen, Irrtümer, Fehler und Ausfälle von Software verstehen	14
5.6 Klassifizierung von Softwarefehlern	14
5.7 Strategie zur Implementierung der Softwarezuverlässigkeit.....	15
6 Methodik zur Verwirklichung der Softwarezuverlässigkeit	15
6.1 Vorgehensweisen bei der Softwareentwicklung zur Erzielung der Zuverlässigkeit	15
6.2 Softwarezuverlässigkeitsmaßgrößen und Datenerfassung.....	16
6.3 Beurteilung der Softwarezuverlässigkeit	17
6.4 Verbesserung der Softwarezuverlässigkeit.....	26
7 Softwaresicherung	30
7.1 Überblick über die Softwaresicherung.....	30
7.2 Technologische Einflüsse auf die Softwaresicherung.....	30
7.3 Optimale Verfahren der Softwaresicherung	31
Anhang A (informativ) Softwareorganisationen und Kategorisierung der Software und Anwendungen.....	32
A.1 Softwareorganisationen.....	32
A.2 Kategorisierung von Software	32
A.2.1 Softwarekategorien.....	32
A.2.2 Merkmale	32
A.2.3 Umgebung	33
A.2.4 Daten	33
A.3 Softwareanwendungen.....	33
Anhang B (informativ) Zusammenfassung von Anforderungen an Software und zugehörige Zuverlässigkeitstätigkeiten	34
B.1 Festlegung der Anforderungen.....	34

	Seite
B.2 Analyse der Anforderungen	34
B.3 Architekturentwurf	34
B.4 Detailentwurf	35
B.5 Realisierung	35
B.6 Integration	36
B.7 Abnahme	36
B.8 Betrieb und Wartung	36
B.9 Aktualisierung und Erweiterung der Software	37
B.10 Rückzug	37
Anhang C (informativ) Zusammenfassung der Klassifizierung von Softwarefehlerattributen	38
Anhang D (informativ) Beispiele für Softwaremaßgrößen aus Datensammlungen	42
Anhang E (informativ) Beispiel für kombinierte Hardware/Software-Zuverlässigkeitsfunktionen	45
Anhang F (informativ) „Capability Maturity Model Integration“-Prozess	47
Anhang G (informativ) Zusammenstellung von Maßgrößen für Modelle der Funktionsfähigkeit von Software	50
Anhang H (informativ) Auswahl und Anwendung von Modellen für Softwarefunktionsfähigkeit	51
Literaturhinweise	54
 Bild 1 – Hinweise zur Anwendung dieser Norm	 10
Bild 2 – Tätigkeiten im Softwarelebenszyklus	12
Bild 3 – Beurteilung der Softwarezuverlässigkeit mit Schwerpunkt auf der Funktionsfähigkeit	18
Bild E.1 – Blockdiagramm eines Steuerungssystems für funktionale Sicherheit	45
 Tabelle C.1 – Klassifizierung von Softwarefehlerattributen nach der Fehlerentdeckung	 38
Tabelle C.2 – Klassifizierung von Softwarefehlerattributen bezüglich Fehlerbeseitigung	38
Tabelle C.3 – Bei Entwurfsbewertung/Codeinspektion bewirkte Auslöser	39
Tabelle C.4 – Beim Einheitentest bewirkte Auslöser	40
Tabelle C.5 – Beim Funktionstest bewirkte Auslöser	40
Tabelle C.6 – Beim Systemtest bewirkte Auslöser	41
Tabelle F.1 – Vergleich von Fähigkeitsgraden und Reifegraden	47
Tabelle H.1 – Beispiele von Modellen Softwarefunktionsfähigkeit	51

Nationales Vorwort

Das internationale Dokument IEC 56/1349A/CD:2009 „Guidance on Software Aspects of Dependability“ (CD, en: Committee Draft) ist unverändert in diesen Norm-Entwurf übernommen worden. Dieser Norm-Entwurf enthält eine noch nicht autorisierte deutsche Übersetzung.

Um Zweifelsfälle in der Übersetzung auszuschließen, ist die englische Originalfassung des CD entsprechend der diesbezüglich durch die IEC erteilten Erlaubnis beigefügt. Die Nutzungsbedingungen für den deutschen Text des Norm-Entwurfes gelten gleichermaßen auch für den englischen IEC-Text.

Das internationale Dokument wurde vom IEC/TC 56 „Dependability“ der Internationalen Elektrotechnischen Kommission (IEC) erarbeitet und den nationalen Komitees zur Stellungnahme vorgelegt.

Die IEC und das Europäische Komitee für Elektrotechnische Normung (CENELEC) haben vereinbart, dass ein auf IEC-Ebene erarbeiteter Entwurf für eine Internationale Norm zeitgleich (parallel) bei IEC und CENELEC zur Umfrage (CDV-Stadium) und Abstimmung als FDIS (en: Final Draft International Standard) bzw. Schluss-Entwurf für eine Europäische Norm gestellt wird, um eine Beschleunigung und Straffung der Normungsarbeit zu erreichen. Dokumente, die bei CENELEC als Europäische Norm angenommen und ratifiziert werden, sind unverändert als Deutsche Normen zu übernehmen.

Da der Abstimmungszeitraum für einen FDIS bzw. Schluss-Entwurf prEN nur 2 Monate beträgt, und dann keine sachlichen Stellungnahmen mehr abgegeben werden können, sondern nur noch eine „JA/NEIN“-Entscheidung möglich ist, wobei eine „NEIN“-Entscheidung fundiert begründet werden muss, wird bereits der CD als DIN-Norm-Entwurf veröffentlicht, um die Stellungnahmen aus der Öffentlichkeit frühzeitig berücksichtigen zu können.

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

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

Einleitung

Software wird in heutigen elektrotechnischen Produkten und Systemen in breitem Maße angewendet. Beispiele sind Softwareanwendungen in programmierbaren Steuerungseinrichtungen, in Rechnersystemen und in Kommunikationsnetzen. Im Laufe der Jahre sind viele Normen und Standards zu Softwaretechnik, Softwareprozesslenkung, Softwarequalität und Zuverlässigkeitssicherung entwickelt worden, aber es gibt nur wenige Normen und Standards, die Softwareprobleme aus Sicht der Zuverlässigkeit behandeln.

Zuverlässigkeit ist die Fähigkeit eines Systems, geforderte Ziele (Funktionen) 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 über das Internet und ermöglichen so Plattform-, Dienstanbieter-, und Domäne- übergreifende 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. Softwaredesign ist zur primären Funktion bei der Datenverarbeitung, der Überwachung und dem Schutz Sicherheits-relevanter Güter und Belange und bei Kommunikationsbeziehungen in Netzdiensten avanciert. Dieser Paradigmenwechsel hat die globale Geschäftswelt dazu gebracht, einer Situation zu vertrauen, die in ganz besonderem Maße auf Softwaresystemen beruht und auf die wiederum ihre geschäftlichen Vorgänge aufbauen. Softwarezuverlässigkeit spielt eine dominante Rolle für eine erfolgreiche Systemleistung und eine unversehrte Datenübertragung.

In dieser Norm werden die gegenwärtig in der Industrie üblichen Vorgehensweisen vorgestellt und es wird eine Methodik beschrieben, wie die Softwarezuverlässigkeit leichter erreicht werden kann. Es wird ein Rahmen aufgezeigt, wie die Software im Hinblick auf die Zuverlässigkeit gestaltet werden sollte, und es werden relevante technische Prozesse beschrieben, wie Softwarezuverlässigkeit in Systemen technisch realisiert werden kann. Die Evolution der Softwaretechnik und die schnelle Adaptation von Softwareanwendungen in der Industrie haben das Bedürfnis für eine Praxis-orientierte Norm zur Softwarezuverlässigkeit für weltweite Geschäftsbeziehungen geschaffen. Ein Überblick über die Softwaresicherung wird präsentiert. 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 Systeme implementiert werden. Für kritische Aufgaben und bei Sicherheits-relevanten Anwendungen sind zusätzliche Anforderungen von Nöten. Industrie-spezifische Belange der Softwarequalifizierung mit Blick auf die Konformität von Zuverlässigkeit und Qualität werden in dieser Norm nicht behandelt.

Diese Norm behandelt keine Produkte mit eingebundener Software (en: embedded software products). Solches sind Geräte, bei denen die Software in Hardwarechips (elektronische Bauelemente) eingebrannt ist und die so eine ganz bestimmte, festgelegte Funktion erbringen. Beispiele sind anwendungsspezifische integrierte Schaltungen und durch Mikroprozessor gesteuerte Geräte. Diese Produkte werden häufig als Teil der physischen Hardwaremerkmale entwickelt, wie beispielsweise Echtzeitanwendungen in Mobiltelefonen. Obwohl die in dieser Norm beschriebenen allgemeinen Zuverlässigkeitsgrundsätze und Vorgehensweisen beim Entwurf und der Anwendung von Produkten mit eingebetteter Software verwendet werden dürfen, so sind doch für deren Entwicklung und Implementierung spezielle Anforderungen von Nöten.

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

1 Anwendungsbereich

Diese Norm behandelt die Auswirkung von Software und Software-relevanten Belangen auf die Zuverlässigkeit, und sie leitet dazu an, wie ein gefordertes Zuverlässigkeitsverhalten der Software abhängig von Führungsgrundsätzen, Entwicklungsprozessen und der Anwendungsumgebungen erreicht werden kann. Es wird ein allgemeines Rahmenwerk der Anforderungen an die Software im Hinblick auf Zuverlässigkeit ausgearbeitet, es wird ein Prozess für die Software im Hinblick auf Zuverlässigkeit 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 Zuverlässigkeitsmaßgrößen 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 das gewünschte Zuverlässigkeitsverhalten von Software und Software-intensiven Systemen zu erreichen.

2 Normative Verweisungen

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

IEC 60050-191, *International Electrotechnical Vocabulary – Chapter 191 Dependability*

IEC 60300-3-15, *Dependability management – Part 3-15: Guidance to engineering of system dependability*

3 Begriffe und Abkürzungen

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC 60050(191) sowie nachfolgende Begriffe.

3.1.1

Software

Programme, Verfahren, Regeln, Dokumentation und Daten eines Informationen verarbeitenden Systems [191-41-07]^{N1)}

ANMERKUNG Im Zusammenhang mit dieser Norm bezieht sich Software auf Rechnerprogrammcode, Daten und Anweisungen.

3.1.2

Softwareeinheit

Softwaremodul

niedrigste Ebene eines Softwareelements, das für sich alleine in Rechnerprogrammcode kompiliert werden kann, um eine Aufgabe auszuführen oder eine Tätigkeit zu starten, um das gewünschte Ergebnis einer Softwarefunktion zu erzielen

ANMERKUNG Die Begriffe Modul, Komponente und Einheit werden häufig je nach dem Textzusammenhang in austauschbarer Weise verwendet oder als Unterelemente voneinander auf unterschiedliche Weisen definiert. Die Beziehung dieser Begriffe ist noch nicht genormt.

^{N1)} Nationale Fußnote: Die Teile 41 bis 43 des IEV 191 sind noch nicht veröffentlicht.

3.1.3

Softwarekonfigurationsteil

Softwareeinheit, die als ein Konfigurationsteil etabliert wurde und die als eine einzelne Einheit im Konfigurationsmanagementprozess behandelt wird

ANMERKUNG Eine Softwarekonfigurationsteil kann aus einer oder mehreren Softwareeinheiten zur Durchführung einer Softwarefunktion bestehen.

3.1.4

Softwarefunktion

Durch ein(e) Software-Element, -Modul oder -Einheit ausgeführte elementare Operation

3.1.5

Softwaresystem

Software intensives System, bei dem Software das einzige zu entwickelnde oder modifizierende Element ist

ANMERKUNG 1 Ein Softwaresystem ist ein aus Software gebildetes System, das Teil eines Rechnersystems ist, welches aus einer Kombination von Hardware- und Softwareelementen gebildet ist. Beispiele von Softwaresystemen sind Webbrowser, Datenbankmanagementsysteme und Steuerungs- und Überwachungssysteme.

ANMERKUNG 2 Zu den wichtigsten Kategorien von Softwaresystemen gehören Anwendungssoftware (z. B. Software für Abrechnungs- und Informationsmanagement), Programmiersoftware (z. B. Software zur Leistungsanalyse und rechnergestützte Werkzeuge) und Systemsoftware (z. B. Software zur Steuerung und Überwachung von Rechnerhardware wie etwa Betriebssystemen).

3.1.6

Softwarezuverlässigkeit

Fähigkeit der Software, sich nach ihrer Integration in den Systembetrieb wie gefordert zu verhalten

3.1.7

Fehlaussage, Irrtum

Nichtübereinstimmung zwischen Rechenergebnissen, beobachteten oder gemessenen Werten oder Beschaffenheiten und den betreffenden wahren, spezifizierten oder theoretisch richtigen Werten oder Beschaffenheiten [191-05-24]

3.1.8

Softwarefehler

Beschaffenheit eines Softwareteils, in dem es unfähig sein kann, sich wie gefordert zu verhalten [191-44-03]

3.1.9

Softwareausfall

Ausfall, der eine Manifestation (das Sichtbarwerden oder Auftauchen) eines Softwarefehlers ist [191-43-21]

3.1.10

Code

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

ANMERKUNG 1 Quelltext (Quellcode) sind Anweisungen an einen Rechner und Datendefinitionen, die in solch einer Form ausgedrückt sind, die zur Eingabe in Assembler, Compiler oder andere Übersetzer geeignet sind.

ANMERKUNG 2 Kodierung ist der Prozess der Transformierung von Logik und Daten aus Entwurfsspezifikationen oder Beschreibungen in eine Programmiersprache.

3.1.11

(Rechner-)Programm

Kombination von Anweisungen an einen Rechner^{N2)} und Datendefinitionen, welche die Rechnerhardware in die Lage versetzen, Rechen- oder Steuerungsfunktionen auszuführen

ANMERKUNG Programmierung ist die generelle Tätigkeit bei der Softwareentwicklung.

3.2 Abkürzungen

CASE Rechner-gestützte Softwareentwicklung (en: Computer-Aided Software Engineering)

COTS handelsüblich (en: Commercial-Off-The-Shelf)

KSLOC tausend Zeilen Quelltext (en: Kilo-(thousand) Source Lines Of Code)

Q&A Fragen und Antworten

IT Informationstechnik

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 Rechnerprogrammcode, Daten und Anweisungen. Ein Softwaresystem besteht aus strukturierten Rechnerprogrammen und ausführbarem Code, die in eine physische Host-Rechnerhardware eingegliedert sind und zusammen den Systembetrieb ermöglichen und die geforderten Funktionen erbringen. Die Hierarchie des Softwaresystems kann als ein Gebilde aus Systemarchitektur und aus Subsystem-Softwareprogrammen und Softwareeinheiten der unteren Ebenen angesehen werden. Eine Einheit ist ein für sich prüfbares Softwaremodul, das beim Entwerfen eines Rechnersoftwareprogramms spezifiziert wurde. Zur Bildung einer Softwarefunktion können mehr als eine Einheit erforderlich sein. Das System umfasst sowohl Hardware- als auch Software-Elemente, die zur Erbringung nützlicher Funktionen zusammenwirken und so gemeinsam die geforderten Dienste leisten.

In einem kombinierten Hardware/Software-System tragen Softwareelemente 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 Analyse der Systemzuverlässigkeit 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 Zuweisung der Funktionsfähigkeit und zur Beurteilung der Zuverlässigkeit Software-basierter Systeme modelliert werden.

