Formale und stochastische Methoden zur Qualitätssicherung technischer Software

Peter Liggesmeyer

Fachgebiet Softwaresystemtechnik II: Qualitätsmanagement
Hasso-Plattner-Institut für Softwaresystemtechnik
Mathematisch-Naturwissenschaftliche Fakultät
Universität Potsdam
Peter.Liggesmeyer@hpi.uni-potsdam.de

Zusammenfassung. Technische softwareintensive Systeme besitzen oft Sicherheitsanforderungen und quantifizierte Verfügbarkeitsziele, die entsprechende Qualitätssicherungstechniken fordern. Die Eigenschaften formaler und stochastischer Techniken befriedigen einerseits die existierenden Anforderungen; andererseits besitzen sie in der industriellen Praxis der Softwareentwicklung keine wesentliche Verbreitung. Im folgenden werden eine Erweiterung einer formalen Technik im Hinblick auf die Analyse ausfallbehafteteter technischer Systeme sowie ein Ansatz zur stochastischen Analyse und Prognose von Zuverlässigkeitskennwerten vorgestellt. Beide Techniken sind umfassend automatisiert, um sowohl eine einfache Benutzbarkeit in der Praxis als auch die Anwendbarkeit auf umfangreiche Systeme zu gewährleisten.

1 Motivation

Durch das Eindringen von Software in technische Systeme erhalten diese Systeme eine hybride Realisierung. Das betrifft oft sowohl ihren Aufbau aus unterschiedlich ausgeführten Bestandteilen, wie auch ihre Entwicklung in getrennten Entwicklungsteams. Sowohl für die Hardware, wie auch für die Software gibt es systematische Entwicklungs- und Qualitätssicherungstechniken. Hardware-Qualitätssicherungstechniken liefern oft quantifizierte Ergebnisse. Für Software gilt das nicht in gleichem Maße. Bei der Entwicklung hybrider technischer Systeme ist die übliche getrennte Qualitätssicherung der Hardware- und der Software-Bestandteile nicht sinnvoll, da der Systemhersteller Eigenschaften für das Gesamtsystem angeben und garantieren muß, und die Einzelresultate in der Regel nicht zusammengeführt werden können. Auch hier gilt: Das Ganze ist mehr als die Summe seiner Teile. Insbesondere die hier betrachteten technischen Anwendungsbereiche sind kritisch. Sie sind oft durch explizite oder implizite Sicherheitsanforderungen charakterisiert. Darüber hinaus existieren häufig quantitative Zielvorgaben, z. B. für Zuverlässigkeit, und teilweise benötigen Systeme Zulassungen [23]. Aufgrund dieser Anforderungen sind oft formal vollständige bzw. stochastisch abgesicherte Aussagen über Systemeigenschaften erforderlich.

In Kapitel 2 wird eine Analyse der Situation durchgeführt. Ferner werden wichtige Techniken in diesem Bereich beschrieben.

Als Beispiel einer formalen Technik zur Analyse ausfallbehafteter technischer Systeme wird in Kapitel 3 ein Fehlerbaumgenerierungsverfahren auf Basis zustandsendlicher Beschreibungen [25] dargestellt. Weitere hier nicht dargestellte Fehlerbaumgenerierungsverfahren dienen zur Erzeugung von Fehlerbäumen aus Software-Quellcode, aus einer sogenannten FMECA *(Failure Mode, Effects and Criticality Analysis)* und aus Schaltungsbeschreibungen elektronischer Hardware [22].

In Kapitel 4 werden stochastische Zuverlässigkeitsanalysetechniken dargestellt. Im Rahmen der Forschungsarbeiten ist ein Werkzeug zur Durchführung derartiger Analysen realisiert worden, das Zuverlässigkeitsmodelle anhand einer vorliegenden Ausfallbeachtung bewerten und kalibrieren kann. Mit Hilfe des Werkzeugs können stochastisch abgesicherte Prognosen für die Zuverlässigkeit gestellt werden. Derzeit sind die verfügbaren Zuverlässigkeitsfunktionen schwerpunktmäßig für Software konzipiert. Dennoch kann das Verfahren unter bestimmten Voraussetzungen auch auf hybride Systeme angewandt werden. Es gibt bereits zahlreiche Anwendungen auf industrielle Entwicklungsprojekte. Die empirischen Ergebnisse aus diesen Anwendungen werden ebenfalls in Kapitel 4 dargestellt.

Kapitel 5 enthält Schlußfolgerungen.

2 Stand der Technik

2.1 Zur Situation

Software dringt zunehmend in kritische Anwendungsbereiche ein, z. B. Medizintechnik und Verkehrstechnik. Gleichzeitig steigen der Umfang und die Komplexität von Software-Systemen erheblich an, was ein Gefahrenpotential aufgrund der Schwierigkeit erzeugt, diese Systeme zu beherrschen. Es gibt bereits Berichte über zahlreiche Fehlverhalten von Systemen unter Software-Beteiligung (siehe auch [18, 22]).

Die Ursachen von Fehlverhalten in Systemen können sehr vielfältig sein. Die Beobachtung, daß Software-Systeme in der täglichen Benutzung meistens zufriedenstellend funktionieren, könnte Anlaß zu der Vermutung geben, daß es sich bei Ausfällen um Ausnahmen handelt.

Einerseits ist dies ist in bezug auf schwerwiegende Ausfälle glücklicherweise korrekt. Andererseits zeigen empirische Daten, daß Fehler in Software-Systemen nicht die Ausnahme, sondern der Regelfall sind. Eine amerikanische Untersuchung [30, S. 118], die sich auf etwa 130 Projekte stützt, kommt zu dem Ergebnis, daß die durchschnittliche Software zu Beginn des Modultests rund 19,7 Fehler pro Tausend Quellcodezeilen des Programmtextes enthält. Zu Beginn des Systemtests verbleiben rund 6 Fehler pro Tausend Quellcodezeilen in dem Produkt. Freigegeben werden Softwareprodukte nach dieser Untersuchung mit rund 1,5 Fehlern pro Tausend Quellcodezeilen.

Auf den ersten Blick scheint das ein niedriger Wert zu sein. Verdeutlicht man sich aber, daß umfangreiche Software heute oft einige Millionen Quellcodezeile Umfang besitzt, so bedeutet das, daß derartige Produkte unter Umständen Tausende von Fehlern enthalten können.

Von Jones [17, S. 142-143] veröffentlichte Daten zeigen einen deutlichen relativen Anstieg des Aufwands für die analytische Qualitätssicherung in Abhängigkeit des

Gesamtentwicklungsaufwands eines Projektes. Die Codierung von Software stellt nach diesen Daten bei sehr kleinen Projekten den größten Aufwandsanteil dar. Bei dem größten in [17] aufgeführten Projekt erfordert die analytische Qualitätssicherung 37 % des gesamten Entwicklungsaufwands, während die Codierung nur noch 12 % benötigt.

Betrachtet man all diese Daten gemeinsam, so ist klar zu erkennen, daß in vielen Fällen Qualitätssicherung den größten Teil des Entwicklungsaufwands verschlingt. Andererseits werden aber oft keine befriedigenden Ergebnisse erzielt. In der Praxis führt das zu der unbefriedigenden Situation eines erheblichen Fehlleistungsaufwands.

