universität wien

DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

„Wann fällt meine App aus?
Einführende mathematische Betrachtung von
Software-Zuverlässigkeit“

verfasst von / submitted by

Constanze Tamara Kunnert

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magistra der Naturwissenschaften (Mag.rer.nat)

Wien, 2019 / Vienna, 2019

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 190 406 333
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Lehramtsstudium UF Mathematik UF Deutsch
Betreut von / Supervisor:	ao. Univ.-Prof. Mag. Dr. Peter Raith

Entweder es ist perfekt
oder es ist fertig
Sprichwort

Danksagung

Das Verfassen meiner Diplomarbeit, aber auch mein Studium im Allgemeinen, war ein langwieriger und aufwendiger Prozess. Entsprechend viele Menschen haben mich während dieser Zeit inspiriert, begleitet, sowie moralisch, inhaltlich und organisatorisch unterstützt, und so zur Vollendung beigetragen. Wiewohl es mir nicht möglich ist, jede und jeden an dieser Stelle einzeln zu erwähnen, hoffe ich doch, dass sich alle Personen meiner Wertschätzung und Dankbarkeit bewusst sind.

Ich kann mich glücklich schätzen, durch so viele Menschen aus verschiedenen Kontexten vielfältige Unterstützung erfahren zu haben. Daher möchte ich zumindest einige Hintergründe näher spezifizieren.

Zunächst möchte ich mich in besonderer Weise bei meiner Familie, meinem Freundeskreis und meiner Schreibgruppe bedanken. Ohne die Inspiration zur Studienwahl, ihren Beistand und Halt in auch schwierigen Phasen, ihr Vertrauen in meine Fähigkeiten, das gemeinsame Lernen und Schreiben, die anregenden Gespräche, die Motivation, die kontinuierlichen Aufforderungen zum Weitermachen oder auch einfach die frohen und fröhlichen Stunden wäre ich gewiss nicht so weit gekommen.

Ebenso möchte ich mich bei der psychologischen Studienberatung und dem Schreibprogramm der Universität bedanken. Diese haben mir Wege und Maßnahmen gezeigt, neben meiner beruflichen Tätigkeit Strukturen zu etablieren, um den Schreib- und Lernprozess konsequent zu Ende zu führen.

Gleichzeitig möchte ich mich für die Wertschätzung und das Verständnis von Seiten meines Arbeitgebers bedanken. Die Möglichkeit zur effizienteren Zeitgestaltung haben mich motiviert und mir insbesondere in der Schlussphase Antrieb gegeben.

Ein außerordentlicher Dank gilt meinen Korrekturlesern, die sich trotz begrenzter zeitlicher Ressourcen die Mühe gemacht haben, sich durch die Diplomarbeit zu lesen, um mir wertvolles und umfangreiches Feedback zu geben. Ebenso möchte ich mich in diesem Zusammenhang bei meiner persönlichen Schreibunterstützerin bedanken, die mir geholfen hat, die Anmerkungen im richtigen Maß und an passender Stelle einzuarbeiten.

Schließlich möchte ich mich besonders bei meinem Betreuer für seine Geduld über den gesamten Schreibprozess hinweg, seine Zuversicht gegenüber dem Inhalt meiner Diplomarbeit, sowie seine Unterstützung, die Hilfestellungen und die Rückmeldungen bedanken.

Zusammenfassung

Wenngleich sich in der Software-Entwicklung die Erkenntnis durchgesetzt hat, dass Qualitätssicherung – und somit Software-Testen – essentiell ist, hat sich die Anwendung des Konzepts der Software-Zuverlässigkeit als ein relevantes Qualitätskriterium scheinbar noch nicht etabliert. Ziel dieser Arbeit ist es daher, einen Einblick in die Grundlagen und Konzepte der Software-Zuverlässigkeit zu vermitteln.