Aus Sicht der Zuverlässigkeit sind das Softwareverhalten und die Leistungsmerkmale anders als die bei Hardware beobachteten Merkmale. Softwarecode wird von Menschen geschaffen, diese Arbeiten unterliegen menschlichen Fehlhandlungen, welche durch die Entwicklungsumgebung und die Organisationskultur beeinflusst sind. Ausfalldaten von Hardware und Geräten sind wohl bekannt und dokumentiert, wogegen die Art und Ursache von Softwarefehlern und die Rückverfolgbarkeit von Fehlaussagen und Irrtümern beim 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 rückverfolgbaren Systemsoftwareausfällen garantieren keine vollständige Beseitigung der eigentlichen Ursachen von Softwareirrtümern. Im Gegensatz zum Austausch ausgefallener Hardwareeinheiten zur Wiederherstellung des normalen Systembetriebs kann eine Softwareberichtigung im Wartungsprozess manchmal neue Irrtümer einführen. Die menschlichen Aspekte der Zuverlässigkeit [16] spielen bei einer wirkungsvollen Softwareentwicklung und Implementierung eine entscheidende Rolle. Daraus leitet sich die strategische Notwendigkeit für eine auf die Software ausgerichtete Gestaltung der Zuverlässigkeit ab sowie für verbessernde Wartungsbemühungen im Softwarelebenszyklusprozess. Techniken zur Realisierung der Funktionsfähigkeit von Software [3] sind entwickelt worden und werden in der Industrie verbreitet angewendet.

^{N2)} Nationale Fußnote: umgangssprachlich auch Computer genannt.

det. So stehen Methodiken für die Vorhersage der Funktionsfähigkeit von Software, für die Modellierung und die Schätzung der Testdauer für eine Freigabe der Software zur Verfügung.

Softwarezuverlässigkeit ist die Fähigkeit der Software, sich nach ihrer Integration in den Systembetrieb wie gefordert zu verhalten und ihre Funktionen zu erfüllen. Der theoretische Rahmen bei den Betrachtungen zur Softwarezuverlässigkeit beruht auf den Grundsätzen der von der Hardware her bekannten Zuverlässigkeitstechniken. Einige dieser Techniken sind modifiziert worden, und neue Merkmale sind in die Methodik eingefügt worden, um den Softwareanwendungen zu genügen. Bei neu aufkommenden Technologien ist noch viel Forschungsarbeit zu leisten im Hinblick auf deren schnelle Anpassung an den industriellen Einsatz. Neue Vorgehensweisen müssen erkundet werden, um brauchbare Anwendungen zu validieren. In dieser Norm wird versucht, eine vereinheitlichte Vorgehensweise vorzustellen, bei der existierende Zuverlässigkeitstechniken und in der Industrie erfolgreich eingeführte Verfahren ermittelt und diese für die Gestaltung und Entwicklung der Software im Hinblick auf Zuverlässigkeit und die Implementierung von Softwaresystemen verwendet werden können. Die Zuverlässigkeitsmanagementsysteme [1, 2] beschreiben, wo und wie im Lebenszyklusprozess relevante Zuverlässigkeitstätigkeiten wirksam durchgeführt und umgesetzt werden können. Für kombinierte Hardware- und Softwaresysteme sollte beachtet werden, dass es beim Einsatz existierender Verfahren und Werkzeuge bei den Anwendungen auf die Systemzuverlässigkeit Einschränkungen gibt.

Zuverlässigkeit ist eine technische Disziplin mit dem Ziel der Sicherstellung der Unversehrtheit des Systems (Systemintegrität) und dem Erreichen der Funktionsfähigkeit im Systembetrieb. Softwaretätigkeiten während des Systemlebenszyklus sollten geplant, koordiniert und einher mit den entsprechenden Hardwaretätigkeiten gelenkt und geleitet werden. Zuverlässigkeitstätigkeiten sollten in die entsprechenden Projektpläne integriert und in die Aufgaben der Systemtechnik mit aufgenommen werden, um wirksame Systementwicklung, Realisierung, Implementierung, Betrieb und Wartung zu gewährleisten. Die Anleitung zur technischen Realisierung der Systemzuverlässigkeit in IEC 60300-3-15 gilt für diese Norm mit. Dort werden spezielle Belange der Softwarezuverlässigkeit aus Sicht des Lebenszyklusmanagements, der Implementierung technischer Prozesse und der Methodik der Softwarezuverlässigkeit behandelt.

4.2 Hinweise zur Anwendung dieser Norm

Das Erreichen eines zuverlässigen Softwaresystems wird neben anderen Merkmalen von Managementdisziplinen, Entwicklungsprozessen und den Anwendungsumgebungen beeinflusst. Bild 1 enthält einen Überblick über die einzelnen Schritte, die zur geforderten Zuverlässigkeit in einem Softwaresystem führen, und stellt die einzelnen Einflüsse in Beziehung zu den entsprechenden Abschnitten dieser Norm. Hiermit wird eine logische Vorgehensweise aufgezeigt, wie diese Norm bei der Projektimplementierung angewendet werden kann.

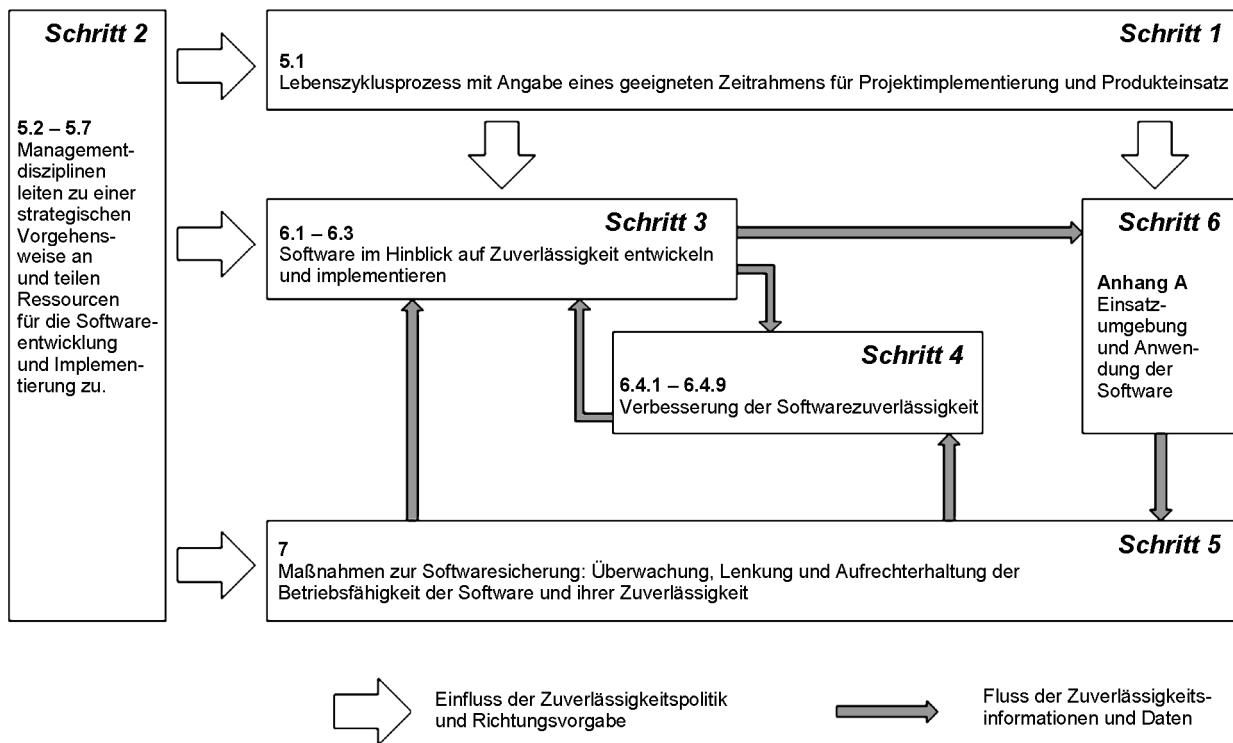


Bild 1 – Hinweise zur Anwendung dieser Norm

- Schritt 1:** den für das Projekt geltenden Lebenszyklus und Zeitrahmen ermitteln
- Schritt 2:** die Eignung des Zuverlässigkeitsmanagementprozesses und die verfügbaren Mittel dahingehend überprüfen und beurteilen, ob diese der Projektentwicklung und Implementierung gerecht werden
- Schritt 3:** die zutreffende Zuverlässigkeitsmethodik für Softwareentwicklung und Realisierung anwenden und diese umsetzen
- Schritt 4:** Maßnahmen zur Zuverlässigkeitsverbesserung unter Berücksichtigung der verschiedenen Einschränkungen initiieren, soweit diese praktikabel und durchführbar sind
- Schritt 5:** den Entwicklungsfortschritt und den Implementierungsprozess auf Einhaltung der Vorgaben überwachen
- Schritt 6:** Informationen über tatsächliche Einsatzumgebungen und über die im Betrieb erreichte Zuverlässigkeit für mögliche Verbesserungsmaßnahmen zur Verfügung stellen

5 Managementeinfluss auf die Softwarezuverlässigkeit

5.1 Systemlebenszyklus- und Softwarelebenszyklus-Prozesse

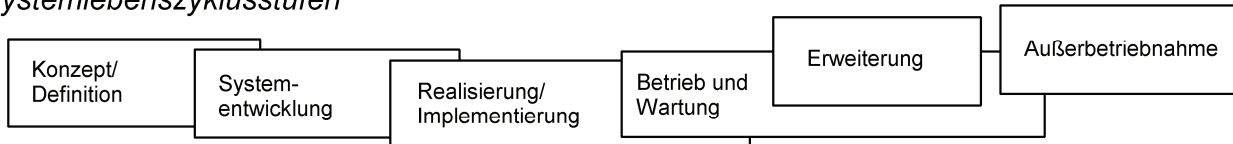
Der Systemlebenszyklus setzt sich aus einer Folge sich überlappender Stufen zusammen: Konzeption-/Definition, Systementwicklung, Realisierung und Implementierung des Systems, Betrieb und Wartung, Erweiterung und Außerbetriebnahme. Der Systemlebenszyklus beschreibt die Projektmanagementphasen anhand von Entscheidungspunkten, die Tätigkeiten können sich dabei von einer Stufe zur anderen überlappen. Die einzelnen Stufen eines Systemlebenszyklus sind in IEC 60300-3-15 beschrieben, somit kann der Abschluss einer Aufgabe ermittelt und die technische Verantwortung weitergegeben werden. Die Softwarelebenszyklusprozesse werden mit dem Systemlebenszyklus abgestimmt und umfassen die folgenden Tätigkeiten:

- **Anforderungen festlegen:** Aus Systemsicht werden die Anforderungen an kombinierte Hardware- und Software-Elemente unter Berücksichtigung der Bedürfnisse der Anwender und den 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 Software-Subsysteme 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 zu bilden 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 Softwarefunktion erarbeitet und entwickelt; die benötigten Softwareeinheiten und Schnittstellen für die Softwarefunktion werden in ausreichender Detaillierung erschaffen, 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, dieses schließt Tätigkeiten in unteren Ebenen ein wie
 - *Kodierung* von Rechner Quellcode 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 in das Host-Hardware-System zum Testen installiert;
- *Abnahme*: In dieser Stufe werden die Systemfähigkeiten hergestellt und die Softwareanwendungen werden 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, sie 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.

In Bild 2 sind die einzelnen Tätigkeiten im Softwarelebenszyklus-Prozess den entsprechenden Systemlebenszyklusstufen gegenübergestellt. Die aus Sicht des Softwarelebenszyklus wichtigen Zuverlässigkeitstätigkeiten werden ermittelt und in das Projekt aufgenommen.

Systemlebenszyklusstufen



Tätigkeiten im Softwarelebenszyklus

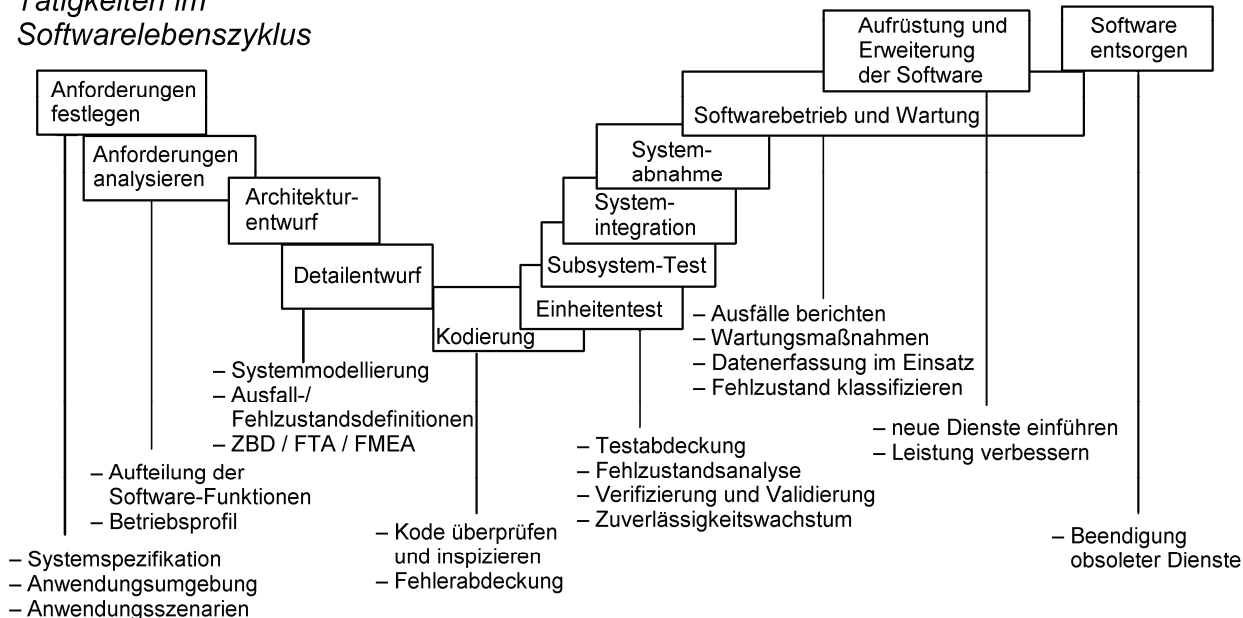


Bild 2 – Tätigkeiten im Softwarelebenszyklus

5.2 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 beim 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:** Schutz vor Eindringen in die Softwareanwendung und den Softwareeinsatz;
- **Sicherheit:** Verhinderung von Schäden durch die Softwareanwendung und den Softwareeinsatz;
- **Funktionstüchtigkeit (Operabilität):** robuster, fehlertoleranter und unterbrechungsfreier Betrieb;
- **Wiederverwendbarkeit:** Einsatz einer existierenden Software in anderen Anwendungen;
- **Wartungsbereitschaft:** Bereitstellung und Management von Mitteln zur Durchführung von Softwarewartungsaufgaben;
- **Portabilität:** Plattform-übergreifende Anwendung der Software.

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

5.3 Softwareentwicklungsumgebung

Die Zielsetzung des Zuverlässigkeitsmanagements [2] ist die Schaffung einer gut ausgewogenen und kreativen Entwicklungsumgebung im Rahmen der Projektbudgetmittel, dem Zeitplan und der Liefertermine. Organisationen im Zusammenhang mit Softwareentwicklung und Softwaredienstleistungen sind auf die Anwendung durch die Nutzer orientiert. Typische Softwareorganisationen, Softwarekategorien und Anwendungen sind im Anhang A beschrieben. 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 Projektzuschchnitts für ein erfolgreiches Zuverlässigkeitsmanagement. Dabei sollten mögliche Gelegenheiten für die Vergabe der Entwicklungsarbeiten, für eine Wiederverwendung der Software und die Verwendung handelsüblicher Softwareprodukte für die Systemintegration ausgelotet werden.