In einer Untersuchung von Möller [29] sind Daten zu Fehlerkosten publiziert, die für Fehlerkorrekturen in den Phasen Analyse, Entwurf und Codierung Korrekturkosten in Höhe von 500 DM ausweisen. Beim Entwicklertest (Modultest) ist mit Korrekturkosten von 2000 DM zu rechnen. Die Korrektur eines Fehlers, der beim Systemtest gefunden wird, kostet rund 6000 DM. Fehler, die während der Produktnutzung erkannt werden, verursachen im Mittel Korrekturkosten von 25000 DM. Dieser nahezu exponentielle Anstieg der Korrekturkosten erfordert aus wirtschaftlichen Gründen eine frühzeitige Qualitätssicherung.

Spillner und Liggesmeyer [31] haben Daten zur Nutzung von Software-Qualitätssicherungstechniken in der Praxis publiziert, die zeigen, daß die Techniken *Reviews* und dynamisches Testen in der Praxis eine starke Verbreitung besitzen, während formale Techniken nur in geringem Umfang genutzt werden.

Insgesamt existiert zwischen dem Wissen über Qualitätssicherungstechniken in der Theorie und der Nutzung dieser Techniken in der Praxis eine erhebliche Diskrepanz.

2.2 Techniken zur Qualitätssicherung softwareintensiver, hybrider Systeme

Es existieren nur wenige Techniken, die auf hybride Systeme aus mechanischen und elektronischen Komponenten mit Softwarebeteiligung anwendbar sind. Dies gilt insbesondere bei Systemen, die sicherheitskritisch sind, für die quantifizierte Zuverlässigkeitsaussagen gefordert sind oder die Echtzeiteigenschaften besitzen.

Die FMECA [11] ist eine Technik, die für mechanische und elektronische Komponenten und Systeme eine hohe Verbreitung in der Praxis besitzt. Sie kann eingeschränkt zur Sicherheitsanalyse und für Zuverlässigkeitsbetrachtungen verwendet werden, liefert aber z. B. keine echten quantifizierten Zuverlässigkeitskennwerte. Analysen von Echtzeiteigenschaften sind nicht direkt Gegenstand der FMECA. Die FMECA setzt voraus, daß die Ausfallmodi der betrachteten Komponenten bekannt sind oder mindestens begründete Annahmen gemacht werden können. Bei der Analyse von Hardware ist das möglich. Bei mechanischen Komponenten kann das z. B. der Bruch einer Achse sein. Bei elektronischen Komponenten ist der Kurzschluß eines Bauteils ein Beispiel. Es ist schwierig, für Software ähnlich sinnvolle Annahmen zu machen, da die Art der Ausfälle anders beschaffen ist. Die FMECA ist daher für die Softwarebestandteile von Systemen nur eingeschränkt verwendbar. Die Analyse von Ausfallmechanismen innerhalb eines Softwaremoduls ist kaum möglich. Die Untersuchung der Ausfallfortpflanzung in einer Softwarearchitektur ist möglich, da aufgrund des Systementwurfs die Schnittstellen der Softwarekomponenten bekannt sind. Die Fehlermodi einer Komponente sind durch die unterschiedlichen ausgangsseitigen

Fehlermöglichkeiten gegeben. Das Ziel der FMECA ist die Untersuchung, welche Wirkungen diese Fehlverhalten auslösen. Das entspricht einer System-FMECA des Software-Subsystems, die als Bestandteil der System-FMECA des hybriden Systems genutzt werden kann.

Zuverlässigkeitsblockdiagramme [13] gestatten die Berechnung von Zuverlässigkeitskennwerten eines Systems aus den Zuverlässigkeiten der Komponenten und ihren Wirkzusammenhängen, die bei Hardware oft aus der Systemarchitektur abgeleitet werden können. Da die Wirkzusammenhänge bei Software oft kompliziert sind, ist die Anwendbarkeit auf Software gering. Außerdem verfügt Software im allgemeinen über vielfältige Ausfallmechanismen, die unterschiedliche Wirkungen besitzen. Derartige Eigenschaften können in Zuverlässigkeitsblockdiagrammen nicht berücksichtigt werden. Zuverlässigkeitsblockdiagramme können zur Analyse der Zuverlässigkeit eines Systems aus Hardware- und Softwarekomponenten verwendet werden, falls die Software bei der Analyse nicht in ihre Komponenten aufgelöst wird. Über Sicherheits- und Echtzeiteigenschaften machen Zuverlässigkeitsblockdiagramme keine Aussage.

Fehlerbäume [12] sind eine zur Sicherheits- und Zuverlässigkeitsanalyse von Hardwaresystemen verbreitete Technik [20, 34]. Es gibt Vorschläge, diese Technik zur Analyse von Software zu verwenden [21]. Fehlerbäume sind gleichzeitig formal und anschaulich. Sie gestatten die Ermittlung quantifizierter Zuverlässigkeitskennwerte.

Markov-Modelle [14] sind eine zur Zuverlässigkeits- und Leistungsanalyse von Systemen verbreitete Technik. Ihre Anwendung auf umfangreiche Systeme ist kritisch, da Zustandsautomaten bei einer großen Anzahl von Zustandsvariablen oft aufgrund ihres Umfangs nicht mehr handhabbar sind. Es gibt Versuche, Markov-Modelle auch für Software zu verwenden [2].

In der Software-Entwicklung werden verbreitet dynamische Testtechniken eingesetzt. Diese sind auch auf hybride Systeme anwendbar. Sie können Einflüsse der Betriebsumgebung berücksichtigen. Das ist insbesondere bei harten Echtzeitanforderungen wichtig. Man verwendet spezielle Testsysteme; sogenannte *Hardware-in-the-Loop*-Systeme. Aussagen zur Zuverlässigkeit sind durch dynamisches Testen nur mit Hilfe von stochastischen Verfahren möglich. Dies ist ein aktueller Gegenstand von Forschungsarbeiten. Zuverlässigkeitsanalysen mit Hilfe dynamischer Tests sind bei hochzuverlässigen Systemen nicht möglich, falls Ausfallereignisse für eine statistische Auswertung zu selten auftreten. Sicherheitsanalysen können durch dynamisches Testen kaum erbracht werden.

Statistische Analysen des Ausfallverhaltens sind bei Hardwaresystemen verbreitet [33]. Auch für Software existieren zahlreiche Ansätze (siehe z. B. [27]). Offene Fragen existieren im Bereich der Analyse von hybriden Systemen sowie in bezug auf die praktische Anwendung der umfangreichen Theorie.

Formale Techniken zur Spezifikation und Verifikation sind für Hardware und Software bekannt. Eine breite Anwendung in der Praxis finden diese Ansätze jedoch noch nicht. Es gibt auch formale Techniken für die Verifikation hybrider Systeme. Es wird in der Regel angenommen, daß die Hardware-Komponenten des betrachteten Systems ausfallfrei funktionieren. Diese Annahme ist für Sicherheits- und Zuverlässigkeitsanalysen nicht akzeptabel.