Dazu werden grundlegende Begriffe sowie Hintergründe aus der Software-Entwicklung und dem Software-Testen vorgestellt. Gleichzeitig wird Software-Zuverlässigkeit über die Wahrscheinlichkeitstheorie definiert. In weiterer Folge werden wichtige Wahrscheinlichkeitsverteilungen in diesem Zusammenhang, sowie das Maximum Likelihood Prinzip vorgestellt und angewendet.

Schließlich werden zwei NHPP-Modelle näher betrachtet und diskutiert. Dabei können Schwierigkeiten in Bezug auf ihren Gebrauch identifiziert werden. Einerseits stellen sich Modellannahmen in Bezug auf ihre Realisierung in der Praxis als problematisch dar, andererseits gestaltet sich die Anwendung von der Selektion eines geeigneten Modells über die Datenerfassung bis hin zur Schätzung der Zuverlässigkeit als aufwendig und komplex.

Abstract

Although it has become widely accepted in software development engineering that quality assurance – and thus software testing – is essential, the application of the concept of software reliability as a relevant quality criterion has apparently not yet been established. The aim of this work is therefore to provide an insight into the basics and concepts of software reliability.

For this purpose, basic concepts and background information from software development and software testing will be presented. At the same time, software reliability is defined by probability theory. Subsequently, important probability distributions in this context and the maximum likelihood principle are presented and applied.

Finally, two NHPP models are considered and discussed. Thereby difficulties with their use can be identified. On the one hand, model assumptions are problematic in terms of their implementation in practice, on the other hand, the application – from the selection of an appropriate model via data acquisition through to the estimation of reliability – proves to be extensive and complex.

Inhaltsverzeichnis

Einleitung

Im Zuge meiner beruflichen Tätigkeit als Beraterin im Bereich des Softwaretests bin ich von einem damaligen Kollegen auf das Thema Software-Zuverlässigkeit aufmerksam gemacht worden. Schnell stellte sich die Frage, was genau unter diesem Begriff zu verstehen ist und in welcher Relation das Thema zu meinem beruflichen Kontext steht. Genauso schnell stellte sich allerdings heraus, dass die Beantwortung dieser Frage keineswegs trivial ist und zumindest mir ohne Betrachtung der Grundlagen schwer fällt zu beantworten. Insbesondere auch deshalb, da es auf den ersten Blick umfangreiche Literatur zu geben schien, die verschiedenste Weiterentwicklungen von Modellen vorstellten, kaum jedoch Werke über die Grundkonzepte. Gleichzeitig hatte ich den Eindruck, dass die Anwendung der Modelle in Widerspruch zu aktuellen Konzepten der Software-Entwicklung und des Softwaretests stehen, sowie einen hohen Aufwand im Zuge der Datengewinnung verursachen.

Für den Versuch, diese Widersprüche zumindest für mich aufzulösen – und weil ich gerne bemüht bin Themen von der Basis her zu verstehen – war es mir ein Anliegen, zunächst die grundlegenden Konzepte von Software-Zuverlässigkeit zu betrachten. Gleichzeitig war mir klar, dass das Verfassen einer Diplomarbeit eine endliche Zeitspanne umfassen sollte. Daher kam ich zu der Erkenntnis, mich in diesem Zusammenhang auf eine einführende Betrachtung von Software-Zuverlässigkeit beschränken zu müssen.

Ziel dieser Arbeit ist es somit, einen Einblick in die Grundlagen und Konzepte der Software-Zuverlässigkeit zu vermitteln. Dazu sollen verschiedene Perspektiven auf das Thema eingenommen werden. Einerseits möchte ich den Kontext der Qualitätssicherung und Software-Entwicklung – insbesondere des Testens – betrachten, andererseits sollen die Konzepte aus der Stochastik erläutert werden, die für Software-Zuverlässigkeit von besonderem Belang sind. Schließlich möchte ich zwei relativ einfache Modelle und ihre Herleitung exemplarisch vorstellen, sowie Überlegungen zu einigen Aspekte in Zusammenhang mit der praktischen Anwendung anstellen.