Entwurfsabwägungen und Risikomanagement werden häufig zur Projektoptimierung für eine fristgerechte Fertigstellung eingesetzt. 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 vorgegangen. Dieses ist der formalen Implementierung der *Capability Maturity Model Integration* [22] nach Anhang F für Software-Prozessmanagement 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 eingesetzt. Viele der Softwarezuverlässigkeits-Vorhersagemodelle, der Methoden der statistischen Analyse und der Software-Testwerkzeuge sind in der gegenwärtigen Industriepraxis automatisiert worden und werden jetzt 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.

5.4 Erarbeitung der Anforderungen an die Software und der Zuverlässigkeitsziele

Anforderungen an die Software sollten für die einzelnen Stufen des Softwarelebenszyklus erarbeitet werden. Für die Implementierung in der jeweiligen Stufe sollten geeignete Zuverlässigkeitstätigkeiten ermittelt werden. Die zeitliche Einplanung der Ausführung der einzelnen Zuverlässigkeitstätigkeiten ist wichtig. Zuverlässigkeitsanwendungen sind zeitabhängig und haben erhebliche Auswirkung auf die Lebenszykluskosten des Systems [4]. Der Projektzuschchnitt ist für Entwurfsabwägungen und die Überwindung von Einschränkungen wesentlich. Die Erarbeitung der Anforderungen an die Software und die Zuverlässigkeitszielsetzung sollten Teil der gesamten Softwareproduktspezifikationen darstellen. Die Strategie für die Implementierung der Softwarezuverlässigkeit wird in 5.7 beschrieben. Die Methodik für die technische Realisierung der Zuverlässigkeit in Softwaremodule und die Erarbeitung der Softwaresystemarchitektur wird in Abschnitt 6 behandelt. Eine Zusammenfassung der typischen Anforderungen an die Software und zugehöriger Zuverlässigkeitstätigkeiten für die Softwarelebenszyklusstufen wird im Anhang B dargestellt.

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 für zu erbringende Dienstleistungen benötigt werden. Systematische Vorgehensweisen bei der Implementierung der jeweiligen Zuverlässigkeitstätigkeiten in allen Stufen des Softwarelebenszyklus könnten das Erreichen der Zuverlässigkeitszielsetzungen sichern. Spezielle Zuverlässigkeitszielsetzungen werden aus den wichtigsten, nach 5.2 ausgewählten Softwarezuverlässigkeitsmerkmalen

abgeleitet, diesbezügliche quantitative Maßgrößen werden in Abschnitt 6.2 vorgestellt. Anforderungen an die Softwarezuverlässigkeit können projektweise formuliert werden.

Die Einflüsse kombinierter Hardware/Software-Systeme auf deren Zuverlässigkeitsspezifikationen sind in den Systemzuverlässigkeitsspezifikationen beschrieben [15]. Zu den typischen Einflussfaktoren, die sich während der Softwareentwicklung auf das Erreichen der Zuverlässigkeit auswirken, gehören:

- die Entwicklungskultur der Organisation und ihre Erfahrung mit Softwareentwurf und Implementierung;
- Verständnis für die Anwendungsumgebungen und dynamische Marktentwicklungen hin zu neuen Plattformen oder hin zur Entwicklung von Leistungsmerkmalen für praktische Implementierungen;
- Dokumentationsprozesse wie Ausfallberichterstattung, Datenerfassung, Softwarekonfigurationsmanagement zur Überwachung der Softwareversionen und Pflege der Aufzeichnungen gemachter Erfahrungen;
- Anwendung von Softwareentwurfsregeln zur Fehlervermeidung durch Überwachung und Lenkung des Entwurfsprozesses zur Optimierung des Softwareverhaltens hinsichtlich Softwarekomplexität, Programmkomplexität und funktionaler Komplexität;
- wirkungsvoller Einsatz geeigneter Softwaremethoden und Werkzeuge wie etwa strukturierter Entwurf, Fehlertoleranz, Entwicklungsbewertung [10] und Softwarefehlermanagement zur Förderung des Zuverlässigkeitswachstums;
- Wahl geeigneter höherer Programmiersprachen wie etwa Ada, C++, die gut für eine gezielte und strukturierte Softwareentwicklung geeignet sind;
- anerkannte Vorgaben zur Qualifizierung und Messung von Systemzuverlässigkeitskenngrößen.

5.5 Fehlaussagen, Irrtümer, Fehler und Ausfälle von Software verstehen

Eine Software-Fehlaussage ist ein Irrtum oder Versehen, der bzw. das von Menschen begangen wird, und sich dann in eine Fehlaussagen enthaltende Software niederschlägt. Softwarefehlaussagen werden meistens von einem Programmierer bei der Kodierung erzeugt. Der Programmierer könnte aber auch eine Vorgabe in der Softwarespezifikation falsch interpretieren, oder es könnte eine nicht korrekte Übersetzung oder Auslassung einer Vorgabe in der Entwurfsspezifikation geben. Auch können Softwarefehlaussagen durch fehlerhafte Hardware wie Rechenfehler in fehlerhaften Rechengeräten entstehen.

Befindet sich die Software in einem Fehlzustand, so kann sie die geforderte Leistung aufgrund der Fehlaussagen nicht erbringen. Softwarefehler ergeben sich entweder durch Spezifizierungsfehler, Entwurfsfehler, Programmierfehler, vom Compiler eingeschleuste Fehler oder bei der Wartung der Software versehentlich eingebrachte Fehler. Ein Softwarefehler bleibt solange latent, bis dieser durch einen speziellen Trigger ausgelöst wird, er fällt nach Entfernung des Triggers wieder in einen schlafenden Zustand. Ein Trigger in der Software ist das Auftreten betrieblicher Umstände, die einen schlafenden Fehler aktivieren. Der Trigger kann der interne Zustand des Systems sein, die gerade verarbeitenden Daten, Eingriffe des Operators, Umwelteinflüsse oder Kombinationen davon. Ein Softwarefehler kann zu einem Systemausfall führen.

Ein Ausfall ist ein Ereignis, welches durch den Verlust der Fähigkeit, eine geforderte Softwarefunktion innerhalb gegebener Grenzen zu erfüllen, ausgelöst wird. Ein Softwareausfall ist eine Manifestation eines Softwarefehlers, durch den die Existenz des latenten Fehlers aufgedeckt wird. Ein einzelner Softwarefehler wird sich solange als ein Ausfall manifestieren, bis der Fehler beseitigt ist. Softwareausfälle zeigen systematisch gleiche Fehlereigenschaften bei Eintritt des Ausfalls.

Softwareausfälle tauchen zufällig während des Systembetriebs auf und können jederzeit im Systemlebenszyklus eintreten.

5.6 Klassifizierung von Softwarefehlern

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 wie eine gemeinsame Ausfallart auftauchen. Diese Methode ist als *Orthogonal Defect Classification* [25] bekannt und wird in der Softwareindustrie verbreitet angewendet. Die Fehlerinformation wird regelmäßig erfasst und analysiert, um den Reifegrad der Software in Bezug auf

die Fehlerarten zu ermitteln. Die kausale Analyse der Fehlerdaten stellt ebenfalls einen Weg zur Verringerung von Softwarefehlern und zur Zuverlässigkeitsverbesserung dar.

Beim Klassifizierungsprozess werden die erfassten Informationen anhand eines vordefinierten Satzes von Fehlerkategorien gruppiert. Der Fehlerkategoriensatz kann zu zwei Situationen zusammengefasst werden: a) ein Fehler wird gefunden, der als *opener section* (Öffner) bekannt ist, und b) ein Fehler wird behoben, der als *closer section* (Schließer) bekannt ist.

Der Anhang C enthält eine Übersicht der Klassifizierung von Softwarefehlern und deren Attribute.

5.7 Strategie zur Implementierung der Softwarezuverlässigkeit

5.7.1 Vermeidung von Softwarefehlern

Der Softwarecode für ein Softwareprodukt ist einfach zu erzeugen. Ein Missgriff oder ein Irrtum, der während des Entwurfs gemacht wurde, könnte später im Betrieb 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.

- a) *Prävention von Softwarefehlern* vermeidet das Einschleusen von Fehlern in die Software, diese Vorbeugemaßnahme kann durch das Umsetzen geeigneter Softwareentwicklungsmethoden erfolgen.
- b) Bei der *Beseitigung von Softwarefehlern* versucht man zunächst das Vorhandensein von Softwarefehlern zu erkennen. Dann werden diese Softwarefehler während der Softwareentwicklung und im Rahmen von Wartungsmaßnahmen während des laufenden Softwarebetriebs verifiziert und validiert und dann beseitigt.

5.7.2 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 Testen der Software könnte vom Projektmanagement als zu umfassend und somit zu aufwändig und meist auch als zu lange und zu teuer erachtet werden, ohne eine Garantie für eine vollständige Beseitigung von Softwarefehlern zu bieten. Zur Beherrschung von Softwarefehlern verwendet man Fehlertoleranz- und Vorhersagemethoden und versucht so, die Manifestierung von nach der Freigabe der Software für den Einsatz etwa noch vorhandener latenter Fehler zu verringern.

- a) *Fehlertoleranz* bietet einen Steuerungsmechanismus, der vor einem Systemausfall warnt und den Betrieb trotz eingetretener oder gerade eintretender Fehler in der Software aufrechterhält.
- b) *Fehlervorhersage* ist ein Mittel zur Schätzung oder Vorhersage von vermutlich noch in der Software verbliebener Fehler mit dem Ziel, diese dann doch noch beseitigen zu können oder wenigstens deren Auswirkung zu umgehen. Über die Vorhersage erhält man auch eine zeitliche Vorschau für die Softwarefreigaben (Softwarerelease) gemäß eingeführter Freigabekriterien.

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 wieder, Software mit Beständigkeit zu entwickeln und zuverlässige Produkte für beabsichtigte Anwendungen zu liefern. Die folgenden Vorgehensweisen zur Vermeidung und zur Überwachung und Beherrschung von Softwarefehlern werden in der Praxis der Softwareentwicklung verwendet.

- a) *Methoden standardisieren* für den Architekturentwurf auf hoher Ebene (en: high-level architectural design), Detailentwurf, Kodierung und Konstruktion und Dokumentation zur leichteren Kommunikation und Fehlervermeidung;

- b) *Modulare Entwürfe konstruieren* für Softwareeinheiten und Subsysteme mit gut definierten Softwarefunktionen und Schnittstellen durch die Schaffung einfacher, getrennter und unabhängiger Softwaremoduleinheiten zur Erleichterung von Wechselwirkungen im Entwurf, für die Wartung, die Verfolgbarkeit von Fehlaussagen und Irrtümern und die Beseitigung von Fehlern;
- c) *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;
- d) *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 Softwarefehlern, um deren Ausbreitung zu verhindern;
- e) *Inspektionen und Überprüfungen durchführen* zu den Anforderungen an den Softwareentwurf, von Softwarecodes, Benutzerhandbüchern, Schulungsunterlagen und Testdokumenten mit dem Ziel, so viel wie möglich (unbeabsichtigte) Missgriffe und Softwarefehlaussagen und Irrtümer zu beseitigen; dabei können verschiedene Überprüfungsteams zum Vergleich der Ergebnisse eingesetzt werden;
- f) *Änderungsüberwachung* zur Verringerung des Auftretens von Fehlern wie beispielsweise Versions- und Änderungsüberwachungsprozesse im Softwarekonfigurationsmanagement.

Zu den wesentlichen Verfahren zur Erzielung der Softwarezuverlässigkeit gehören folgende, industrieübliche Methoden: formale Entwurfsmethoden, Softwarezuverlässigkeitstechnik, Verifizierung und Validierung der Software, Ursache-Folgenanalyse und Datenerfassung.

6.2 Softwarezuverlässigkeitsmaßgrößen und Datenerfassung

Softwarezuverlässigkeitsmaßgrößen sind die Zuverlässigkeitskenngrößen eines Softwaresystems. Die Softwarezuverlässigkeitsmaßgrößen liefern eine quantitative Skale und Methode zur Bestimmung des Wertes einer spezifischen Kenngröße im Zusammenhang mit dem Softwaresystem. Die folgenden Maßgrößen (en: metric) werden üblicherweise zur Beschreibung des Verhaltens und Zustandes und zur Leistungsmessung eines Softwaresystems verwendet und dienen der Beurteilung der Systemzuverlässigkeit aufgrund direkter Messungen oder über Rückschlüsse.

- a) *Fehlerdichte* ist ein Maß für die Anzahl der Fehler je tausend Zeilen Quelltext (KSLOC) und dient der Beurteilung der Systemzuverlässigkeit.
- b) *Funktionspunkt* ist ein Maß für den funktionalen Umfang der Anwendungssoftware zur Planung eines Softwareprojektes anhand der „function point analysis“-Methode [20].
- c) *Codeüberdeckung* (en: *Code coverage*) ist ein Maß für den Grad, zu dem der Quellcode eines Softwareprogramms systematisch getestet worden ist; Codeüberdeckung ist ein Indikator für die Gründlichkeit der Belastungsprüfung der Software, er wird zur Darstellung des Fehlerüberdeckungsgrades (en: *fault coverage*) verwendet, der den Anteil der bei der versuchsweisen Ausführung des Codes erkannten Fehler angibt.
- d) *Grad der Beseitigung von Softwarefehlern* 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 Softwarefehlern* wird beim Zuverlässigkeitswachstum dazu benutzt, den Wachstumstrend festzustellen.
- e) *Restfehler in der Software* sind ein Maß für die geschätzte Anzahl der in dem Softwareprodukt nach dem Testen auf Beseitigung von Softwarefehlern noch verbliebenen latenten Fehler.
- f) *Zeit bis zur Freigabe der Software* ist ein Maß für die geschätzte Zeit bis zur Freigabe des Softwareprodukts bzw. für den Zeitplan der Freigabe. Bei dieser Schätzung bedient man sich etablierter Kriterien bezüglich eines akzeptablen Niveaus noch verbliebener latenter Fehler im Softwareprodukt, deren Angabe für das Softwareprojektmanagement dienlich ist.
- g) *Softwarekomplexität* ist ein Maß für den Schwierigkeitsgrad des Entwurfs und der Implementierung einer Softwarefunktion oder eines Softwaresystems; weitere Komplexitätsmaßgrößen auf der Grundlage des Komplexitätskonzepts sind u. A. Programmkomplexität, funktionale Komplexität, operationelle Komplexität; auf die Komplexität bezogene Maßgrößen (Metriken) dienen als Eingabe in die Zuverlässigkeitsbeurteilung und die Vorhersagemodelle.

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

- *Fehlerdatenmaßgrößen* erfassen die zu Softwareproblemen berichteten Daten, zur Messung der Auswirkung der Fehlzustände sowie zur Messung der Effizienz des Berichtssystems zur Verbesserung der Softwarewartung.
- *Prozessdatenmaßgrößen* erfassen Informationen über den Wiederherstellungsprozess der Software und der Bedingungen zum Zeitpunkt des Erkennens und der Beseitigung eines Fehlers, diese dienen als Eingaben in Zuverlässigkeitsvorhersagemodelle.
- *Produktdatenmaßgrößen* 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 Maßgrößen liefern Erfahrungsberichte über das Verhalten und die Leistung und weitere Informationen zu verschiedenen Softwareproduktgruppen.

Der Datenerfassungsprozess ist für das Messen der Softwarezuverlässigkeit von entscheidender Bedeutung. Ein wirksames Datenerfassungssystem sollte leicht umsetzbar sein. Umfang und Art der zu erfassenden Daten sollten relativ einfach sein, für die Datenanalyse leicht interpretierbar und für die Beurteilung der Systemzuverlässigkeit und für die Verbesserung und Erweiterung nützlich sein. Die erfassten Daten werden benutzt zur Bestimmung der Systemzuverlässigkeitstrends, der Häufigkeit und Dauer der Softwarewartung, der Reaktionszeit nach Anforderung des Wartungsdienstes, der Dauer für die Wiederherstellung nach reduzierter Leistung und der Anforderungen an die Instandhaltungsunterstützung.