3 Eine Technik zur formalen Analyse ausfallbehafteter Systeme

3.1 Automatisierung von Fehlerbaumanalysen

Im folgenden wird eine Technik zur Generierung von Fehlerbäumen [9, 12] auf Basis einer zustandsendlichen Beschreibung dargestellt. Fehlerbäume sind zur Analyse umfangreicher hybrider Systeme aus den folgenden Gründen eine geeignete Technik:

- Fehlerbäume sind modular aufgebaut und unterstützen daher eine wichtiges Prinzip zur Beherrschung großer Systeme. Darüber hinaus ist es möglich, für unterschiedlich realisierte Systemkomponenten (z. B. Elektronik, Software) getrennte Fehlerbaumanalysen durchzuführen und anschließend die Fehlerbäume entsprechend der Wirkzusammenhänge der Komponenten zusammenzuführen.
- Fehlerbäume sind eine formale Technik, die quantifizierte Ergebnisse liefert.
- Auch sehr große Fehlerbäume können noch quantitativ ausgewertet werden.

Problematisch bei der Nutzung von Fehlerbäumen zur Analyse großer Systeme ist der hohe Erstellungsaufwand und die zu erwartende Unvollständigkeit des Analyseergebnisses. Beides kann durch Automatisierung von Fehlerbaumanalysen vermieden werden.

Während der Entwicklung eines Systems entstehen Dokumente, die Informationen über Abläufe oder die Struktur enthalten. In frühen Phasen sind dies z. B. Spezifikationen und Architekturbeschreibungen, die in späteren Phasen z. B. um Software-Quellcode ergänzt werden. Diese Dokumente tragen neben der Information über das Verhalten des Systems im Normalbetrieb implizit Information über das Verhalten des Systems im Fehlerfall. So reagiert z. B. eine Steuerung mit dem ihr eingeprägten Verhalten auch auf fehlerhafte Daten. Die möglichen gefährlichen Situationen können basierend auf diesen Beschreibungen automatisch und vollständig bestimmt werden.

Architekturbeschreibungen und strukturierte Spezifikationen erklären, wie sich Systeme aus Komponenten zusammensetzen. Man kann diesen Beschreibungen entnehmen, wie sich Fehlersituationen in dem System ausbreiten können.

Ziel von Generierungsverfahren ist, die in derartigen, bereits existierenden Dokumenten enthaltene Information für eine automatisierte Sicherheits- und Zuverlässigkeitsanalyse zu verwenden. Generierungsverfahren müssen vorhandene Informationen über die statische Struktur und das dynamische Verhalten des Systems möglichst umfassend nutzen. Darüber hinaus ist es sinnvoll, vorhandene Analysen systematisch wiederzuverwenden. Ferner können generierte Fehlerbäume jederzeit manuell modifiziert und ergänzt werden.

3.2 Fehlerbaumgenerierung auf Basis zustandsendlicher Beschreibungen

3.2.1 Motivation

Es ist möglich, daß eine korrekte Software in bezug auf eine gegebene Menge von Systemanforderungen sehr instabil ist, wenn Systemkomponenten Ausfälle zeigen. In diesem Fall besitzt die Software eine niedrige Robustheit. In der Praxis wird die Robustheit normalerweise durch Streßtests ermittelt. Das ist nachteilig, weil die Ergebnisse niemals vollständig sind und es notwendig ist, daß das System bereits verfügbar ist, um die Tests auszuführen. Da die Tests in der Regel zu einem späten Zeitpunkt

der Systementwicklung durchgeführt werden, sind Softwareänderungen zur Verbesserung der Systemrobustheit kaum noch durchführbar.

Eine durch einen Hardwareausfall verursachte Verletzung von Sicherheitsanforderungen ist in vielen Systemen kritisch. Daher benötigen die Entwickler der Steuerungssoftware eine Technik zur Bewertung der Robustheit. Ein formaler Korrektheitsnachweis für den ausfallfreien Fall ist nur eine Teillösung. Ergänzend ist eine formale Analyse der Software-Robustheit notwendig.

3.2.2 Lösungsansatz

Das im folgenden beschriebene Fehlerbaumgenerierungsverfahren führt eine automatisierte Analyse des Einflusses von Fehlern der Sensoren auf das Verhalten von Systemen durch. Es wird ermittelt, welche Fehlverhalten bzw. Ausfälle der Sensoren zur Verletzung geforderter Eigenschaften eines Systems (z. B. Sicherheitsanforderungen) führen können und wie hoch die Wahrscheinlichkeit für den Eintritt dieses Ereignisses ist. Das Generierungsverfahren läuft – mit Ausnahme der manuellen Festlegung des betrachteten unerwünschten Ereignisses – vollautomatisch ab und erfordert keine zusätzliche Modellerstellung.

Das System besteht aus einer Steuerung, die die Steuerungssoftware enthält, und der gesteuerten Anlage. Weiterhin sind Festlegungen der Systemanforderungen und des betrachteten Ausfallmodells erforderlich.

Die Steuerungssoftware muß als endlicher Automat realisiert sein. Programmier- und Spezifikationssprachen auf Basis endlicher Automaten werden in der Industrieautomatisierung häufig verwendet (z. B. [5, 10, 15, 19]).

Das System, als Kombination aus Steuerungssoftware und gesteuertem Prozeß, kann im Hinblick auf die Erfüllung von Systemanforderungen verifiziert werden. Dies ist ein formaler Beweis von Systemeigenschaften durch symbolisches *Model Checking* [4, 6, 28]. Typischerweise wird die Erfüllung von Sicherheits- und Lebendigkeitsanforderungen geprüft. Die Ergebnisse dieses Nachweises sind nur gültig, wenn der gesteuerte Prozeß und die Schnittstelle zwischen dem Prozeß und der Steuerung ausfallfrei funktioniert. In der Praxis gilt das jedoch nicht verläßlich, da sowohl der gesteuerte Prozeß als auch die Schnittstelle zwischen Prozeß und Steuerung Hardwarekomponenten enthalten, die eine gewisse Ausfallwahrscheinlichkeit besitzen. Deshalb ist es möglich, daß Systemanforderungen verletzt werden, obwohl sie durch einen formalen Beweis bestätigt wurden.

Der im Rahmen der Forschungsarbeiten entwickelte Werkzeugprototyp verarbeitet Steuerungsbeschreibungen in der zustandsautomatenbasierten Sprache CSL *(Control Specification Language)*. Der gesteuerte Prozeß verhält sich im allgemeinen nichtdeterministisch und wird mit einem entsprechenden nicht-deterministischen Zustandsautomaten dargestellt. Die Prozeßspezifikation schränkt die Reaktionen des gesteuerten Prozesses auf die physikalisch sinnvollen Reaktionen ein. Im wesentlichen drückt die Prozeßspezifikation aus, daß der Prozeß bestimmten physikalischen Gesetzen gehorcht. Die zu prüfenden Eigenschaften des Systems (z. B. Lebendigkeits- und Sicherheitsanforderungen) können mit Hilfe temporaler Logik [7, 8] formuliert werden. Die Analyse wird in zwei Hauptschritten durchgeführt:

- Formaler Korrektheitsnachweis des ausfallfreien Systems gegen gegebene Systemanforderungen.