Auf eine umfangreiche Analyse verschiedener Modelle – insbesondere eine Betrachtung weiterer Modellparameter – oder eine Evaluierung dieser bezüglich ihrer Vorhersagekraft anhand tatsächlicher Ausfalldaten musste in dieser Arbeit verzichtet werden. Ebenso konnte ich nicht alle Themen (in der Tiefe) behandeln, die für mich persönlich von größtem Interesse sind. Dazu zählen etwa die Fragen, inwieweit risikobasierte Teststrategien mit Software-Zuverlässigkeitsmodellen vereinbar sind, oder wie sich agile Software-Entwicklung auf die Modellierung auswirkt.

Zum Aufbau dieser Arbeit: Zunächst gehe ich in Kapitel 1 auf die grundlegendsten Begriffe und Überlegungen ein – Software-Zuverlässigkeit, Qualität und Fehler – um einen sehr allgemeinen Einblick zu ermöglichen.

Um das Thema besser in einen Anwendungskontext einordnen zu können, möchte ich in Kapitel 2 den Hintergrund der Software-Entwicklung und Qualitätssicherungsmaßnahmen darstellen. Da die Ausfalldaten insbesondere in der Testphase ermittelt werden, soll in Kapitel 3 dann diese Phase im Detail vorgestellt werden. Insbesondere möchte ich Testvorgehen und -strategien vorstellen, um hier einen Einblick in die Zusammenhänge und möglichen Konflikte in Bezug auf Zuverlässig zu vermitteln.

Anschließend wird in Kapitel 4 eine Einführung der wichtigsten mathematische Begriffe und Größen vorgenommen, sowie ein einfaches Modell vorgestellt.

Da in verschiedenen Texten Software-Zuverlässigkeit als Weiterentwicklung, aber auch in Diskrepanz zu Hardware-Zuverlässigkeit dargestellt wird und aus meiner Sicht nicht immer eindeutig ersichtlich wurde, welche Aspekte im jeweiligen Bereich gültig sind, war es mir ein Anliegen, dieses Spannungsfeld zu betrachten. Das soll in Kapitel 5 passieren, da aus meiner Sicht die Differenzen leichter verständlich sind, wenn zuvor die mathematischen Größen definiert wurden. In diesem Kapitel ergeben sich auch die in der (Software-)Zuverlässigkeit angewendeten Verteilungsfunktionen.

Im Anschluss findet ein Wechsel zu den mathematischen Konzepten statt. Das heißt, dass in Kapitel 6 die benötigten Verteilungsfunktionen betrachtet werden, sowie in Kapitel 7 ein kurzer Einblick in die in diesem Kontext interessanten Schätzmethoden gegeben wird. Das in diesem Zusammenhang wichtigste Prinzip – die Maximum Likelihood Schätzung – soll dann in Kapitel 8 auf die zuvor vorgestellten Verteilungsfunktionen angewendet werden.

Schließlich werden in Kapitel 9 zwei Software-Zuverlässigkeitsmodelle aus der Kategorie der nicht homogenen Poisson Prozess Modelle (NHPP) und ihre Herleitung gezeigt. Die Wahl fiel auf diese Kategorie an Modellen, da sie in der Praxis am weitesten verbreitet oder zumindest in der Literatur am meisten diskutiert erscheint. Innerhalb der NHPP ist das Goel-Okumoto NHPP ein häufig genanntes und relativ einfaches Modell, weshalb ich zunächst auf dieses eingehe. Gleichzeitig wird in vielen Texten die Annahme, dass im Software-Entwicklungsprozess Fehler immer erfolgreich korrigiert werden, in Frage gestellt. Daher fiel die Wahl für ein zweites Modell auf ein imperfect debugging Modell. Wenngleich es viele weitere Eigenschaften, Annahmen, Modellerweiterungen und Vorgehensweisen gibt, die für eine nähere Betrachtung lohnenswert sind, musste ich im Zuge dieser Arbeit darauf verzichten und versuchte stattdessen, einige Ideen über weitere Möglichkeiten vorzustellen.