Anhang D enthält Beispiele von Softwaredatenmaßgrößen, die aus Datenerfassungen erhalten wurden.

6.3 Beurteilung der Softwarezuverlässigkeit

6.3.1 Prozess der Beurteilung der Softwarezuverlässigkeit

Der Prozess der Beurteilung der Softwarezuverlässigkeit konzentriert sich auf die Tätigkeiten zur Software-sicherung, die systematisch zur Beurteilung der Systemzuverlässigkeit während des Softwarelebenszyklus durchgeführt werden. Das Ziel der Prozessimplementierung ist die Sicherstellung der Softwarereife während der Entwicklung und das Erreichen der Zuverlässigkeit durch Bewertung der Testergebnisse zur Verbesserung des Zuverlässigkeitswachstums. Der Beurteilungsprozess ist der Mechanismus, der die Verifizierung der Softwareanforderungen und die Validierung der erzielten Ergebnisse der Softwarezuverlässigkeit ermöglicht.

Der Prozess der Beurteilung der Zuverlässigkeit umfasst entscheidende Tätigkeiten zur Softwaresicherung, die von einer erprobten Industriepraxis herrühren, die als Technik zur Realisierung der Softwarefunktionsfähigkeit (en: software reliability engineering) [23] bekannt ist. Softwarefunktionsfähigkeit ist ein Schlüsselmerkmal des Systembetriebs. Der Begriff *Softwarezuverlässigkeit* (en: software dependability) wird in dieser Norm anstelle von *Softwarefunktionsfähigkeit* (en: software reliability) aus Gründen der Ausweitung des Anwendungsbereichs verwendet. Dieses steht in Übereinstimmung mit der Definition in IEC 60050-191, nach der die Funktionsfähigkeit eines der Hauptmerkmale der Zuverlässigkeit ist. Es wird darauf hingewiesen, dass das Konzept der Technik zur Realisierung der Softwarefunktionsfähigkeit ausgeweitet werden könnte auf weitere Zuverlässigkeitsmerkmale und diese Technik nicht auf die Anwendung auf Funktionsfähigkeit begrenzt ist. So könnte etwa der Prozess der Realisierung der Softwarefunktionsfähigkeit auf die Sicherheit von Softwaresystemen und deren Beurteilung angepasst werden. Zur Beurteilung von Sicherheitsanwendungen sind jedoch besondere Kenngrößen nötig.

Das untenstehende Beispiel konzentriert sich auf die Sicherung der Softwarefunktionsfähigkeit, da das Konzept der Funktionsfähigkeit weitgehend eingeführt ist und dieses für eine leichtere Erläuterung hilfreich ist. Die folgenden Schlüsselthemen der Sicherstellung der Softwarefunktionsfähigkeit während des Softwarelebenszyklus werden behandelt.

- Zielsetzung der Funktionsfähigkeit,
- Betriebsprofil,
- Aufteilung der Funktionsfähigkeit,
- Analyse und Bewertung der Funktionsfähigkeit,

- Testen der Software und Messungen,
- Wachstum der Softwarefunktionsfähigkeit und Vorausschau,
- Beurteilung der Funktionsfähigkeit und Rückmeldung.

Der Prozess der Beurteilung der Softwarezuverlässigkeit mit dem Schwerpunkt auf das Thema der Sicherung der Funktionsfähigkeit ist in Bild 3 dargestellt.

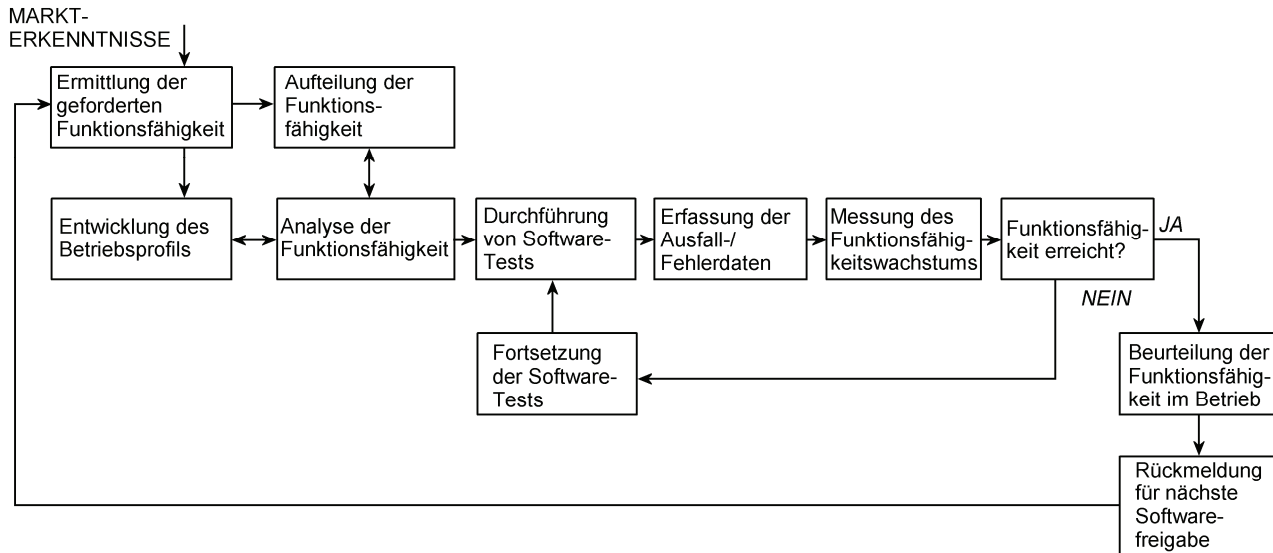


Bild 3 – Beurteilung der Softwarezuverlässigkeit mit Schwerpunkt auf der Funktionsfähigkeit

6.3.2 Zielsetzung der Betrachtungen zur Funktionsfähigkeit

Zielsetzung der Betrachtungen zur Funktionsfähigkeit ist die Bestimmung der quantitativen Maßgrößen der Funktionsfähigkeit aus Sicht der Dienstleistung an den Kunden und der Anwendung durch den Nutzer. Der Prozess der Bestimmung eines Systemfunktionsfähigkeitszieles für Spezifikationen kombinierter Hardware-/ Software-Systeme ist in den Spezifikationen der Zuverlässigkeit von Systemen [15] beschrieben. Die Kenngrößen der Systemfunktionsfähigkeit sind üblicherweise die mittlere Dauer bis zum Ausfall, Ausfallrate oder andere Kenngrößen mit Zeitbezug. Aus dem generellen Wert für die Funktionsfähigkeit leiten sich die Zielwerte für das kombinierte Hardware-/Software-System ab. Die Funktionsfähigkeitswerte werden auf die einzelnen Hardware- und Softwaresubsysteme aufgeteilt. Für ein Softwaresystem ist es unerlässlich, das Betriebsprofil zu betrachten, das sich bei auf Anforderung gestarteten Anwendungen auf die Ausführungszeiten der Softwarefunktionen auswirkt. Dieses ist den Lastzyklen eines Hardwaresystems ähnlich. Latente Softwarefehlzustände werden dann nicht offengelegt, wenn der Softwarecode nicht ausgeführt wird. Ein Modell zur Beschreibung der Systemfunktionsfähigkeit wie etwa das funktionale Zuverlässigkeitsblockdiagramm (*Reliability Block Diagram* [9]) kann zur Darstellung des Hardware- und Softwaresystems konstruiert werden. Die dabei gebildeten funktionalen Blöcke erleichtern die Aufteilung der entsprechenden Softwarefunktionsfähigkeitswerte auf die Softwarefunktionen, die entsprechend der Softwaresystemarchitektur und der Softwarekonfiguration gebildet wurden.

6.3.3 Betriebsprofil

Ein Betriebsprofil ist die Abfolge der geforderten Tätigkeiten, die vom kombinierten Hardware/Software-System ausgeführt werden müssen, um die gestellte Aufgabe oder die Dienstleistung zu erbringen. Die Systemleistung hängt in starkem Maße von der Umgebung ab, in der das System betrieben wird. Durch die Umgebungseinwirkungen, beispielsweise durch Hitze und Vibrationen, können sich physikalische Änderungen in der Hardware ergeben, die zum Ausfall der Hardware während des Systembetriebs führen können. Die Umgebung wirkt sich jedoch nicht auf die Softwarefunktionen aus, die durch die Ausführung der Rechnerprogramme während des Systembetriebs erbracht werden. Softwareausfälle werden möglicherweise durch Eingaben in die Software verursacht, die diese zur Ausführung erhält. Diese können von außerhalb der Umgebung stammen, wie beispielsweise irrtümliche Dateneingabe oder ein während der Ausführung angetroffener Irrtum im Quellcode.

Durch die Ausarbeitung eines Betriebsprofils erhält man eine quantitative Kennzeichnung der Nutzung der Software. Betriebsdaten und zugehörige Angaben werden üblicherweise durch Kundenbefragungen erfasst und aus Einsatzerfahrungen zusammengetragen. Nachfolgend wird der Prozess der Ausarbeitung eines Betriebsprofils beschrieben.

- a) Das Kundenprofil wird durch die Zusammenstellung der Bedürfnisse der Kunden und der Kundenarten, etwa eine Organisation oder eine einzelne Person, die den Erwerb oder Kauf eines Softwaresystems beabsichtigen, bestimmt.
- b) Das Nutzerprofil wird für die verschiedenen Nutzerarten, etwa eine Person oder ein Angestellter einer Organisation bestimmt, welche das Softwaresystem für bestimmte Anwendungen nutzen möchten.
- c) Es wird das Systemmodusprofil bestimmt, das angibt, wie das System in welcher Abfolge oder Reihenfolge betrieben wird. Dieses drückt man durch Betriebsmodi aus, wie etwa Software testen bei Wartungsarbeiten nach Aufrüstung oder normale Stapelverarbeitung mit Ausführung des Softwaresystems.
- d) Das funktionale Profil wird durch Bewertung jedes Systemmodus für Leistungsfunktionen und Dienstmerkmale bestimmt, wie beispielsweise eine E-Mail schreiben oder eine Adresse nachschlagen, wie diese Merkmale von den funktionalen Anforderungen an die Software vorgesehen sind.
- e) Das Betriebsprofil wird ausgehend von den funktionalen Profilen bestimmt, die für die Systemleistungsfunktionen erarbeitet wurden.

Das funktionale Profil ist eine Nutzersicht der Systemfähigkeiten. Aus der Perspektive des Entwicklers stellt das funktionale Profil die Systemoperationen dar, die die geforderten Funktionen tatsächlich umsetzen. Aus einer Zuverlässigkeitsperspektive ist das Betriebsprofil ein disjunkter Satz unterschiedlicher Systembetriebs-szenarien und deren Eintrittshäufigkeiten. Das Betriebsprofil liefert die für die Entwicklung von Testfällen notwendigen Eingaben, mit denen der Einsatz des Softwaresystems und spezielle Anwendungen der funktionalen Softwareleistungsmerkmale simuliert werden können. Die Ausführung der Testfälle zum Testen und Prüfen der Software ergibt wertvolle Informationen, und die dabei erfassten Daten dienen zur Schätzung der Softwarefunktionsfähigkeit im Einsatz.

6.3.4 Aufteilung der Funktionsfähigkeit

Die Aufteilung der Funktionsfähigkeit auf ein Hardware-/Software-System findet auf der Grundlage des Systemfunktionsfähigkeitsmodells statt, das geschaffen wurde, um die Anforderungen an die Funktionsfähigkeit des Systems widerzuspiegeln. Die anfänglich aufgeteilten und zugewiesenen Werte sind die für die Systemkonfiguration spezifizierten Werte der verschiedenen Funktionsfähigkeitskenngrößen. Diese anfänglichen Aufteilungen der Funktionsfähigkeit sind durch Erfahrungsdaten geprägt. Diese Werte werden dann durch eine iterative Analyse und einen Bewertungsprozess weiter verfeinert. Die Aufteilung und Zuweisung der Funktionsfähigkeitswerte auf die verschiedenen Subsysteme und Einheiten orientiert sich an deren Komplexität, Kritikalität, der schätzungsweise erzielbaren Funktionsfähigkeit und anderen für den Aufteilungsprozess relevanten Einflussfaktoren.

Die Konstruktion des Funktionsfähigkeitsmodells für Software unterscheidet sich aufgrund der inhärenten Betriebsmerkmale der Software erheblich von der Konstruktion eines Hardwaremodells. Für jeden Modus des Systembetriebs, an dem Softwareprogrammfunktionen als Konfigurationsteile beteiligt sind, werden unterschiedliche Sätze von Softwarebestandteilen ausgeführt. Jeder Modus hat einen eigenen Anwendungszeitpunkt, der mit der Dauer der Ausführung der Softwareeinheit, auf Anforderung durch den Systembetrieb, verbunden ist. Somit wird die Zeitdauer jedes Systemmodus angezeigt. Zur Modellierung der Softwarefunktionsfähigkeit gehören die Anzahl der Zeilen des Quellcodes in jeder Softwareeinheit, die Komplexität des Codes und andere Angaben zu den Software-Entwicklungsmitteln, wie Programmiersprache und Entwurfsumgebung. Sie werden dazu verwendet, die anfängliche Fehlerdichte für die Vorhersage der Funktionsfähigkeit der Softwareeinheiten zu ermitteln.

6.3.5 Analyse und Bewertung der Funktionsfähigkeit

Die Analyse der Funktionsfähigkeit und ihre Bewertung sind zur Entwicklung eines Softwaresystems erforderlich. Es handelt sich um einen iterativen Prozess der Optimierung der Anforderungen an die Funktionsfähigkeit, damit die geforderte Systemleistung erreicht werden kann.

a) Modellierung der für die Funktionsfähigkeit relevanten Funktionen

Ein funktionales Funktionsfähigkeitsmodell für ein aus funktionalen Blöcken bestehendes kombiniertes Hardware-/Software-System kann mittels der Technik des *Zuverlässigkeitsblockdiagramms* [9] konstruiert werden. Das Modell wird in separate Subsystemmodelle zerlegt, die die Hardware- und Software-Bestandteile des Systems repräsentieren.

Das Funktionsfähigkeitsmodell des Hardware-Subsystems besteht aus allen Hardwareelementen des Systems. Die Blöcke des Zuverlässigkeitsblockdiagramms sind dabei in Reihe oder parallel angeordnet, je nach der erforderlichen Darstellung der entsprechenden Struktur der Hardware-Subsysteme und der Redundanzkonfiguration. Dieses Modell erleichtert die Vorhersage der Ausfallraten der einzelnen Hardwarekomponenten und die Bestimmung der Funktionsfähigkeit der Hardware-Subsysteme anhand der Zuverlässigkeitsvorhersagetechniken. Hierbei können ein oder mehrere Hardware-Subsysteme verschiedene Funktionen der Systemkonfiguration bedienen.