- Formale Analyse des ausfallbehafteten Systems gegen die gleichen Systemanforderungen.

Bei der formalen Analyse des ausfallbehafteten Systems werden Systembeschreibungen *und* fest gewählten Ausfallarten verwendet. Es ist möglich, Einzel- und Mehrfachausfälle sowie intermittierende und dauerhafte Ausfälle zu modellieren. Der vorliegende Werkzeugprototyp analysiert die Einflüsse von Sensorausfällen auf die Erfüllung von Sicherheitsanforderungen eines technischen Systems. Dies wird folgendermaßen durchgeführt:

- Bestimmung aller möglichen Ausfälle aller Sensoren auf Basis des Ausfallmodells, z. B. intermittierende Einzelausfälle. Für ein gegebenes Ausfallmodell kann dies unter Nutzung von Informationen, die in der Software bereits vorhanden sind, automatisiert werden.
- Generierung eines Zustandsautomaten, der das Systemverhalten in Gegenwart des betrachteten Ausfalls beschreibt. Dies wird für alle Ausfälle aller Sensoren durchgeführt.
- Analyse, ob die ausfallbehafteten Systeme noch die Systemanforderungen erfüllen. All jene Situationen, die die Anforderungen verletzen, werden bestimmt.
- Für jede Systemanforderung Generierung eines erweiterten Fehlerbaums, der den Zusammenhang zwischen Ausfällen und hervorgerufener Wirkung beschreibt.

Es gibt verschiedene Möglichkeiten, die Sensorausfälle in der Systembeschreibung zu modellieren. Für intermittierende Sensorausfälle werden diese im folgenden erörtert. Falls ein intermittierender Ausfall des Sensors *X* auftritt, so daß *X* sporadisch x_1 anstelle von x_0 signalisiert, so kann dies durch Hinzufügen von Übergängen zur zustandsendlichen Beschreibung des Systems modelliert werden. Abb. 1a zeigt ein einfaches Beispiel, das den Zustandsautomaten der Software und eine zustandsendliche Beschreibung des gesteuerten Prozesses enthält. Die Zustandsautomaten können asynchron Zustandsübergänge auslösen. Der Sensor *X* und der Aktuator *Y* bilden die Schnittstelle. Sensor *X* liefert der Software Eingaben x_j. Aktuator *Y* wandelt Softwareausgaben in Prozeßeingaben um. Der Anfangszustand des Systems ist (x_0, y_0). Dies ist die Kombination der Anfangszustände der Steuerungssoftware und des gesteuerten Prozesses. Aufgrund der Verarbeitungslogik des Systems sind nur einige Kombinationen der Zustände der Steuerungssoftware und des Prozeßstatus erreichbar.

Abb. 1b zeigt ein explizites Modell des Ausfalls *Sensor X signalisiert intermittierend* x_1 *statt* x_0. Dieses explizite Ausfallmodell verursacht eine Zunahme des Rechenzeitbedarfs der Analyse, da die Anzahl der Zustände ansteigt. Eine alternative Beschreibungsform, die die Zunahme des Umfangs vermeidet, ist durch Ergänzen von Transitionen im Zustandsautomaten möglich. Der beschriebene Ausfall kann durch Hinzufügen des fetten, gestrichelten Übergangs dargestellt werden (Abb. 1c).

Die zur Darstellung der Ausfälle erforderlichen Modifikationen der zustandsendlichen Beschreibung des Systems können automatisiert durchgeführt werden, da die erforderlichen Informationen in der Systembeschreibung enthalten sind. Der Softwareentwickler kann diese Technik verwenden, um Änderungen der Software zu identifizieren, die die Robustheit in bezug auf Sensorausfälle steigern. Eine detaillierte Darstellung des Verfahrens ist in [22] enthalten.