Schließlich möchte ich an dieser Stelle darauf hinweisen, dass eine große Schwierigkeit im Verlauf der Arbeit war, die verschiedenen Begrifflichkeiten, Notationen und (Modell-)Kategorien in der Literatur in Zusammenhang zueinander bringen. Daher wurde an mancher Stelle versucht, eine entsprechende Übersicht vorzustellen.

1 Grundbegriffe der Software-Zuverlässigkeit

1.1 Was ist Software Reliability

1.1.1 Problemstellung

Es ist unbestritten, dass Software-Programme heutzutage unseren Alltag bestimmen. Dabei ist die Erwartung der Nutzer/innen, dass die Programme „richtig“ funktionieren: Eben so, wie es vorgesehen ist und zumeist ohne konkrete Vorstellungen davon, wie dieses richtige Verhalten gewährleistet wird. Gleichzeitig ist es normalerweise ein aufwändiger und teurer Prozess, „gute“ und zuverlässige Software zu entwickeln und zur Verfügung zu stellen.

Doch was sind die möglichen Konsequenzen, wenn die Qualität nicht ausreichend ist?

Abgesehen davon als Anwender/in unzufrieden mit schlechten Softwareprogrammen zu sein, können die Auswirkungen von Fehlern – je nach System und Systemeinsatz – unterschiedlich gravierende Folgen nach sich ziehen. So können Fehler in manchen Anwendungsgebieten zur Verletzung oder Tötung von Menschen oder zu Sachschäden führen (zum Beispiel in der Autoindustrie). [23] In den meisten Fällen jedoch sind Fehler mit erheblichen Kosten verbunden. Angefangen von Schadens- oder Strafzahlungen, wenn etwa gesetzliche Bestimmungen oder Regulatorien nicht eingehalten wurden (beispielsweise in der Bankenbranche), über den Ausfall von Gewinnen (etwa durch Imageverlust beziehungsweise den Verlust von Kunden), bis hin zu den Reparaturkosten, um die Software zu korrigieren.

1.1.2 Software ohne Fehler?

Man könnte daraus schlussfolgern, dass – um all diese unangenehmen Konsequenzen auszuschließen – nur Software produziert werden sollte, die vollständig fehlerfrei ist.

Um eine Vorstellung von der Größenordnung zu erhalten sei darauf hingewiesen, dass nach einer Studie von [21] zunächst von 19,7 Fehlern pro Tausend Zeilen Code auszugehen ist. Nach der ersten Prüfung einzelner Einheiten durch die Programmierer/innen selbst (nach dem Unittest – mehr dazu in Kapitel 3.2) sind dann laut [23] noch acht Fehler pro Tausend Zeilen Code beziehungsweise im Fall von erfahrenen Programmiererinnen und Programmierern noch sechs Fehler pro Tausend Zeilen Code vorhanden.

Daraus ist ersichtlich, dass bereits während des Programmierens selbst versucht wird, laufend Fehler zu reduzieren, dies aber scheinbar nicht vollständig möglich ist. Tatsächlich gibt es ein Set an verschiedenen Methoden, um Qualität in der Software-Entwicklung zu gewährleisten. Auf diese wird in Kapitel 2.2.3 eingegangen. Um jedoch zu garantieren, dass die Qualitätssicherungsmaßnahmen auch wirkungsvoll angewendet werden, ist es notwendig, die Software als Ganzes zu prüfen, also zu testen. So können einerseits doch noch vorhandene Fehler identifiziert werden, andererseits kann „bewiesen“ werden, dass das Programm tatsächlich einwandfrei funktioniert.

Dennoch – oder gerade deshalb – ist es nach [23] nur theoretisch möglich, vollständig fehlerfrei zu produzieren. Einerseits ist neue Software heutzutage bereits zu umfangreich, um vollständig getestet werden zu können, andererseits kann sie nicht nach jeder Anpassung (BugFixing oder neue Features) erneut in voller Detailtiefe geprüft werden.