Das Funktionsfähigkeitsmodell der Software-Subsysteme wird mittels Softwareeinheiten konstruiert. Dabei sind die Softwareeinheiten die Bausteine für die Bereitstellung der Softwareprogrammfunktionen. Eine Softwareeinheit oder ein Modul bilden die unterste Ebene eines Softwareelements. Softwaremodule fallen nicht unabhängig voneinander aus, wie dies bei Hardwarekomponenten der Fall ist. Softwarecodes sind virtuelle Betrachtungseinheiten, die keine physikalischen Änderungen erfahren können. Softwaremodule fallen in Verbindung mit dem Systembetriebsprofil aus, das sich auf das Konfigurationsschema der Modellstruktur der Softwarefunktionsfähigkeit auswirkt. Bei der Modellierung der Softwarefunktionsfähigkeit müssen die Angaben zum Betriebsprofil bei der Ausarbeitung der Softwarekonfigurationsstruktur mit berücksichtigt werden. Das Software-Subsystemprogramm kann aus einem oder mehreren Softwarebestandteilen bestehen, die die geforderten Funktionen erbringen. Ein Software-Subsystemprogramm, das in einem Host-Hardware-Subsystem residiert, ist so konfiguriert, dass es ein Softwarekonfigurationsteil ist. Die Wechselwirkungen und gegenseitigen Abhängigkeiten des Software-Subsystems und seinem designierten Hardware-Host werden benötigt, um die jeweiligen Subsystem-Softwarefunktionen für den Systembetrieb zu erbringen. Es kann ein oder mehrere kombinierte Software- und Hardware-Subsysteme geben, die verschiedene Funktionen in der gesamten Systemkonfiguration erbringen.

Das Beispiel im Anhang E stellt die Wechselwirkungen kombinierter Hardware-/Software-Funktionsfähigkeits-Funktionen dar und zeigt, wie man die Systemausfallrate für die Beurteilung der Zuverlässigkeit ableiten kann.

b) Bestimmung der Funktionsfähigkeit der Softwarefunktionen

Während der Ausführung des Systems können Softwareausfälle auftreten. Die Bestimmung der Softwareausfallraten zur Verwendung bei der Modellierung der Funktionsfähigkeit erfordert, dass die Software als ein Subsystem behandelt wird, welches in einem Host-Hardware-Subsystem residiert und so konfiguriert ist, dass es ein Softwarekonfigurationsteil ist. Das Software-Subsystem kann eine oder mehrere der geforderten Funktionen ausführen. Eine Funktion ist eine Fähigkeit des Systems, eine geforderte Dienstleistung aus der Perspektive des Endnutzers zu erbringen. Die Funktion kann von einem Softwarekonfigurationsteil erbracht werden, wobei die geforderte Dienstfunktion von der Versionssteuerung anerkannt ist. Eine Softwareeinheit, die exakt zur Ausführung einer einzigen Funktion gestaltet wurde, kann ein Konfigurationsteil sein. Ein Software-Subsystemprogramm, das mehrere Softwareeinheiten zur Ausführung einer einzigen Funktion benötigt, kann ebenfalls ein Konfigurationsteil sein. Ein Software-Subsystem kann aus mehreren Softwareprogrammen bestehen, die einen Satz verbundener Funktionen erbringen. Jedes dieser Softwareprogramme ist per Definition ein Konfigurationsteil. Das Konzept der Softwarekonfigurationsteile wird aus der Perspektive der Softwareentwurfssteuerung und Versionskontrolle betrachtet. Softwarekonfigurationsteile sind für die Verfolgung von Entwurfsänderungen von wesentlicher Bedeutung. Jeder Entwurfsänderung wird zur Identifizierung eine Versionsnummer zugewiesen. Der Bezug auf die Softwareversion ist für die Verfolgung der Wirksamkeit aufgerüsteter Softwarewartung nötig. Der Trend des Zuverlässigkeitswachstums wird über ein angezeigtes besseres Systemverhalten ermittelt, wenn das System mit der neueren, die alte ersetzende Version läuft. Das Erbringen der geforderten Funktionen während des Systembetriebs ist die Herausforderung, die Zielsetzung für die Systemzuverlässigkeit zu erreichen.

Die ein System ausmachenden Softwarefunktionen beziehen sich auf eine an den zeitlichen Abläufen orientierte Konfiguration und auf eine Topologie der Funktionsfähigkeit.

Die an den zeitlichen Abläufen orientierte Konfiguration ist dann von großer Bedeutung, wenn die verschiedenen Funktionen während einer bestimmten zeitlichen Periode des Systembetriebs aktiv oder inaktiv sind. Die wichtigsten zeitlichen Beziehungen zwischen den Softwarefunktionen sind „gleichlaufend“ und „sequentiell (nacheinander)“. Funktionen sind dann gleichlaufend, wenn sie gleichzeitig aktiv sind. Funktionen sind dann sequentiell, wenn eine nach der anderen aktiv ist. Es kann aber auch sein, dass sich die Funktionslaufzeiten teilweise überlappen, was zu einer hybriden gleichlaufend/sequentiellen zeitlichen Konfiguration führt. Gleichlaufend aktive Softwarekonfigurationen werden in solchen Systemen angetroffen, die über mehr als eine zentrale Rechneinheit verfügen, Beispiele sind Multiprozessorsysteme oder Systeme mit verteilter Steuerung.

Topologie der Funktionsfähigkeit betrifft die Anzahl der Funktionen in einem System, die versagen können, ehe das ganze System ausfällt. Topologie der Funktionsfähigkeit ist die Beziehung eines einzelnen Funktionsausfalls zum Ausfall des aggregierten Systems. Softwarefunktionen stehen im Allgemeinen in einem seriellen Topologiebezug. Der Ausfall einer einzigen Funktion würde den Ausfall des Softwaresystems verursachen. Ein Softwarefehler tolerierender Entwurf kann das Überleben des Systems beim Ausfall einer oder mehrerer Funktionen sichern.

c) Bezug zur Softwarelaufzeit

Eine Softwareausfallrate kann hinsichtlich drei verschiedener Zeiträumen ausgedrückt werden.

- *Ausführungszeit* ist die Laufzeit (Laufdauer) der zentralen Rechneinheit, die sich anhäuft solange das Softwareprogramm Anweisungen ausführt. Die Ausführungszeit wird zur Bestimmung der von der Ausführungszeit abhängigen Ausfallrate des angewendeten Software-Subsystems benützt.
- *Systembetriebsdauer* erhöht sich immer dann, wenn das Hardware-/Software-System als Ganzes arbeitet. Dies wird zur Bestimmung der von der Betriebsdauer abhängigen Ausfallrate des andauernd angewendeten Software-Subsystems benützt.
- *Kalenderzeit* wird für Projektplanungszwecke verwendet. Kalenderzeit nimmt immer zu.

Während des Systembetriebs laufen die Rechnersoftwareprogramme nicht zwangsweise andauernd. Einige Programme können sich eine einzelne zentrale Rechneinheit zeitlich teilen. Multiple zentrale Rechneinheiten können ebenfalls vorhanden sein, dadurch können sich Programmausführungen überlappen. Die Ausfallraten der verschiedenen Rechnerprogramme müssen zu einer gesamten Softwareausfallrate kombiniert werden. Sie werden in ein gemeines Zeiträhmengerüst der Systembetriebsdauer umgewandelt. Dieses ist derselbe Zeiträhmengerüst, der für die Angabe von Hardware-Ausfallraten genommen wird, um die Angabe der Ausfallraten eines kombinierten Hardware-/Software-Systems zu erleichtern.

d) Kritizität der Softwarefunktionen

Kritizität ist ein Maß für die Schwere und Auswirkung der Abweichung von einer spezifizierten Funktion eines Systemelements. Die Abweichung kann ein Ausfall oder eine Verschlechterung je nach vorgegebenen Bewertungskriterien sein. Softwarefunktionen werden häufig zur Steuerung eines kritischen Systems eingesetzt, dessen Ausfall katastrophale Auswirkungen und Folgen hätte. Die Kritizität der Softwarefunktionen sollte früh bei der Systemkonzeption und Definition ermittelt und später während des Softwarearchitektur-entwurfs des Systems bewertet werden. Die Kritizität funktionaler Ausfälle sollte in den Systemspezifikationen klassifiziert und im Rahmen der Analyse der Systemfunktionsfähigkeit verifiziert werden.

Das potentielle durch eine kritische Softwarefunktion entstehende Risiko kann aus zwei Standpunkten heraus bestimmt und bewertet werden; a) aus einer Perspektive der Fehlervermeidung, bei der die Ausfallauswirkung durch Isolierung vermieden werden kann; und b) aus einer Perspektive der Fehlertoleranz, bei der die Schwere der Ausfallfolgen abgeschwächt werden können. Risikoreduktion ist eine strategische Vorgehensweise, bei der eine technische Lösung lediglich eine von mehreren Optionen ist. Zu den weiteren Optionen eines Projektrisikomanagements gehören Risikobehandlung und Risikoteilung [13].

Fehlzustandsbaumanalyse [8] kann zur Ermittlung der Ursache eines bekannten Ausfalls verwendet werden. Sie wird zur Untersuchung der potentiellen Fehler, deren Arten und Ursachen verwendet, und anhand dieser Analyse kann man den Beitrag zur Systemnichtverfügbarkeit quantifizieren. Fehlzustandsbaumanalyse ist eine von oben nach unten schreitende technische Vorgehensweise. Startpunkt ist hierbei das Software-Subsystemprogramm in der höchsten Ebene, die Analyse schreitet dann durch die hierarchische Softwarestruktur herunter bis zur niedrigsten Softwareeinheit. Die potentiellen Fehler können einzeln ermittelt und nach der jeweiligen Wahrscheinlichkeit für eintretende Ausfälle beurteilt werden. Die quantitative Beurteilung

liefert einen Hinweis oder ergibt die Größenordnung der Kritizität der Softwarefunktion. Diese Angaben sind von Interesse für die Entwurfsoptimierung und Fehlervermeidung.

Fehlzustandsart- und -auswirkungsanalyse [6] kann mögliche Ausfallarten und Fehler in den Softwareeinheiten ermitteln und deren Auswirkungen auf das in der nächst höheren Ebene der hierarchischen Softwarestruktur befindliche Subsystem feststellen. Fehlzustandsart- und -auswirkungsanalyse ist eine von unten nach oben schreitende technische Vorgehensweise. Sie kann ausgeweitet und zur Kritizitätsanalyse der Softwarefunktionen eingesetzt werden. Die Kritizitätsanalyse liefert quantitative Werte der Wahrscheinlichkeit für eintretende Ausfälle und qualitative Angaben zur Ausfallbeschreibung, die bei Abwägungen von Entwurfsänderungen und zur Fehlervermeidung dienlich sein können.

Weitere Verfahren zur Analyse der Systemzuverlässigkeit wie etwa Softwarezerlegung und Systemsimulation sind in IEC 60300-3-15 beschrieben. Diese Verfahren können selektiv zur detaillierten Zuverlässigkeitsbeurteilung von Softwarefunktionen in kombinierten Hardware/Software-Systemen verwendet werden.

Die Stufen der Unversehrtheit der Software (en: software integrity level) sind Werte für dem Projekt eigene Kenngrößen, mit denen die Bedeutung der Software für den Nutzer definiert wird. Beispiele projekteigener Kenngrößen sind u. A. Softwarekomplexität, Kritizität, Risiko, Sicherheitsniveau, gewünschte Leistung und Funktionsfähigkeit. Die Stufen der Unversehrtheit der Software werden bestimmt durch die Klassifizierung der Kritizität der Auswirkung von Ausfallfolgen mit den zugehörigen Eintrittshäufigkeiten [19]. Die Kritizität der Softwarefunktionen ist ebenfalls anwendungsspezifisch. Bei Systemen mit Sicherheitsbezug sollten die Stufen der Unversehrtheit definiert und in die Entwicklung des Softwaresystems mit aufgenommen werden, damit die funktionalen Sicherheitsanforderungen [12 und 17] erfüllt werden. Bei Systemen mit Bezug zur IT-Sicherheit sollten spezifische Anforderungen für diesen Sektor einbezogen werden.

6.3.6 Testen und Messen von Software

a) Allgemeine Betrachtungen zum Testen von Software

Testen von Software ist der Prozess der Ausführung eines Rechnerprogramms mit der Absicht, die Softwarefunktionen zu verifizieren und Fehlaussagen und Irrtümer zu finden. Die Zielsetzung von Softwaretests variiert mit den Projektbedürfnissen, der Verfügbarkeit des Softwareprodukts, dem Reifestatus der Software und der Zeitplanung für das Testen im Softwarelebenszyklus. Bei der Planung von Softwaretests sollte Nachfolgendes beachtet werden.

- Das Testen von Software ist eine Fertigkeit, die gute Testpraxis erfordert. Obgleich viele der Testroutinen und -werkzeuge automatisiert worden sind und in breitem Maße in der Industrie eingesetzt werden, verlangen gute Testtechniken die Fähigkeiten, die Erfahrung, die Intuition und die Kreativität der Tester, um zuverlässige Resultate zu erzielen. Die Aufzeichnung und Aufbewahrung der Testaufzeichnung ist wichtig für die Genauigkeit und Rückverfolgbarkeit von Testdaten.
- Testen ist mehr, als bloß das Softwareprogramm auf Fehlzustände zu durchsuchen (en: debugging) und diese zu orten und Fehlaussagen und Irrtümer zu korrigieren. Testen wird auch zur Verifizierung und Validierung von Software und zum Messen der Funktionsfähigkeit eingesetzt.
- Testen ist teuer. Die Testeffizienz und Prozesswirksamkeit sind Kriterien für Überdeckungs-basierte (en: coverage based) Testverfahren. Testautomatisierung kann die Softwaretestdauer beschleunigen und Projektkosten verringern. Die Wahl der geeigneten Testwerkzeuge, die Kosten für Schulung und Unterstützungskosten in Verbindung mit dem Erwerb der Testwerkzeuge sollten beachtet werden.
- Testen kann möglicherweise nicht das wirkungsvollste Mittel sein, die Softwarequalität zu verbessern, es sei denn, passende Folgemaßnahmen werden ergriffen. Alternative Methoden, wie Codeinspektion und Codeüberprüfung sollten erwogen werden.
- Softwaretesten ist nur ein Teil des Softwarezuverlässigkeits-Wachstums- und -verbesserungsprozesses. Zum Erreichen der Zuverlässigkeitsziele bedarf es der Zusammenarbeit mit weiteren Sicherungsbemühungen.
- Ein vollständiges Testen ist nicht machbar oder praktikabel, und verbietet sich meist schon aus Kostengründen. Softwarekomplexität wirkt sich auf das Ausmaß eines vollständigen Testens aus. Das Komplexitätsproblem grenzt häufig die Fähigkeit des Testers ein, Softwarefehlzustände während des Testprozesses zu entdecken und zu beseitigen.

- Nach der Freigabe der Software für den Einsatz gibt es in ihr weiterhin latente Softwarefehler. Die Vorhersage der Funktionsfähigkeit der Software ist ein Weg, die erforderliche Testdauer zu schätzen, ehe eine Software freigegeben werden kann.

b) Arten von Softwaretest

Nachfolgend sind die während des Softwarelebenszyklus durchgeführten verschiedenen Arten von Softwaretests aufgelistet.