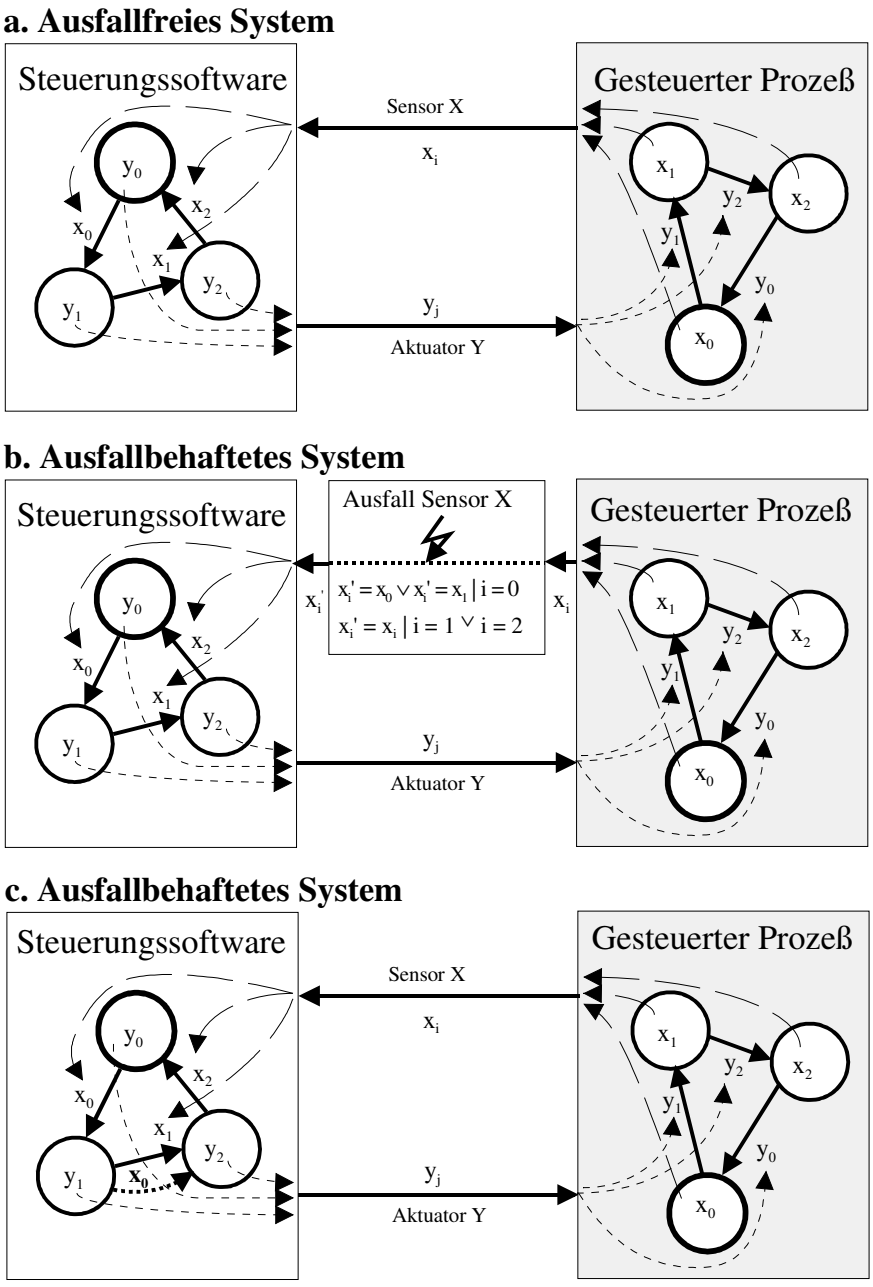

Fig. 1. Ausfallmodellierung

4 Stochastische Zuverlässigkeitsanalyse

4.1 Motivation und Ziele

Zuverlässigkeit ist eine stochastische Eigenschaft eines betrachteten Systems. Sie wird durch die zeitliche Verteilung des Eintretens von Fehlverhalten und Ausfällen bestimmt. Für die Hardwarebestandteile eines Systems wird oft davon ausgegangen, daß nur spontane Ausfälle aufgrund von Fertigungsfehlern, Verschleiß oder äußeren Einflüssen – z. B. Strahlung, Temperatur – auftreten. Systematische Konstruktionsfehler werden als Quelle von Fehlverhalten häufig vernachlässigt. Bei Softwarebestandteilen sind die Fehler, die die Ausfälle verursachen, statisch vorhanden. Spontane Ausfälle einer fehlerfreien Software sind nicht möglich. Der zufällige Eintritt von Fehlverhalten bei Software kommt durch die in der Regel vielfältigen Benutzungsmöglichkeiten einer Software zustande. Diese können als ein Ziehungsprozeß verstanden werden, der Fehlverhalten hervorbringt. Daher beeinflußt bei Software die Intensität und die Art der Benutzung die Häufigkeit, mit der Ausfälle auftreten, und damit auch die Zuverlässigkeit.

Quantifizierte Zuverlässigkeits- bzw. Verfügbarkeitsziele von Systemen fordern ein Verfahren, das eine verläßliche Messung und Prognose dieser Eigenschaften ermöglicht.

Die Durchführung stochastischer Zuverlässigkeitsanalysen für reine Hardware-Systeme ist Stand der Technik [3, 32, 33]. Software-Zuverlässigkeitsmodellierung ist etwa seit Beginn der 70er Jahre eine eigenständige Disziplin. Eine aktuelle Übersicht des Themas ist in [27] enthalten. Es sind viele stochastische Zuverlässigkeitsmodelle vorgeschlagen worden, die unterschiedlichen Kategorien angehören. Neben der Unterscheidung von Modellen mit endlicher und unendlicher Ausfallanzahl können elementare Modelltypen unterschieden werden. Verbreitete Modelle sind z. B. das elementare Ausführungszeiten-Modell von Musa [30], das Jelinski-Moranda-Modell [16] und das Littlewood-Verrall-Modell [26]. Darüber hinaus werden für Software auch klassische Modelle für Hardware-Zuverlässigkeit (z. B. Weibullverteilungen) verwendet.

Die Einschätzung der Eignung der Modelle ist aufgrund der Modellvielfalt und ihrer spezifischen Voraussetzungen für den Anwender kompliziert. Die Eignung eines Modells in bezug auf eine vorliegende Ausfallbeobachtung kann z. B. mit dem sogenannten *U-Plot*- oder dem *Prequential-Likelihood*-Verfahren sowie durch *Holdout*-Bewertung beurteilt werden. Darüber hinaus ist neben der Bewertung der grundsätzlichen Eignung eines Modells eine Festlegung der Modellparameter erforderlich. Dies kann mit Hilfe des Verfahrens der kleinsten Fehlerquadrate oder mit dem *Maximum-Likelihood*-Verfahren durchgeführt werden (siehe z. B. [22]).

Die Erfahrung zeigt, daß der kritische Aspekt der Software-Zuverlässigkeitsanalyse die Anwendung der mathematischen Verfahren ist. Es ist wichtig, daß Zuverlässigkeit leicht gemessen und prognostiziert werden kann, ohne daß der Anwender den theoretischen Hintergrund vollständig verstehen muß. Das im Rahmen der Forschungsarbeit entwickelte Werkzeug RAT *(Reliability Assessment Tool)* unterstützt verschiedene Software-Zuverlässigkeitsmodelle und enthält Techniken zur Modell-Auswahl und -Kalibrierung basierend auf beobachteten Ausfällen.

RAT wird in zahlreichen System-Entwicklungen angewendet [24]. Die Anwendungsbereiche sind u. a. Telekommunikationssysteme, verkehrstechnische Systeme und Anlagen, Medizintechnik, militärische Systeme, Kraftwerke und Industrieautomation. Der Umfang dieser Systeme variiert stark. Obwohl RAT zur Zeit nur Modelle enthält, die für die Modellierung von Software-Zuverlässigkeit vorgeschlagen wurden, ist das Werkzeug auch auf hybride Systeme angewendet worden, um zu prüfen, ob und wie derartige Zuverlässigkeitsmodelle für Systeme benutzt werden können, bei denen Software- und Hardware-Komponenten betrachtet werden.

4.2 Werkzeugunterstützte Zuverlässigkeitsanalyse

4.2.1 Das Zuverlässigkeitsanalysewerkzeug RAT

Um eine gewisse Breite von Situationen abzudecken, enthält RAT mehr als 10 unterschiedliche Zuverlässigkeitsmodelle, die von verschiedenen Autoren vorgeschlagen worden sind. Jedes dieser Modelle kann als nicht-homogener Poissonprozeß [35] ausgedrückt werden. Die Modellkalibrierung wird mit Hilfe des bereits dargestellten Verfahrens der minimalen Fehlerquadrate oder durch *Maximum-Likelihood*-Anpassung durchgeführt. Die Beurteilung der Modelleignung erfolgt auf Basis des *U-Plot*-Kriteriums, des *Prequential-Likelihood*-Verfahrens und der *Holdout*-Bewertung (siehe auch [1]). Abb. 2 zeigt die Benutzungsoberfläche des Werkzeugs, die eine intuitive, unkomplizierte Nutzung dieser Techniken ermöglicht.

4.2.2 Einsatzerfahrungen

Die Erfahrungen zum RAT-Einsatz stammen aus einer Vielzahl von Anwendungsbereichen, Umfängen, Systemtypen und Lebenszyklusphasen. Normalerweise existierten die Ausfalldaten schon, die verwendet werden, um die Zuverlässigkeitsmodelle auszuwählen und zu kalibrieren. Sie wurden nicht ausdrücklich für diesen Zweck erfaßt. Dies hat bestimmte Vorteile und Nachteile:

+ Da die gesamten Ausfalldaten bereits verfügbar sind, kann die Verläßlichkeit von Prognosen durch eine *Holdout*-Bewertung leicht geprüft werden.
+ Die Datenerfassung erfordert keinen Zusatzaufwand.
+ Die Modellierung kann unmittelbar begonnen werden.
- Es ist nicht möglich, die Datenerfassung zu beeinflussen.
- Normalerweise basieren die Ausfalldaten auf Kalenderzeit.
- Die Daten sind nicht immer geeignet strukturiert (z. B. ungeeignete Ausfallklassen).
- Fehlende Information nachträglich zu erfassen, ist nicht immer möglich und erfordert zusätzlichen Aufwand.