So schreibt [23]

„[...] In theory, software can be error-free, and unlike hardware, does not degrade or wear out but it does deteriorate. The deterioration here, however, is not a function of time. Rather, it is a function of the results of changes made to the software during maintenance, through correcting latent defects, modifying the code to changing requirements and specifications, environments and applications, or improving software performance. All design faults are present from the time the software is installed in the computer. In principle, these faults could be removed completely, but in reality, the goal of perfect software remains evasive [...].“ [23]

Eine weitere Überlegung ist, dass Testen sehr zeit- und kostenintensiv ist. In [23] wird auf eine Studie von Microsoft verwiesen, die ergab, dass zur Identifizierung und Korrektur eines Fehlers etwa 12 Programmierstunden notwendig sind.

Insofern stellt sich die Frage, ob beziehungsweise wann die Kosten des Testaufwands die Kosten im Falle von Fehlern während des Betriebs überwiegen.

Dazu muss auch der Schweregrad des Verlustes quanitfiziert werden. Man spricht dabei von Risikokosten oder „risk cost of failure“. [23]

Interessant festzuhalten ist, dass nach [21] die in der Studie untersuchten Systeme mit circa 1,5 Fehlern pro Tausend Zeilen Code in Betrieb genommen wurden.

1.1.3 Zielsetzung von Software Reliability

Es wurde also festgestellt, dass es nicht möglich und nicht unbedingt von Interesse ist, so lange zu testen, bis sichergestellt wurde, dass alle Fehler gefunden wurden, die Qualität entsprechend hoch und das Risiko von Fehlern und Folgeschäden minimiert oder gar Null ist. Das heißt, es muss entschieden werden, wann die Software „gut genug“ ist im Vergleich zu Risiko und Kosten. Es muss also auch abgeschätzt werden können, wie viele Fehler noch in der Software vermutet werden können.

Ein Ziel der Software Reliability ist es, die verbleibende Anzahl an Fehlern im System durch geeignete Modelle und Analyse-Methoden vorherzusagen. Somit liegt ein Schwerpunkt auch darin, den optimalen Zeitpunkt zu ermitteln, an dem die Tests beendet werden und die Software somit in Produktion gehen kann. Dabei sollen Daten zur Verfügung gestellt werden, um zwischen Testzeit, Zuverlässigkeit, Kosten und Performance abwiegen zu können. [9]

Es geht also darum, in der Software-Entwicklung den optimalen Zeitpunkt zu identifizieren, um das Produkt auf den Markt zu bringen – selbst wenn noch nicht alle Fehler identifiziert oder behoben wurden.

Software Reliability – beziehungsweise Software-Zuverlässigkeit – soll also dabei unterstützen, Risiken, Kosten und Marktchancen abzuschätzen. [12] Ziele sind

- geringe Kosten
- ein geringes Risiko
- hohe Qualität.

Dazu wird auf die Wahrscheinlichkeitstheorie zurück gegriffen. Eine gängige Definition von Software-Zuverlässigkeit ist:

„Software reliability is the probability that the software will not fail for a specified period of time under specified conditions. The greatest problem facing the software industry today is how to quantitatively assess software reliability characteristics.“ [23]

Oder auch: „Zuverlässigkeit ist die Wahrscheinlichkeit dafür, dass das betrachtete Objekt in einem bestimmten Zeitraum unter bestimmten Einsatzbedingungen einwandfrei funktioniert. Die Zuverlässigkeit ist somit auf ein festes Zeitintervall bezogen und bedeutet die Wahrscheinlichkeit dafür, dass in diesem Zeitintervall kein Ausfall stattfindet [...]“ [12]

Das heißt, wenn ein System beispielsweise eine Zuverlässigkeit von 95% für den Zeitraum von 8 Stunden besitzt, so wird ein durchschnittlicher Anwender in 95 von 100 Fällen das System 8 Stunden lang ohne Ausfälle nutzen können. [23]

1.2 Qualitätsbegriff

Nach der Norm DIN ISO 9000 [2] ist Qualität der „Grad, in dem ein Satz inhärenter Merkmale eines Objekts Anforderungen erfüllt“.