- *Einheitentest*: Testen einer Softwareeinheit, die kompiliert werden kann, bevor sie in das Softwareprogramm oder Subsystem integriert wird. Durch den Test der Softwareeinheit soll verifiziert werden, ob der Detailentwurf der Einheit korrekt durchgeführt wurde.
- *Subsystemtest*: Testen eines aus einer oder mehreren Softwareeinheiten bestehenden Subsystem-Softwareprogramms als Softwarekonfigurationsteil, um die funktionalen Leistungsanforderungen zu verifizieren
- *Integrationstest*: Testen eines Softwaresystems in einem Hardware-Host als Ganzes. In dem aus integrierten Subsystemen bestehenden Host soll die funktionale Betriebsfähigkeit getestet werden, es sollen Probleme mit den Softwareschnittstellen und bei den Wechselwirkungen offengelegt und die Funktionsfähigkeit validiert werden.
- *Zuverlässigkeitswachstumsprüfung*: Testen der Software in einem iterativen Prozess zur Verbesserung der Funktionsfähigkeit durch Testen bis zum Ausfall, Ausfälle analysieren, Korrekturmaßnahmen zur Aufrüstung der bestehenden Softwareversion umsetzen und den Test mit der neuen Softwareversion fortsetzen. Die Zuverlässigkeitswachstumsprüfung wird beendet, wenn die vorgegebenen Funktionsfähigkeitsziele erreicht sind.
- *Qualifikationsprüfung*: Testen zum Nachweis, dass die Software ihre Spezifikationen einhält, nachdem sie in das Host-Hardwaresystem integriert und bereit zum Einsatz in ihrer Zielumgebung ist. Vor Freigabe der endgültigen Version zur Softwareauslieferung werden häufig Alpha- und Betatests zu Zwecken der Qualitätssicherung gemacht. *Alphatest* ist ein von den Softwareentwicklern durchgeführter hausinterner Versuch vor Freigabe der Software an externe Nutzer. *Betatest* ist ein Feldversuch, der von einer begrenzten Anzahl von Nutzern in der beabsichtigten Anwendung durchgeführt wird mit der Absicht, ggf. deren Rückmeldungen und gemachte Erfahrungen zu erhalten.
- *Abnahmeprüfung*: Testen eines Softwaresystems zur Validierung der Kundenanforderungen.
- *Regressionstest*: Testen der Software die schon zuvor getestet worden war in dem Bemühen, etwaig eingeschleuste Wartungsirrtümer, neu entwickelten Code, unsaubere Konfiguration oder unpassende Quellenkontrolle aufzudecken.

c) Prüfbarkeit der Software

Mit *Prüfbarkeit* oder *Testbarkeit* der Software bezeichnet man das Ausmaß, zu dem die Software sich eignet, Testkriterien aufstellen und Tests durchführen zu können. Beim Entwurf der Prüfbarkeit werden die Softwaremerkmale analysiert und die Wahrscheinlichkeit vorhergesagt, dass es Fehler in der Software gibt, die durch Testen aufgedeckt werden könnten. Die Analyse dient der Optimierung der Testprozesse, um herauszufinden, wann genug getestet wurde. Sie ist ein Weg, die Testmittel richtig einzusetzen und den Wert einer bestimmten Testvorgehensweise zu ermitteln.

d) Testfälle

Testfälle werden zur Simulation tatsächlicher Einsatzbedingungen der Software entwickelt, bei denen möglicherweise bestimmte interessierende Bereiche oder potentielle Problembereiche zu erwarten sind. Ein Testfall ist ein Satz von Testeingaben, Ausführungsbedingungen und erwarteten Ergebnissen, die auf ein bestimmtes Testziel hin entwickelt wurden. Solche Ziele könnten beispielsweise das Durchlaufen eines besonderen Pfades in dem Softwareprogramm sein oder die Verifizierung der Erfüllung einer bestimmten Anforderung. Bei der Spezifizierung von Testfällen werden die zu machenden Eingaben vorgegeben und dokumentiert, die Ermittlung der erwarteten Testergebnisse vorbereitet und die Bedingungen für die Ausführung des Prüflings, also des zu testenden Teils, hergestellt. Zu einem wirkungsvollen Testprozess gehören sowohl manuelle als auch automatisch erzeugte Testfälle. Manuell gestaltete Tests gehen in die Tiefe und versuchen so, Softwarefehler zu finden, und sie reflektieren dabei, wie der Entwickler das Problemgebiet und die Datenstruktur gesehen hat. Automatische Tests decken die Breite der Fehleruntersuchungen ab, indem sie den gesamten Bereich der Testwerte durchlaufen, einschließlich derjenigen Extreme, die Menschen vergessen haben könnten. Der automatische Testprozess bedingt den Einsatz eines Testfall-

generators, der Quellcode, Testkriterien, Spezifikationen oder Definitionen von Datenstrukturen als Eingabe akzeptieren muss, um daraus Testdaten zu erzeugen und die erwarteten Ergebnisse zu bestimmen. Das Einfügen von Fehlern könnte als ein Testfall (von vielen) erwogen werden. Dabei wird ganz bewusst ein Fehler in einen Teil des Softwaresystems eingefügt, um herauszufinden, ob ein anderer Teil wie beabsichtigt reagiert. Die Testergebnisse werden zur Bestimmung möglicher Fehlerbedingungen verwendet, und sie erleichtern einen fehlertolerierenden Entwurf.

Das Testen eines Softwareprogramms ist der Versuch, die Software zum Ausfallen zu bringen. Jede fehlgeschlagene Ausführung muss einen Testfall hervorbringen, der in die Testsuite des Softwareprojektes aufgenommen wird. Der wichtigste Aspekt einer Teststrategie ist die Anzahl der Fehler, die der Test als Funktion der Zeit aufgedeckt hat. Dieses ergibt einen Anhaltspunkt für die Testeffizienz und Testwirksamkeit.

e) Softwaremessung

Messung ist der Prozess der Bestimmung oder Schätzung eines quantitativen Wertes einer Maßgröße. Die Messdaten erhält man auf unterschiedliche Weisen aus verschiedenen Perspektiven zur Vorhersage der Funktionsfähigkeit der Software und ihrer Verbesserung.

- *Maßgrößen (Metriken) der Entwurfsstruktur:* Messung der Vorgehensweise beim Entwurf, der Komplexität und der Unabhängigkeit des Softwareentwurfs.
- *Maßgrößen der Entwurfsvollständigkeit:* Messung der Fähigkeit der Software, gegebene Anforderungen und Funktionen vollständig zu erfüllen und zu erbringen.
- *Prozessmaßgrößen:* Messung des Managements, der Wirtschaftlichkeit (Kosteneffizienz) und von Entwurfsabwägungen der Software.
- *Produktmaßgrößen:* Messung der Kenngrößen der Software, die dem entwickelten Softwareprodukt eigen sind.

Viele dieser Maßgrößen sind Eingaben für die Parameter des Softwarefunktionsfähigkeitsmodells zur Vorhersage und Schätzung gegebenenfalls benötigter quantitativer Werte.

Anhang G enthält eine Übersicht der üblicherweise in der industriellen Praxis für Softwarefunktionsfähigkeitsmodelle verwendeten Maßgrößen.

6.3.7 Wachstum und Vorhersage der Zuverlässigkeit von Software

Softwarezuverlässigkeitswachstum ist eine Eigenschaft der Software, gekennzeichnet durch eine mit der Zeit progressive Verbesserung einer Funktionsfähigkeitskenngröße des Softwaresystems. Eine Verbesserung der Softwarefunktionsfähigkeit wird durch Entwerfen und Gestalten erreicht und das progressive Erreichen der Funktionsfähigkeit wird durch eine Zuverlässigkeitswachstumsprüfung verifiziert. Software fällt nicht aus, solange sie nicht ausgeführt wird. Ein Rechnerprogramm kann nur dann ausfallen, wenn es läuft beziehungsweise ausgeführt wird. Softwareausfälle decken Fehler auf, und die Beseitigung dieser Fehler führt zu einem Zuverlässigkeitswachstum. Die Trends eines Softwarezuverlässigkeitswachstums ergeben sich aus der Fehlerbeseitigungsrate bezogen auf die kumulative Softwareausführungszeit. Zu Planungszwecken kann die Ausführungszeit in Kalenderzeit konvertiert werden, damit man Softwareausfallraten zur Angabe der Funktionsfähigkeit bilden kann. Ein Softwarezuverlässigkeits-Wachstumsprogramm [7] kann für kombinierte Hardware/Softwaresysteme aufgestellt werden. Typische Verfahren zur Verbesserung des Softwareentwurfs sind in Abschnitt 6.4. angegeben. Das Testen auf ein Funktionsfähigkeitswachstum stellt sich wie folgt dar.

a) Testen auf Software-Funktionsfähigkeitswachstum

Testen auf Funktionsfähigkeitswachstum hat zum Ziel, die momentane Funktionsfähigkeit zu beurteilen, Fehler zu ermitteln und zu beseitigen und die zukünftige Funktionsfähigkeit vorherzusagen. Die Funktionsfähigkeitswerte anhand der während der Ausführungszeit gezählten entdeckten und beseitigten Fehler werden mit den zwischenzeitlichen Zielsetzungen der Softwarefunktionsfähigkeit verglichen. Damit kann der Trend der Funktionsfähigkeit während des Testens gemessen werden, und man erhält Angaben zum möglichen Erreichen der geforderten Softwarefunktionsfähigkeit.

b) Softwareausführungsumgebung

Zu der Umgebung, in der die Software ausgeführt wird, gehören die Hardwareplattform, wie etwa das Hardware-Host-System, die Systembetriebssoftware, die Parameter zur Systemgenerierung, die Rechnerauslastung und das Betriebsprofil. Das Betriebsprofil ist in 6.3.3 beschrieben.

Ein *Lauf* ist das Ergebnis eines ausgeführten Softwareprogramms. Ein Lauf hat bestimmte Eingabe- und Ausgabevariablen. Zum Testen der Softwarefunktionsfähigkeit wird ein Satz Eingabevariablen mit ihren jeweiligen Werten für einen bestimmten Lauf ausgewählt. Jede Eingabevariable besitzt einen erklärten Datentyp, der einen Bereich und Reihenfolge zulässiger Werte darstellt. Ein Betriebsprofil ist eine Funktion, die im Zusammenhang zur Wahrscheinlichkeit der Eingabevariable steht, die zur statistischen Schätzung des Funktionsfähigkeitswachstums dient.

c) Mehrfache Softwarekopien

Die Dauer eines Funktionsfähigkeitstests im Rahmen einer Zuverlässigkeitswachstumsprüfung kann durch mehr als eine Softwarekopie akkumuliert werden. Die Kopien können zur Testbeschleunigung gleichzeitig laufen. Dieses Verfahren erlaubt die Akkumulation der Laufzeit mittels mehrfacher Kopien zur Beschleunigung des Testprozesses, insbesondere wenn hohe Funktionsfähigkeitsziele nachgewiesen werden sollen. Somit wird die gesamte Testdauer (gemessen in Kalenderzeit) reduziert. Der Einsatz mehrfacher Kopien kann wirtschaftliche und planerische Vorteile bringen.

d) Vorhersage der Softwarefunktionsfähigkeit

Wachstum der Funktionsfähigkeit der Software ist die positive Verbesserung der Softwarefunktionsfähigkeit mit der Zeit, die durch die systematische Beseitigung von Softwarefehlzuständen erzielt wird. Die Wachstumsrate der Funktionsfähigkeit hängt davon ab, wie schnell Fehler aufgedeckt und beseitigt werden können. Ein auf Wachstumsbedingungen anwendbares Softwarefunktionsfähigkeitsmodell erlaubt es dem Projektmanagement, den Fortschritt bei der Softwarefunktionsfähigkeit durch statistischen Rückschluss zu verfolgen; damit können Trends aufgezeigt und zukünftige Funktionsfähigkeitsziele vorhergesagt werden. Auch können geeignete Managementmaßnahmen getroffen werden, sollte der Trend negativ sein.

e) Softwarefunktionsfähigkeitsmodelle

Messung und Vorhersage des Wachstums der Funktionsfähigkeit der Software erfordert die Verwendung eines geeigneten Softwarefunktionsfähigkeitsmodells, welches die Variation der Softwarefunktionsfähigkeit mit der Zeit beschreibt. Die Parameter dieses Funktionsfähigkeitsmodells können entweder durch Vorhersage anhand von Erfahrungsdaten oder durch Schätzung von während des Systemtests erfasster Testdaten erhalten werden. Die Schätzung erfolgt anhand der Ausfallzeitpunkte und einem - für eine signifikante Akkumulation der Ausführungszeit - ausreichendem Stichprobenumfang. Damit soll ein vernünftiges Maß an statistischem Vertrauen zur Validierung des Wachstumstrends der Funktionsfähigkeit erhalten werden.

Anhang H enthält typische Beispiele für in der industriellen Praxis verwendete Softwarefunktionsfähigkeitsmodelle.

6.3.8 Beurteilung der Softwarefunktionsfähigkeit und Rückmeldung

Die Erfassung von Daten zur Softwarefunktionsfähigkeit wird in 6.2 behandelt. Die Datenerfassung des Betriebsverhaltens der Software dient der Beurteilung der Funktionsfähigkeit im Einsatz beim Kunden. Damit soll sichergestellt und bestätigt werden, dass das Funktionsfähigkeitsniveau für den Einsatz der Software aufrechterhalten bleibt. Die im Betrieb erreichte Funktionsfähigkeit und die zugehörigen Daten werden zusammen mit zugehörigen Angaben des Kunden erfasst. Anhand dieser Informationen können Änderungen für neue Anforderungen an die Software gerechtfertigt und Entwicklungen für eine neue Version der Software angestoßen werden.

Angesichts der Dynamik der Anwendungsumgebungen und der technologischen Weiterentwicklung stehen Entscheidungen für die Freigabe neuer Softwareversionen häufig unter dem Einfluss des Wettbewerbs und werden somit von Geschäftsstrategien bestimmt.

6.4 Verbesserung der Softwarezuverlässigkeit

6.4.1 Überblick über die Möglichkeiten zur Verbesserung der Softwarezuverlässigkeit

Softwarezuverlässigkeit kann verbessert werden durch einen besseren Softwareentwurf, durch Zuverlässigkeitswachstumsprüfungen und eine bessere Wartung der Software im Rahmen des Kundendienstes, einschließlich Erweiterungsleistungen.

Funktionsfähigkeit ist ein Hauptmerkmal der Softwarezuverlässigkeit. Die Verbesserung der Softwarefunktionsfähigkeit durch Zuverlässigkeitswachstumsprüfungen wird in 6.3.7 beschrieben. Nachfolgend werden einige praktische Vorgehensweisen zum Entwurf zuverlässiger Software mit dazu relevanten Techniken zur Erweiterung und Implementierung der Software vorgestellt. Die Zielsetzungen beim Entwurf konzentrieren sich auf die *Prüfbarkeit* zur leichteren Verifizierung der Softwarefunktionen, auf die *Modularität* für die Unabhängigkeit jeder Softwarefunktion zur leichteren Isolierung und Eindämmung von Fehlern und auf die *Instandhaltbarkeit* zur leichteren Modifizierung während des Softwarelebenszyklus.

6.4.2 Vereinfachung der Softwarekomplexität

a) Strukturelle Komplexität

Strukturelle Komplexität beschreibt die Logikpfade der Verbindungen von Softwaremoduleinheiten im Softwareentwurf. Jede Moduleinheit könnte so programmiert (codiert) werden, dass sie eine ausführbare Einheit einer Softwarefunktion in der Softwarestruktur wird. Strukturelle Komplexität bezieht sich auf die Prüfbarkeit von Programmcode, die Auswirkung auf das Erkennen von Fehlern hat, somit wirkt sich die Prüfbarkeit auf die Zuverlässigkeit (Funktionsfähigkeit und Wartbarkeit) der Softwarearchitektur aus. Je komplexer die Struktur, desto schwieriger wird es, die Software zu testen. Die Regeln für den Softwareentwurf sollten ein Komplexitätsniveau vorgeben, um die Gestaltung der Zuverlässigkeit zu erleichtern.

b) Funktionale Komplexität

Funktionale Komplexität beschreibt die geforderten Funktionen, die das Softwaremodul oder Codesegment in der Einheit ausführen und erbringen muss. Idealerweise sollte eine Moduleinheit so gestaltet werden, dass sie nur eine Funktion ausführt, dies würde Einfachheit durch einen Satz kohärenter Eingaben und Ausgaben bedeuten und so die Isolierung und Beseitigung von Softwarefehlern erleichtern. In der Praxis sollten sowohl die strukturelle als auch die funktionale Komplexität für die Bewertung des Softwareentwurfs betrachtet werden. Die Softwareentwurfsstrategie bezüglich Komplexität hat einen direkten Bezug zur Anzahl von Testfällen, die zur vollständigen Verifizierung der Software benötigt werden.