Obwohl das Werkzeug nicht fordert, daß für die Modelle eine Vorauswahl getroffen wird, zeigt die Erfahrung, daß eine Vorauswahl in einigen Situationen günstig ist. Die Auswahl bestimmter Modellklassen produziert oft bessere Ergebnisse. In einigen Fällen ist es z. B. möglich, a priori zu entscheiden, ob ein Modell mit einer begrenzten

Ausfallanzahl angemessen ist oder ob ein Modell mit einer unendlichen Ausfallanzahl verwendet werden sollte.

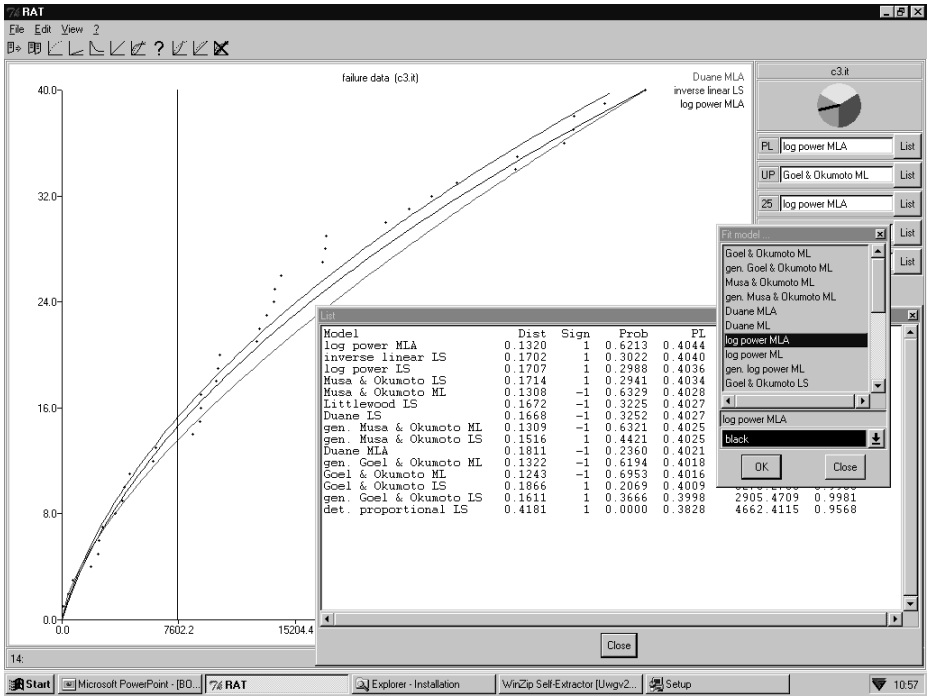

Fig. 2. Benutzungsoberfläche des Werkzeugs RAT

RAT unterstützt 3 verschiedene Auswahlkriterien für Zuverlässigkeitsmodelle: *Prequential-Likelihood*-, *U-Plot*- und *Holdout*-Bewertung. Bei der *Holdout*-Bewertung kann das Verhältnis der zur Modellanpassung und zur Bewertung der Voraussagequalität verwendeten Datenpunkte vom Benutzer ausgewählt werden. Es ist auf diese Weise möglich, mit Langfrist- oder Kurzfristprognosen zu experimentieren.

In den hier zugrundeliegenden Anwendungen zeigte sich eine Kombination der Kriterien besonders geeignet. RAT ordnet die vorausgewählten Modelle in bezug auf jedes der Auswahlkriterien. Ein Eignungsmaß kann verwendet werden, um festzustellen, ob zwei benachbarte Modelle eine ähnliche Eignung besitzen. Wenn jedes Auswahlkriterium ein anderes optimales Zuverlässigkeitsmodell identifiziert, dann können diese Modellbewertungen verwendet werden, um zu prüfen, ob ein gewisses Modell nach allen Kriterien relativ gut geeignet ist. Modelleignung ist ein unscharfes Merkmal. In den meisten praktischen Fällen existiert *das* optimale Modell nicht. Die Ursachen können z. B. verrauschte Ausfalldaten sein. Wenn mehrere Modelle ähnlich gut geeignet sind, so können sie alle im Sinne einer *best case- / worst case*-Bewertung verwendet werden. In der Kombination der Auswahlkriterien sind nicht alle Kriterien gleichgewichtig. *Prequential-Likelihood* hat sich als sehr vertrauenswürdiges Kriterium erwiesen und wird daher stärker gewichtet.

Die Qualität der Ausfalldaten ist der wichtigste Einflußfaktor für die Verläßlichkeit der Zuverlässigkeitsanalysen. Häufig sind die Daten verrauscht oder enthalten systematische Störungen. Dennoch ist es oft möglich, diese Daten als Basis für Zuverlässigkeitsanalysen zu verwenden. Dies erfordert eine Analyse und Korrektur der Ausfalldaten. Das umfaßt die Entfernung von Instabilitäten, die oft zu Beginn von Ausfalldatensätzen auftreten. Obwohl Modelle existieren, die Einschwingverhalten beschreiben können, ist deren Verwendung nicht in jedem Fall angebracht. Wenn Einschwingvorgänge zu Beginn von Datensätzen klar erkennbar sind, ist es sinnvoll, diese Daten zu entfernen. Wenn der Testaufwand oder die Installationszahlen variieren, muß auch dies korrigiert werden. Testaufwände sind nicht immer eindeutig dokumentiert, was derartige Korrekturen behindert. In diesem Fall ist eine nachträgliche Bewertung des Testaufwands sinnvoll. Die Installationszahlen eines Systems werden während des Feldeinsatzes normalerweise erfaßt und können zur Korrektur der Ausfalldaten verwendet werden.

Die Verwendung von Ausfallprioritäten hat sich besonders in großen System-Entwicklungen bewährt. Dort ist der Test oft durch die verfügbaren Ressourcen limitiert. Daher konzentrieren sich die Tester zu Testbeginn oft auf Ausfälle mit hoher Priorität. Wenn das System in bezug auf solche Ausfälle stabiler wird, werden Ressourcen frei, um auch Ausfälle mit niedrigerer Priorität zu bearbeiten. Es kommt vor, daß bei nahezu unveränderter Ausfallhäufigkeit des Systems die Häufigkeit der Ausfälle mit hoher Priorität abnimmt, was durch eine Häufigkeitszunahme der Ausfälle mit niedriger Priorität kompensiert wird. Die Gesamtausfälle sind in einem derartigen Fall für Zuverlässigkeitsanalysen ungeeignet, da die Häufigkeit der Ausfälle mit niedriger Priorität nicht durch die Zuverlässigkeit des Systems bestimmt wird. Sie wird von den verfügbaren Testressourcen festgelegt. Entscheidend ist jedoch die Zuverlässigkeit in bezug auf Ausfälle mit hoher Priorität. Da deren Häufigkeit durch Veränderungen der Systemzuverlässigkeit bestimmt ist, können diese Ausfälle mit Hilfe von Zuverlässigkeitsmodellen prognostiziert werden. Voraussetzung ist, daß die Prioritäteneinteilung verläßlich ist. Die Erfahrung aus den Anwendungen des Werkzeugs RAT zeigt, daß die Verwendung funktionaler Ausfallklassen (z. B. *totaler Systemzusammenbruch*, *geringfügiges Fehlverhalten*) dies begünstigt.