Obwohl diese Definition sehr allgemein und relativierend gehalten ist, sind nach [28] bereits drei grundlegende Schlussfolgerungen daraus abzuleiten:

So wird nach [28] durch den Begriff „Grad“ bereits festgehalten, dass Qualität über die Zuordnung von Messwerten auf eine vordefinierte (Bewertungs-)Skala erfolgt und so ein gewisses Qualitätsniveau erreicht werden kann.

Mit „inhärenten Merkmalen“ werden in der Software enthaltene Eigenschaften bezeichnet, die eine Bewertung über die festgelegten Skalen erst ermöglichen. Somit wird Qualität messbar.[28]

Über „Anforderungen“ wird definiert, welche Merkmale zu welchem Grad erfüllt werden müssen, um dem Verwendungszweck der Software beziehungsweise den Erwartungen des Auftraggebers gerecht zu werden. Dadurch wird auch impliziert, dass Qualität keine objektive, immer-währende Größe darstellt, sondern an einen Zweck und Betrachter gebunden, und somit über die Zeit änderbar sein kann. [28]

Auch [4] betont, dass die Definition nahelegt, dass zur Bestimmung von Qualität die Anforderungen an ein Produkt im Vorhinein festgelegt werden müssen. Gleichzeitig stellt [4] fest, dass sich ebendiese Anforderungen auf Nutzer beziehen, und somit Qualität keinen absoluten Begriff darstellt. Vielmehr hängt sie von der Verwendung der Software ab und somit davon, welche Eigenschaften in welchem Ausmaß gefordert sind.

Beispielsweise ist ein Programm nicht unbedingt von höherer Qualität, wenn es Berechnungen exakter ausführt und mehr Dezimalstellen liefert, solange diese nicht „gefordert“ oder benötigt sind [4].

Tabelle 1 nach [28] gibt typische Qualitätsmerkmale in Bezug auf Software wieder.

Qualitätsmerkmal	Bedeutung
Funktionalität	Vorhandensein von Funktionen mit festgelegten Eigenschaften. Diese Funktionen erfüllen die spezifizierten Anforderungen.
Zuverlässigkeit	Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen definierten Zeitraum zu bewahren.
Benutzbarkeit	Aufwand, der zur Benutzung erforderlich ist und individuelle Beurteilung einer Benutzung durch eine festgelegte Benutzergruppe.
Effizienz	Verhältnis zwischen Leistungsniveau der Software und Umfang der eingesetzten Betriebsmittel unter definierten Bedingungen.
Änderbarkeit	Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist (z. B. Funktionserweiterungen, Korrekturen, Anpassungen).
Übertragbarkeit	Eignung der Software, von einer Systemumgebung in eine andere übertragen zu werden.

Tabelle 1: Qualitätsmerkmale von Software, übernommen aus [28]

In Bezug auf Tabelle 1 kann Zuverlässigkeit als ein Merkmal von Qualität betrachtet werden. Dem entgegen gibt es die Auffassung, Zuverlässigkeit als ein dem Qualitätsbegriff umfassenderes Konzept zu verstehen. Hier liegt der Schwerpunkt der Betrachtung darin, *Zuverlässigkeit als Qualität während einer vorgegebenen Zeitspanne* als Zeitfunktion von Qualität und somit weitreichender als diese selbst zu verstehen. [4, 12]

Umgekehrt unterstützt die Betrachtungsweise, Zuverlässigkeit über die Wahrscheinlichkeitstheorie darzustellen, die Verstehensweise, Zuverlässigkeit als ein Qualitätsmerkmal zu betrachten. In dieser Verstehensweise wird das Qualitätsmerkmal durch die Wahrscheinlichkeitstheorie quantifizierbar gemacht. [4]

In dieser Arbeit soll dieser Zugang angenommen werden. Das heißt, Zuverlässigkeit wird als ein Qualitätsmerkmal aufgefasst, das über die Wahrscheinlichkeitstheorie quantifizierbar gemacht wird.