6.4.3 Softwarefehlertoleranz

Fehlertoleranz ist die Fähigkeit der Software, in Gegenwart gewisser Fehler weiter zu funktionieren und dabei die Unversehrtheit des Datendurchsatzes zu bewahren. Ein fehlertolerierender Entwurf soll Softwarefehler davon abhalten, das System im laufendem Betrieb zum Ausfallen zu bringen. Softwarefehlertoleranz sollte so gestaltet sein, dass es nur eine geringe Wahrscheinlichkeit für Ausfälle gemeinsamer Ursache durch verschiedenartige (diversitäre) Systemeigenschaften gibt; einige solcher Eigenschaften sind:

- *Fehler „eindämmen oder einsperren“*: Die Software wird dergestalt geschrieben, dass beim Auftauchen eines Fehlers dieser keine anderen Abschnitte der Software über die lokale Domäne hinaus, in der er eingetreten ist, kontaminieren kann;
- *Fehlererkennung*: Die Software wird dergestalt geschrieben, dass sie selbst auf das Vorhandensein von Fehlern prüft und bei deren Eintreten darauf reagiert;
- *Erholung nach einem Fehler*: Die Software wird dergestalt geschrieben, dass sie nach dem Erkennen eines Fehlers selbst ausreichende Maßnahmen einleitet, die es der Software erlauben, weiterhin erfolgreich zu funktionieren;
- *Entwurfsdiversität*: Software und ihre Daten werden so geschaffen und gestaltet, dass es nötigenfalls Rückfalllösungen gibt.

Fehlertoleranz steht für die Eigenschaft der Software, trotz eines Teilausfalls im Softwaresystem weiterhin für eine gewisse Zeit richtig funktionieren zu können. Dadurch sollen Ausfälle vermieden werden, die andernfalls zu einem abrupten Zusammenbruch des Systems führen würden. Fehlertoleranz ist von besonderer Wichtig-

keit bei sicherheitskritischen Systemen, die auf eine hohe Systemverfügbarkeit angewiesen sind, selbst in Gegenwart von Fehlern oder bei Betrieb unter abträglichen Bedingungen. Ein Beispiel eines fehlertolerierenden Entwurfs ist das Netzwerkprotokoll „Transmission Control Protocol (TCP/IP)“ für Internetkommunikation. Dies ist ein Softwareprotokoll, welches so gestaltet ist, dass es eine zuverlässige bidirektionale Kommunikation in paketvermittelnden Netzen ermöglicht, selbst in Gegenwart von mangelhaften oder überlasteten Übertragungsstrecken. Den fehlertolerierenden Protokollentwurf erreicht man, indem man die Endpunkte der Übertragung dafür vorbereitet, verlorene Pakete, duplizierte Pakete, Pakete in falscher Reihenfolge oder korruptierte Pakete zu erwarten, so dass diese Bedingungen die Unversehrtheit der Daten nicht beschädigen und lediglich den Durchsatz, der zur Aufrechterhaltung des Systembetriebs nötig ist, um einen entsprechenden Betrag verringern.

6.4.4 Softwareinteroperabilität

Mit Softwareinteroperabilität bezeichnet man die Fähigkeit diverser Softwaresysteme, zum Informationsaustausch zusammenarbeiten und die ausgetauschten Informationen verwenden zu können. In einem offenen System wie bei einem IP-Netz ist es wichtig, die Interoperabilität sehr diverser Softwaresysteme zum Betrieb von Verbindungsleitungen (Kommunikationsbeziehungen) zu erreichen. Ein Ausfall der Verbindungsleitung würde die Zuverlässigkeit beeinträchtigen. Eine Vorgehensweise zur Verbesserung der Interoperabilität in Kommunikationsnetzen ist die Aufnahme eines geeigneten Merkmals in den Softwaresystementwurf, den Zustand der hergestellten Verbindung zu überwachen. Beispielsweise beinhaltet die „Herzschlag“-Technik (Signalverarbeitung und Synchronisierungsverfahren) solch ein Überwachungsmerkmal, danach soll das „Herzschlag“-Signal zu allen anderen an der bereits hergestellten Verbindung beteiligten Stellen gesendet werden. Falls die Verbindung unterbrochen wird oder durch Umwelteinflüsse oder andere Ursachen zusammenbricht, wird das Softwaresystem automatisch versuchen, die Verbindung wieder herzustellen und die unterbrochene Kommunikation wieder dauerhaft aufzunehmen, so dass die Zuverlässigkeit des Kommunikationsnetzes nicht verschlechtert wird.

6.4.5 Softwarewiederverwendung

Softwarewiederverwendung ist die Verwendung vorhandener Software, um daraus neue Software zu bilden. Wiederverwendbare Software ist ein wiederverwendbares Gut. Das bekannteste wiederverwendbare Gut ist Code. Ein einmal geschriebener Programmcode kann in einem später geschriebenen anderen Programm wiederverwendet werden. Die Wiederverwendung von Programmcode ist eine übliche Technik, durch die Zeit und Energie durch Verringerung redundanter Arbeit gespart werden soll. *Softwarewiederverwendbarkeit* ist der Grad, zu dem ein Softwaregut in mehr als einem Softwaresystem eingesetzt werden oder man daraus neu Güter bilden kann.

Aus der Perspektive der Zuverlässigkeit sollte die Wiederverwendung von Software in Projekten und die Merkmale ihre Wiederverwendbarkeit gelenkt und gesteuert werden, um daraus eine Zuverlässigkeitsverbesserung zu erzielen. Wiederverwendbarkeit hängt direkt von der Softwarestruktur und einer modularen Gestaltung ab. Damit eine Softwaremoduleinheit wiederverwendbar wird, sollte sie darauf beschränkt werden, lediglich eine Funktion vollständig ausführen zu können. Diese Beschränkung auf eine Funktion ist unerlässlich. Sollte bei der beabsichtigten Wiederverwendung der Softwaremoduleinheit diese weniger als eine Funktion ausführen sollen, oder sollte sie in der Lage sein, mehr als eine Funktion auszuführen, so würde es schwierig sein, den beabsichtigten Zweck ihrer Wiederverwendung zu implementieren oder diesen beizubehalten. Ein Abweichen von dieser Beschränkung würde die Nützlichkeit der wiederverwendbaren Software mindern. Eine Wiederverwendung von Software, die nicht exakt eine Funktion ausführt, könnte aufgrund der Möglichkeit des Einschleusens von Fehlaussagen und Irrtümern in die Software während der Implementierung oder Wartung nachteilige Auswirkungen auf die Zuverlässigkeit haben.

Die Wiederverwendung von Software sollte nur dann tatsächlich vollzogen werden, falls die funktionalen Anforderungen der neuen Softwaremoduleinheit mit den Anforderungen der wiederverwendbaren Software für sehr ähnliche Anwendungen und betriebliche Umgebungen übereinstimmen. Andernfalls würde dies zu einer Minderung der Wirtschaftlichkeit der wiederverwendeten Software und nach dem Vollzug möglicherweise zu einer Minderung der Zuverlässigkeit führen.

Wiederverwendbare Software sollte für etwaige Rückverfolgungen gut dokumentiert werden und so das Konfigurationsmanagement von Softwaregütern erleichtern. Handelsübliche Softwareprodukte sollten wie wiederverwendbare Software für variierende mehrfache Anwendungen behandelt werden. Qualifikations-

prüfungen für Beurteilungszwecke sollten zur Validierung handelsüblicher Softwareprodukte und der Eignung zur Erfüllung von Projektanforderungen durchgeführt werden.

6.4.6 Wartung und Erweiterung von Software

Softwarewartung ist die Modifizierung eines Softwareprodukts nach Auslieferung zur Korrektur von Fehlern, zur Verbesserung der Leistung der Software oder anderer Merkmale, oder um das Produkt an eine veränderte Umgebung anzupassen. Es gibt vier Hauptkategorien der Softwarewartung.

- *korrigierende Wartung*: rückwirkend nach Auslieferung ausgeführte Modifizierung eines Softwareprodukts zur Behebung aufgedeckter Probleme;
- *adaptive Wartung*: nach Auslieferung eines Softwareprodukts ausgeführte Modifizierung, um das Softwareprodukt in einer veränderten oder sich ändernden Umgebung weiterhin einsetzbar zu halten;
- *verbessernde Wartung*: Modifizierung eines Softwareprodukts nach Auslieferung zur Verbesserung der Leistung oder der Wartbarkeit;
- *vorbeugende Wartung*: Modifizierung eines Softwareprodukts nach Auslieferung, um latente Fehler in dem Softwareprodukt aufzudecken und zu korrigieren, bevor sich diese zu tatsächlichen Ausfällen ausbreiten können.

Die Hauptthemen der Softwarewartung sind lenkender und technischer Art. Zu den Lenkungsthemen gehören Anpassung an Kundenprioritäten, Planung der Wartungsmittel und deren Zuweisung, Schulung des Wartungspersonals, Weitervergabe von Wartungsarbeiten und Rückmeldung über die Zufriedenheit des Kunden. Zu den technischen Themen gehören: über Vorfälle berichten, Lösung technischer Probleme, Analyse der Auswirkungen, Standardisierung der Vorgehensweisen und Testgepflogenheiten, Beurteilung der Softwarewartung und Messung der Testeffizienz.

Softwareerweiterung (en: software enhancement) ist Teil des Softwareevolutionsprozesses. Softwaresysteme im Einsatz sind gekennzeichnet durch zunehmende Komplexität durch Modifizierungs- und Erweiterungsarbeiten zur Erfüllung der Kundenbedürfnisse, kontinuierliche Änderungen der Strategien zur Unterstützung der Wartung aufgrund konkurrierender Dienstleistungsangebote sowie durch die Notwendigkeit, die Fähigkeiten und Techniken weiterzuentwickeln, um der sich ändernden Geschäftsumgebung Rechnung zu tragen. Die Kosten der Softwarewartung sollten Teil der Strategie zur Sicherung der Zuverlässigkeit sein.

6.4.7 Softwaredokumentation

Softwaredokumentation ist der niedergeschriebene Text der Entwurfs- und Anwendungsdokumente bezüglich des Softwareprodukts. Dokumentation ist ein wichtiger Teil der Softwareentwicklung. Dazu gehören unter Anderen folgende verschiedene Arten der Softwaredokumentation.

a) Architektur- und Entwurfsdokumentation

Die Dokumentation gibt einen Überblick über das Softwareprodukt und dessen Beziehungen zur Anwendungsumgebung und den beim Entwerfen zu benützenden Konstruktionsprinzipien. Das Softwareentwurfsskizzenblatt ist ein umfassendes Softwareentwurfsmodell mit folgenden Einzelheiten.

- Der *Datenentwurf* beschreibt die in der Software residierenden Strukturen. Merkmale und Beziehungen zwischen Datenobjekten diktiert die Wahl der Datenstrukturen, die auf die strukturelle Komplexität der Software einwirken, die ihrerseits die Zuverlässigkeit beeinflusst.
- Der *Architekturentwurf* verwendet Informationsflusseigenschaften und bildet diese auf die Programmstrukturen ab, die auf die Modularität des Softwareentwurfs einwirken und der seinerseits die Funktionsfähigkeit des Softwaremodulentwurfs beeinflusst.
- Der *Schnittstellenentwurf* beschreibt die internen und externen Programmschnittstellen sowie auch den Entwurf der Mensch-Maschine-Schnittstelle. Interne und externe Schnittstellenentwürfe basieren auf den bei der Entwurfsanalyse erhaltenen Informationen, welche die Redundanzschemata und Anforderungen an die Funktionsfähigkeit des Softwaremodulentwurfs und Konfiguration beeinflussen.
- Der *prozedurale Entwurf* beschreibt die strukturierten Programmierkonzepte, die graphische, tabellenförmige und textförmige Notationen verwenden. Diese Entwurfsmedien ermöglichen es dem Designer (Softwareentwickler), prozedurale Details darzustellen, die die Übersetzung in Code erleichtern. Dieses

beeinflusst die Beständigkeit der Programmierpraktiken und führt zur Verringerung von eingeschleusten Codierirrtümern.

b) Technische Dokumentation

Zur technischen Dokumentation gehören Code-, Algorithmen- und Schnittstellenbeschreibungen sowie weiterer Text, der die verschiedenen Gesichtspunkte des beabsichtigten Betriebs und Einsatzes der Softwareprodukte beschreibt. Die Dokumentation sollte gründlich sein, aber zur leichteren Wartung und Aktualisierung dennoch den Quellcode knapp und präzise schreiben. Dies erleichtert Codeüberprüfung und Inspektion zur Verbesserung der Softwareunversehrtheit und der Funktionsfähigkeit.

c) Dokumentation für den Anwender (Nutzer)

Zur Dokumentation für den Anwender gehören Handbücher für Endnutzer, Systemadministratoren und Unterstützungspersonal. Der Zweck dieser Dokumentation ist die Unterstützung des Endnutzers bei der Anwendung der Softwareprodukte. Die Dokumentation beschreibt einfach, wie die Software in ihrer Anwendungsumgebung eingesetzt und verwendet werden kann. Die Dokumentation für den Anwender beschreibt außerdem die Merkmale des Softwareprogramms und unterstützt den Nutzer dabei, diese Merkmale in seiner Anwendung einzusetzen, einschließlich Anleitungen zur Fehlersuche und zur Hilfe. Manchmal kann auch eine Online-Hilfe verfügbar sein, die einen Anwender-freundlichen Zugang anbietet und der Kundenzufriedenheit dienen soll.

d) Marketing-Dokumentation

Zur Marketing-Dokumentation gehören Werbemittel, die den zufälligen Beobachter dazu ermutigen sollen, mehr über das Softwareprodukt wissen zu wollen. Internet-Zugang und Kundendienstzentren sind in der heutigen wettbewerbsorientierten Marktumgebung übliche Angebotswege zum Erwerb von Softwareprodukten und für Informationen über Anwendungsmöglichkeiten. Aus Anbietersicht ist die Fokussierung auf den Kunden ein wesentliches Element bei der Entwicklung einer Vermarktungsstrategie. Marketing-Dokumentation ist lediglich eine von vielen Vorgehensweisen zur Informationsverbreitung.

6.4.8 Automatisierte Werkzeuge

Automatisierte Werkzeuge sind für routinemäßige Datenverarbeitung, Rechenanalysen und zum Vergleich von Bewertungsergebnissen nützliche Helfer. Im Handel wird eine breite Palette von automatisierten Werkzeugen angeboten, mit denen die meisten Anwenderbedürfnisse erfüllt werden können, wie beispielsweise Modellierung, Analyse, und wissensbasierte Daten, mit denen alle Arten einer Funktionsfähigkeitsbeurteilung unterstützt werden können. Die Wahl und die Anwendung geeigneter Werkzeuge für besondere Projektaufgaben bedürfen des Wissens und der Erfahrung von auf diesem Gebiet erfahrenen Zuverlässigkeitsingenieuren. Automatisierte Werkzeuge könnten bei routinemäßigen Rechenarbeiten zur Verbesserung der Produktivität beitragen. Die Gültigkeit und Genauigkeit dieser automatisierten Werkzeuge sollte vor einer Anwendung im Projekt untersucht werden. Die Unterstützbarkeit dieser Werkzeuge sollte vor deren Erwerb bestimmt werden.