Ausfalldaten werden in der Regel auf Basis von Kalenderzeit erfaßt. Dies ist nachteilig für die Zuverlässigkeitsanalyse, falls die Beziehung zwischen Kalenderzeit und Ausführungszeit nicht linear ist. Wenn der nichtlineare Zusammenhang einfach ist, können die Daten entsprechend korrigiert werden. Ein Beispiel für eine derartige einfache Nichtlinearität sind Muster in den Ausfalldaten, die durch den Wechsel von Arbeitstagen und Wochenenden zustande kommen. Falls an den Wochenenden keine Tests durchgeführt werden, so sollten die Wochenenden aus dem Datensatz entfernt werden. Aber dies muß mit Vorsicht durchgeführt werden, weil insbesondere am Ende des Systemtests in Folge von Zeitdruck möglicherweise Tests auch am Wochenende durchgeführt werden. Ein anderes Beispiel für Zeiten ohne Tests können Feiertage sein. Veränderte Testintensitäten sind nicht immer einfach erkennbar.

Bei der Verwendung von Ausfalldaten aus dem Feld zur Zuverlässigkeitsmodellierung ist die Korrektur veränderlicher Installationszahlen wichtig. Diese sind normalerweise verfügbar. Die Erfahrung zeigt, daß, wenn die Installationszahlen hinreichend groß sind, das operationale Profil sehr einförmig ist. Daher ist es in diesen

Fällen nicht notwendig, die Ausfalldaten im Hinblick auf das operationale Profil zu korrigieren.

Zeit ist nicht immer die geeignetste Skala. Bei einer der genannten Anwendungen (ein Zugsystem) wurden Ausfalldaten auf zurückgelegte Distanzen bezogen. Für dieses System war diese Skala geeigneter als Zeit.

In vielen Anwendungen ist die Software nur ein Bestandteil eines Systems. Es ist daher notwendig, Zuverlässigkeitsaussagen für Systeme zu erzeugen, die Software- und Hardwarekomponenten enthalten. Weiterhin ist es in einigen Fällen unmöglich, einen Ausfall eindeutig als Hardware- bzw. Softwareausfall zu identifizieren. Im Rahmen der durchgeführten Forschungsarbeiten wurden die ursprünglich für Software entwickelten Zuverlässigkeitsmodelle auf hybride Hardware-/Software-Systeme angewandt, um zu prüfen, ob dies verwendbare Ergebnisse produziert. Es wurde erwartet, daß dieser Ansatz schlecht funktionieren würde, weil nur sehr wenige Software-Zuverlässigkeitsmodelle die für Hardware typischen Verschlechterungen der Zuverlässigkeit durch Verschleiß beschreiben können. Überraschenderweise funktionierte der Ansatz jedoch zufriedenstellend. Verschleißeinflüsse sind bei einigen untersuchten Systemen von untergeordneter Bedeutung. Das hat die folgenden Ursachen:

Wenn die Hardware eines Systems neu ist, enthält sie Konstruktionsschwächen aufgrund ihrer hohen Komplexität. Das verursacht ähnliche Wirkungen wie Softwarefehler. Verschleißeinflüsse besitzen in den analysierten hybriden Systemen nur eine geringfügige Wirkung. Ursache ist die präventive Wartung. Viele Systeme (z. B. Transportsysteme) sind ein teures Investitionsgut. Verschleiß ist daher nicht akzeptabel. Software-Zuverlässigkeitsmodelle können verwendet werden, um solche Systeme zu modellieren, weil ihre fehlende Fähigkeit zur Beschreibung von Verschleißeinflüssen nicht störend wirkt.

5 Schlußfolgerungen

Formale und stochastische Techniken befriedigen den in der technischen Softwareentwicklung existierenden Bedarf nach hochwertigen Qualitätssicherungstechniken. Bei der Entwicklung von Systemen existiert keine einzelne stets optimal geeignete Technik, die in allen Anwendungsfällen optimale Ergebnisse liefert. Für die unterschiedlichen Teiltätigkeiten in der Software- und der Hardware-Entwicklung sind jeweils eigene Vorgehensweisen üblich, die entsprechend passende Techniken in der Qualitätssicherung fordern. Eine einheitliche Technik ist daher in der praktischen Anwendung zum Scheitern verurteilt, weil in vielen Fällen ihre Voraussetzungen nicht erfüllt sind. Im Rahmen der hier in Ausschnitten dargestellten Forschungsarbeit wird ein anderer Weg beschritten. Ziel ist die getrennte Anwendung von Techniken für die unterschiedlichen Systembestandteile unter der Bedingung, daß die getrennt erzeugten Teilergebnisse zu einem gültigen Gesamtergebnis zusammengeführt werden können.

Dieser Ansatz ist eine wesentliche Voraussetzung für die Anwendbarkeit auf reale Systeme. Techniken, die zwingend verlangen, daß Systeme konsistent auf eine bestimmte Weise beschrieben sind, existieren in großer Anzahl in der akademischen Welt. Und dort werden sie bis auf wenige Ausnahmefälle auch solange bleiben, wie-

diese unrealistische und falsche Sicht der Realität aufrechterhalten wird. Das Ziel eines ingenieurmäßigen Ansatzes muß sein, ein hinreichend gut funktionierendes Verfahren zu verwenden, das mit den unterschiedlichen Rahmenbedingungen realer Systeme umgehen kann. Die hinreichend gute Funktion kann ebenfalls nicht einheitlich definiert werden, sondern wird von den spezifischen Eigenschaften des Systems bestimmt. In besonders kritischen Fällen, in denen eine hohe Verläßlichkeit und Präzision der Ergebnisse gefordert ist, müssen eventuell Voraussetzungen explizit geschaffen werden. In vielen anderen Fällen ist es möglich, Aussagen unterschiedlicher Qualität zu akzeptieren, solange klar ist, welche Ungenauigkeiten dabei zustande kommen.

Bei dem dargestellten Fehlerbaumgenerierungsverfahren ist klar definiert, welche Eigenschaften die erzeugten Fehlerbäume erfüllen, d. h. welche Ursachen betrachtet werden. Die hier beschriebenen stochastischen Zuverlässigkeitsanalysetechniken bieten die Möglichkeit zur quantitativen Ermittlung und Prognose von Zuverlässigkeitskennwerten (z. B. Ausfallanzahl, Ausfallrate) auf Basis beobachteter Ausfälle. Die Ergebnisse besitzen eine quantifizierte Verläßlichkeit.

Die Erfahrung aus der Anwendung der Techniken auf reale große Systeme aus unterschiedlichen Anwendungsbereichen zeigt, daß der flexible Umgang mit unterschiedlichen Voraussetzungen in bezug auf die Systemkomponenten die wesentlichste Voraussetzung für die Anwendbarkeit in der Praxis ist. Ist das nicht erfüllt, so beschränken sich die erzeugten Aussagen auf Teile des Systems. Dies verletzt die wichtige Forderung nach Qualitätsaussagen für Gesamtsysteme.