6.4.9 Technische Unterstützung und Anwenderschulung

Technische Unterstützung sind Dienste, die beim Einsatz von Softwareprodukten assistieren. Ziel solcher Dienste ist es, dem Anwender dabei zu helfen, spezielle Probleme beim Betrieb oder bei der Anwendung eines Produkts zu lösen. Technische Unterstützung wird auf vielfältige Weise und Wege geleistet, wie z. B. telefonische Anfragen, Online-Dienste, E-Mail, Reparatur über einen Zugang aus der Ferne und Vor-Ort-Besuche zur Problemlösung. Auf diesem Gebiet ist ein starkes Wachstum von ausgelagerten Callcentern (Telefon-Beratungszentrum) zu verzeichnen, die von Organisationen für die Entwicklung von Technologieprodukten aus geschäftlichen, wirtschaftlichen und geographischen Gründen genutzt werden, um schnelle Antworten auf Anfragen zur Unterstützung zu erleichtern. Diese Callcenter dienen der zentralisierten technischen Unterstützung eines breiten Bereichs von Technologieprodukten wie beispielsweise Rechner-systemen, die Software enthalten, die rund um die Uhr über einen weltweiten, gebührenfreien Nutzerzugang technischer Hilfe bedürfen. Technische Unterstützungsdienste sind Teil der Wartungsunterstützung, die den Betrieb des Produkts und die Funktionsfähigkeit aufrecht erhalten sollen und die so zur Zuverlässigkeitsverbesserung beitragen.

Softwareanwenderschulung ist ein wichtiger Gesichtspunkt der Verbesserung der Softwarezuverlässigkeit. Ziel solcher Schulungen ist es, die Fähigkeiten der Anwender und Nutzer zu erweitern oder das Verständnis

für die Anwendung des Softwareprodukts zu stärken. Softwareanwenderschulung kann viele Formen haben wie z. B. Online-Zugang zur Befragung der Datenbank des Produktlieferanten, Hilfestellung über Callcenter, dedizierte Hilfestellung durch technische Experten zur Behandlung angetroffener unüblicher Probleme.

7 Softwaresicherung

7.1 Überblick über die Softwaresicherung

Softwaresicherung besteht aus geplanten und systematischen Tätigkeiten des Qualitätsmanagements, die gewährleisten, dass die Softwarelebenszyklusprozesse und die Softwareprodukte den Anforderungen, Standards und Normen und den Verfahren entsprechen. Das CMMI-Model [22] wird in breitem Maße als ein Managementwerkzeug zur Implementierung von Softwaresicherungsprogrammen in Software entwickelnden Organisationen eingesetzt.

Zur Softwaresicherung gehören üblicherweise die technischen Disziplinen Qualität, Zuverlässigkeit und Sicherheit im Rahmen der Entwicklung von Softwareprodukten und beim Systembetrieb. Der Softwaresicherungsprozess zielt darauf ab, Grundlagen zu planen, zu entwickeln, zu erhalten, auf denen Vertrauen gebildet werden kann und Entscheidungen getroffen werden können. Damit soll erreicht werden, dass das System und sein Softwareanteil die vorgegebenen Sicherheits- und Zuverlässigkeitsziele während des gesamten Lebenszyklus erfüllen. Der Softwaresicherungsprozess zielt des Weiteren darauf ab, sicherzustellen, dass das System die geforderten physikalischen und funktionalen Eigenschaften besitzt und somit die festgelegten Systemspezifikationen nachgewiesen werden können. Softwaresicherung ist der Risikobeurteilung, Verifizierung und Validierung, Dokumentation und Bewahrung von Aufzeichnungen aus Audits als objektive Beweise verpflichtet. Softwaresicherung benützt relevante projektbasierte Messdaten zur Beobachtung des Softwareprodukts und geeignete Prozesse für mögliche Verbesserungen.

Ein wesentlicher Teil der Softwaresicherung ist die in [23, 24] beschriebene Umsetzung der Entwicklungsprozesse zur Realisierung der Softwarefunktionsfähigkeit. Softwarezuverlässigkeit und Qualität sind Voraussetzungen für einen sicheren Systembetrieb.

7.2 Technologische Einflüsse auf die Softwaresicherung

Das schnelle Anwachsen handelsüblicher Softwareprodukte für diverse Anwendungen hat eine noch nie dagewesene Situation in der IT-Branche geschaffen, die der Aufmerksamkeit bedarf. Softwaretechnologie hat zahlreiche Fortschritte hin zu einer effizienten Softwareentwicklung gemacht, die zu einer großen Vielseitigkeit und zu wirtschaftlichen Vorteilen bei Softwareanwendungen geführt haben. Bis vor kurzem war Softwaresicherung hauptsächlich auf Softwarequalität und die Verbesserung der Zuverlässigkeit aus Sicht der Produktentwicklung ausgerichtet. Gleichzeitig haben jedoch böswillige Attacken auf den Softwarebetrieb stark zugenommen, treten mehr hervor und sind raffinierter und gefährlicher geworden. Sie betreffen nicht nur handelsübliche Software einsetzende Softwareentwickler sondern führen auch bei den Betreibern von Softwaresystemen und den Anwendern der Endprodukte zu beträchtlichen Zeitverlusten. Die gesamte Situation hat sich zu einer Kettenreaktion entwickelt, die sich durch unbekannte Viren und häufig durch schleichende böswillige Attacken ausbreitet und die für die von Software abhängige IT-Operationen eine komplexe und dynamische Risikoumgebung schafft.

Softwaresicherung ist für Organisationen, die auf den Gebieten der Sicherheit und finanziellen Transaktionen tätig sind, hinsichtlich der Verwundbarkeit von Softwareanwendungen von kritischer Bedeutung. Zur Softwaresicherung gehören die Entwicklung und Implementierung von Methoden und Prozessen, die sicherstellen, dass die Software wie beabsichtigt funktioniert, und die gleichzeitig die Risiken für Verwundbarkeit, böartigen Code, Fehler oder Irrtümer abschwächen, die dem Endanwender Schaden zufügen könnten. Softwaresicherung ist für die Sicherheit kritischer IT-Ressourcen von vitaler Bedeutung. Angesichts der sich schnell ändernden Art der Bedrohungen ist selbst Software mit höchstem Qualitätsniveau nicht gegen böswilliges Eindringen gefeit, falls die Software nicht sauber konfiguriert ist und nicht richtig gewartet wird. Die Beherrschung der Bedrohungen im Cyberspace erfordert eine geschichtete Vorgehensweise der Vorsorge und Zusammenarbeit auf dem Gebiet der Sicherheit. Die Entwickler schaffen sicherere und robustere Software, die Systemintegratoren sorgen dafür, dass die Software korrekt installiert wird, die Betreiber halten das System funktionsfähig und die Endanwender nutzen die Software in einer sicheren Weise.

Diese Vorgehensweise führt dazu, dass mit Software befasste Organisationen ihre Softwaresicherung für ihren Betrieb und ihre Tätigkeiten neu definieren. So wird in [26] *Softwaresicherung* beispielsweise definiert als das „Vertrauensniveau, dass Software frei von Verwundbarkeit ist, die weder absichtlich noch unabsichtlich in die Software hinein gebracht oder die sich versehentlich irgendwo im Softwarelebenszyklus ergeben hat, und dass die Software in der beabsichtigten Weise funktioniert“. Softwaresicherung muss ein angemessenes und rechtfertigbares Vertrauensniveau erzeugen, dass die Software korrekt und in vorhersagbarer Weise in Übereinstimmung mit den dokumentierten Anforderungen funktionieren wird. Die Softwarefunktion darf nicht kompromittiert werden, weder durch direkte Attacken noch durch Sabotage über böswillig eingefügten Code, von dem man annimmt, dass dieser eigentlich gesichert sei.

Je nach den einzelnen Anwendungen richtet sich das Vertrauensniveau der Softwaresicherung an folgende Merkmale:

- a) *Vertrauenswürdigkeit* – es existiert keine ausnutzbare Verwundbarkeit, weder bösartig noch versehentlich eingefügt;
- b) *vorhersagbare Ausführung* – es besteht ein rechtfertigbares Vertrauen in die Software, dass diese bei ihrer Ausführung wie beabsichtigt funktioniert;
- c) *Konformität* – geplante und systematische Sätze multidisziplinärer Tätigkeiten zur Sicherung der Softwareprozesse und Softwareprodukte verhalten sich gemäß der ihnen vorgegebenen Anforderungen, Standards und Prozeduren.

Zu den Herausforderungen, die für die Softwaresicherung ermittelt wurden, gehören:

- 1) versehentlicher Entwurf oder Implementierung von Irrtümern, die zu ausnutzbarer Verwundbarkeit von Code führt;
- 2) die sich ändernde technologische Umgebung, die zu neuen Verwundbarkeiten führt und die den böswilligen Angreifern neue Möglichkeiten zum Anrichten von Schaden eröffnet;
- 3) böswillige Insider, die danach trachten, den Entwicklern oder den Endanwendern Schaden zuzufügen.

Die erste Herausforderung ist versehentlich und unabsichtlich (siehe 5.5), wogegen die zweite und die dritte Herausforderung absichtlich und bewusst von den böswilligen Angreifern genutzt werden. Die Gegenmaßnahme ist, Risiken im Zusammenhang mit diesen Herausforderungen durch optimale Verfahren der Softwaresicherung in den Griff zu bekommen.

7.3 Optimale Verfahren der Softwaresicherung

Es gibt Softwaretechnologie- und Softwaresicherungs-Foren [27] mit Beteiligung von Regierungsstellen, Industrie, Akademikern und Anwendern, die sich der Implementierung optimaler Verfahren der Softwaresicherung widmen. Es gibt umfangreiche dokumentierte Methoden und Verfahren zur Softwaresicherung für Softwareentwicklung und -anwendungen [28], die in diesem Standard nicht ausgearbeitet werden.

Anhang A (informativ)

Softwareorganisationen und Kategorisierung der Software und Anwendungen

A.1 Softwareorganisationen

Softwareorganisationen sind entsprechend ihren geschäftlichen Bedürfnissen und Anwendungsumgebungen je nach den verschiedenen Kombinationen von Entwicklung, Betrieb und Dienstleistung strukturiert.

Typische Softwareorganisationen sind solche, die

- a) in erster Linie Software entwickeln;
- b) Hardwareprodukte (Geräte) mit eingebetteter Software entwickeln;
- c) Kunden Softwareunterstützungsdienste anbieten;
- d) Softwarenetze und Systeme betreiben und warten.

A.2 Kategorisierung von Software

A.2.1 Softwarekategorien

Zu den Softwarekategorien gehören relevante Softwareentwicklungs-Produkte und -Daten, die durch den Softwarerealisierungsprozess entstehen. Das Kategorisierungsschema liefert eine geordnete Kombination von Ansichten und Kategorien bezüglich Software [21].

Die Kategorie ergibt sich aus der Gruppierung der Software nach Merkmalen oder Eigenschaften. Einige typische Beispiele solcher Gruppierungen sind nachfolgend aufgeführt.

A.2.2 Merkmale

- *Betriebsmodus* – Kategorien, die sich über die vom Softwaresystem eingenommenen besonderen Verarbeitungstechniken oder -arten definieren, beispielsweise Echtzeit-, Stapel-, zeitgeteilte, parallele und gleichzeitige Verarbeitung
- *Umfang der Software* – Kategorien, die sich über den Umfang (beispielsweise KSLOC) oder die Komplexität (beispielsweise Datenfluss) der Software definieren und dann mit klein, mittel oder groß bzw. einfach oder komplex bezeichnet werden
- *Stabilität* – Kategorien, die sich über den immanenten Evolutionsaspekt oder die Stabilität bezüglich den Eigenschaften des Softwaresystems definieren; beispielsweise: ändert sich andauernd, ändert sich stufenweise, oder ändert sich wahrscheinlich nicht
- *Softwarefunktion* – Kategorien, die sich über die Art der Funktion definieren; beispielsweise Compiler, Verarbeitung geschäftlicher Transaktionen, Textverarbeitung, Steuerungssysteme
- *Sicherheit* – Kategorien, die sich über das Schutzniveau gegen nicht autorisierten Zugang, Prüfpfad, Programm- und Datenschutz definieren
- *Funktionsfähigkeit* – Kategorien, die sich über das Ausmaß der geforderten Funktionsfähigkeit definieren, beispielsweise Reifegrad, Fehlertoleranz und Wiederherstellbarkeit
- *Leistung* – Kategorien, die sich über die Softwareleistung definieren, also Kapazität, Durchsatz, Fertigstellungszeit oder Reaktionszeit
- *Sprache* – Kategorien, die sich über die Art der in erster Linie für die Software verwendete Programmiersprache definieren, beispielsweise traditionell (bspw. COBOL, FORTRAN), prozedural (bspw. C), funktional (bspw. Lisp), objektorientiert (bspw. C++)

A.2.3 Umgebung

- *Anwendungsbereich* – Kategorien, die sich über die Art oder die Klasse externer Systeme definieren, in denen die Software verwendet wird, beispielsweise elektronische Geschäftsvorgänge, Prozesssteuerung und Netzwerksysteme
- *Rechnersystem* – Kategorien, die sich über das jeweilige anvisierte Rechnersystem definieren, in denen die Software läuft, beispielsweise Mikroprozessor gesteuert, Großrechner und Echtzeitbetriebssystem
- *Anwenderklasse* – Kategorien, die sich über den Fähigkeitsgrad oder die Merkmalswerte der beabsichtigten Anwenderklasse definieren, beispielsweise als Neuling, etwas fortgeschritten oder Experte
- *Rechnerausstattung* – Kategorien, die sich über die Einschränkungen und Grenzen der Rechnerausstattung und Rechenressourcen definieren, beispielsweise benötigter Speicherplatz, erforderliche Festplatten und benötigte lokale Netze
- *Softwarekritizität* – Kategorien, die sich über die Anforderungen an die Produktintegrität (Unversehrtheit) definieren, beispielsweise nationale Sicherheit, organisatorische Sicherheit und Vertraulichkeit
- *Softwareproduktverfügbarkeit* – Kategorien, die sich über die Verfügbarkeit der Softwareprodukte definieren, beispielsweise handelsüblich, maßgeschneidert oder proprietäre Software

A.2.4 Daten

- *Datendarstellung* – Kategorien, die sich über die Art, den Typ und die Struktur der Daten definieren, beispielsweise relationale, indizierte, formatierte Datei
- *Softwaredatenverwendung* – Kategorien, die sich über die Art der Verwendung der beabsichtigten Softwaredaten definieren, beispielsweise als einzelner Nutzer, mehrfache sequentielle Nutzer

A.3 Softwareanwendungen

Software wird sehr breit und vielfältig eingesetzt und angewendet. Softwareanwendungen können im Allgemeinen wie folgt zusammengefasst werden.

Systemsoftware stellt die Infrastruktur zur Steuerung der Rechnerhardware bereit, so dass die Anwendungssoftware darauf laufen kann. Beispiele von Betriebssystemen sind Microsoft Windows, Mac OS und Linux.

Anwendungssoftware ist die Rechnersoftware, die die Ausführung einer bestimmten Aufgabe ermöglicht, beispielsweise Textverarbeitung, Tabellenkalkulation und Datenbankanwendungen.

Firmware ist die Software, die in programmierbaren Speichern in den Endanwendergeräten residiert zur internen Steuerung unterschiedlicher elektronischer Geräte, wie Fernsteuerungen, Taschenrechner, Mobiltelefone und Digitalkameras.

Middleware ist die Rechnersoftware, die Softwareelemente für vielfältige Anwendungen oder Dienstleistungen miteinander verbindet, wie Multiprocessing in verteilten Systemen und Webdienste.

Testware ist eine Teilmenge der Software, die den besonderen Zweck hat, Software zu testen und Tests zu automatisieren.

Programmiersoftware ist das Softwareentwicklungswerkzeug, welches es Softwareentwicklern erleichtern soll, andere Programme und Anwendungen zu schaffen, diese von Fehlern zu befreien, diese zu erhalten oder zu unterstützen. Beispiele sind Rechner-gestützte Softwareentwicklungswerkzeuge.

Malware ist schädliche Software, die es darauf abgesehen hat, einen Rechner ohne Zustimmung des Eigentümers zu infiltrieren.