Ein sehr wichtiger Aspekt der hier dargestellten Techniken ist ihre umfassende Automatisierbarkeit. Selbst bei Systemkomponenten findet man heute derartig hohe Komplexitäten, daß eine verläßliche manuelle Analyse nicht möglich ist.

Literatur

1. Abdel-Ghaly A.A., Chan P.Y., Littlewood B., *Evaluation of competing software reliability predictions*, in: IEEE Transactions on Software Engineering, Vol. 12, No. 9, 1986, pp. 950-967
2. Beck A., Beer H., *ESSI Process Improvement Experiment 23843 - USST Usage Specification and Statistical Testing*, in: Proceedings EuroSTAR '98, Munich 1998, pp. 421-427
3. Birolini A., *Qualität und Zuverlässigkeit technischer Systeme: Theorie, Praxis, Management*, Berlin: Springer 1991
4. Bormann J., Lohse J., Payer M., Venzl G., *Model Checking in Industrial Hardware Design*, in: Proceedings of the 32st Conference on Design Automation, San Francisco, June 12-16, 1995, ACM Press 1995, pp. 298-303
5. Brandin B., *The Real-Time Supervisory Control of an Experimental Manufacturing Cell*, in: IEEE Transactions on Robotics and Automation, Vol. 12, No. 1, February 1996, pp. 1-14
6. Bryant R.E., *Graph-Based Algorithms for Boolean Function Manipulation*, in: IEEE Transactions on Computers, Vol. C-35, No. 8, August 1986, pp. 667-691
7. Burch J.R., Clarke E.M., Long D.E., McMillan K.L., Dill D.L., *Symbolic Model Checking for Sequential Circuit Verification*, in: IEEE Transactions on Computers, Vol. 13, No. 4, April 1994, pp. 401-424

8. Clarke E.M., Emerson E.A., Sistla A.P., *Automatic Verification of Finite state Concurrent Systems using Temporal Logic Specifications*, in: ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986, pp. 244-263
9. DIN 25424; DIN 25424-1, *Fehlerbaumanalyse Methoden und Bildzeichen*, September 1981; DIN 25424-2: *Fehlerbaumanalyse Handrechenverfahren zur Auswertung eines Fehlerbaumes*, April 1990; Berlin: Beuth Verlag
10. Heimdahl M.P.E., Leveson N.G., *Completeness and Consistency Analysis of State-Based Requirements*, in: Proceedings 17th International Conference on Software Engineering, Seattle, April 1995, pp. 3-14
11. IEC 812, *Analysis Techniques for System Reliability - Procedure for Failure Mode and Effect Analysis (FMEA)*, International Electrotechnical Commission 1985
12. IEC 61025, *Fault tree analysis (FTA)*, International Electrotechnical Commission 1990
13. IEC 61078, *Analysis techniques for dependability - Reliability block diagram method*, International Electrotechnical Commission 1991
14. IEC 61165, *Application of Markov techniques*, International Electrotechnical Commission 1995
15. Jaffe M.S., Leveson N.G., Heimdahl M.P.E., Melhart B.E., *Software Requirements Analysis for Real-Time Process-Control Systems*, in: IEEE Transactions on Software Engineering, Vol. 17, No. 3, March 1991, pp. 241-258
16. Jelinski Z., Moranda P.B., *Software reliability research*, in: Freiberger W. (Ed.), *Statistical Computer Performance Evaluation*, New York: Academic Press 1972, pp. 465-484
17. Jones C., *Applied software measurement*, New York: McGraw-Hill 1991
18. Leveson N.G., *Safeware: System safety and computers*, New York: Addison-Wesley 1995
19. Leveson N.G., Heimdahl M.P.E., Hildreth H., Reese J.D., *Requirements Specification for Process-Control Systems*, in: IEEE Transactions on Software Engineering, Vol. 20, No. 9, September 1994, pp. 684-707
20. Leveson N.G., Harvey P.R., *Analyzing Software Safety*, in: IEEE Transactions on Software Engineering, Vol. SE-9, No. 5, September 1983, pp. 569-579
21. Leveson N.G., Shimeall T.J., *Safety Verifcation of ADA Programs using Software Fault Trees*, in: IEEE Software, Vol. 8, No. 4, July 1991, pp. 48-59
22. Liggesmeyer P., *Qualitätsicherung softwareintensiver technischer Systeme*, Habilitationsschrift, Fakultät für Elektrotechnik und Informationstechnik, Ruhr-Universität Bochum, Heidelberg: Spektrum-Verlag 2000
23. Liggesmeyer P., Rothfelder M., Rettelbach M., Ackermann T., *Qualitätssicherung softwarebasierter Systeme: Problembereiche und Lösungsansätze*, in: Informatik-Spektrum, Band 21, Heft 5, Oktober 1998, S. 249-258
24. Liggesmeyer P., Ackermann T., *Applying Reliability Engineering: Empirical Results, Lessons Learned, and Further Improvements*, in: Proceedings ISSRE `98, The Ninth International Symposium on Software Reliability Engineering, Paderborn, November 1998, pp. 263-271
25. Liggesmeyer P., Rothfelder M., *Improving System Reliability with Automatic Fault Tree Generation*, in: Proceedings 28th Annual Fault Tolerant Computing Symposium, Munich, June 1998, pp. 90-99
26. Littlewood B., Verall J.L., *A Bayesian reliability growth model for computer software*, in: Applied Statistics, Vol. 22, No. 3, 1973, pp. 332-346
27. Lyu M.R., *Handbook of Software Reliability Engineering*, New York: McGraw-Hill 1995
28. McMillan K.L., *Symbolic Model Checking*, Norwell, Dordrecht: Kluwer Academic Publishers 1993
29. Möller K.-H., *Ausgangsdaten für Qualitätsmetriken - Eine Fundgrube für Analysen*, in: Ebert C., Dumke R. (Hrsg.), *Software-Metriken in der Praxis*, Berlin, Heidelberg: Springer 1996, S. 105-116

30.Musa J.D., Iannino A., Okumoto K., *Software Reliability: Measurement, Prediction, Application*, New York: McGraw-Hill 1987
31.Spillner A., Liggesmeyer P., *Software-Qualitätssicherung in der Praxis - Ergebnisse einer Umfrage*, in: Informatik-Spektrum, Band 17, Heft 6, Dezember 1994, S. 368-372
32.Störmer H., *Mathematische Theorie der Zuverlässigkeit*, München: Oldenbourg 1983
33.Tobias P.A., Trindade D.C., *Applied Reliability*, New York: Van Nostrand Reinhold 1995
34.Ulerich N.H., Powers G.J., *On-Line Hazard Aversion and Fault Diagnosis in Chemical Processes: The Digraph + Fault-Tree Method*, in: IEEE Transactions on Reliability, Vol. 37, No. 2, June 1988, pp. 171-177
35.Zhao M., *Software reliability models based on nonhomogenous poisson processes*, Thesis, Linköping University, 1991