1.3 Was ist ein Fehler?

Wenn sich Qualität aus der Erfüllung der gestellten Forderungen ergibt, so kann umgekehrt ein Fehler als die Nichterfüllung einer Forderung beziehungsweise als unzulässige Abweichung von festgelegten Merkmalen verstanden werden. [4]

Für die Betrachtungen in dieser Arbeit ist es allerdings notwendig, zwischen den Begriffen Fehler und Ausfall zu unterscheiden. Nachfolgend soll Fehler mit „fault“ oder „defect“ übersetzt werden, sowie Ausfall mit „failure“. In [23] werden die Begriffe folgendermaßen beschrieben:

„Software failure means the inability to perform an intended task specified by a requirement. A software fault (or bug) is an error in the program source-text, which causes software failure when the program is executed under certain conditions. Hence, a software fault is generated when a mistake is made.“ [23]

Greift man auf den IEEE Standardglossar für Software Engineering [1] zurück, so wird (etwa in Zusammenhang mit der Definition von „failure“) ebenso differenziert zwischen „[...] a human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error).“ [1]

Festzuhalten ist, dass der Begriff „error“ in der untersuchten Literatur tendenziell als Fehlerhandlung eines Menschen verstanden wird, etwa in [23]. Daher soll diese Verstehensweise auch in dieser Arbeit übernommen werden.

Zusammenfassend kann das Wortfeld folgendermaßen beschrieben werden:

- Error = Mistake = Irrtum = Fehlerursache = Fehlerhandlung
  Darunter wird der Irrtum verstanden, der in den Überlegungen der handelnden Person (Software Engineer / Designerin) vorhanden ist.

- Defect = Fault = Fehlerzustand
  Darunter ist der fehlerhafte Code selbst zu verstehen, der das Resultat des menschlichen Irrtums ist. Der Fehlerzustand kann während der Ausführung des Programms einen Ausfall hervorrufen.

- Failure = Fehlerfolge = Fehlerwirkung = Ausfall = Versagen
  Darunter wird das tatsächliche Eintreten einer Abweichung von dem erwarteten beziehungsweise geforderten Ergebnis verstanden. Das heißt, dass eine spezifizierte Anforderung an das System oder die Komponente nicht erfüllt wird.
  Damit stellt ein Ausfall ein beobachtbares Ereignis zu einem bestimmbaren Zeitpunkt dar, das gezählt werden kann.

Im Bereich der Zuverlässigkeit ist insbesondere der dritte Begriffsbereich von Relevanz. Es werden Ausfälle gezählt beziehungsweise deren Zeitpunkte festgestellt und analysiert. Um das tun zu können, ist es notwendig vor der Datenerhebung festzulegen, was als Ausfall zählt, da das von der Anwendung beziehungsweise den definierten Anforderungen abhängen kann (Vgl. auch [23]). Dazu kann es auch notwendig sein, Grenzwerte festzulegen, falls das Objekt mit der Zeit leistungsschwächer wird (zum Beispiel in Bezug auf die Performance). Normalerweise wird auch eine Skala definiert, mit der die Kritikalität festgehalten wird, um so einschätzen zu können, welche Auswirkungen und Risiken von dem Ausfall zu erwarten sind.

Im Bereich der Hardware unterscheidet man zwischen den Begriffen „Ausfall“ und „Versagen“. Für den Bereich der Software-Zuverlässigkeit ist diese Unterscheidung nicht relevant und die Begriffe werden daher normalerweise synonym verwendet.[4]

Festzuhalten ist, dass in der Literatur eine gewisse Unschärfe bei der Verwendung der Begriffe festgestellt wurde beziehungsweise in gewissen Kontexten eine Unterscheidung nicht notwendig

scheint. Es kann daher vorkommen, dass der Begriff Fehler auch in der Bedeutung eines Ausfalls verwendet wird, wiewohl versucht wurde, nach Möglichkeit eine klare Trennung vorzunehmen.

2 Software Engineering: Hintergrund

In diesem Kapitel sollen Software-Zuverlässigkeit und Software-Zuverlässigkeitsmodelle aus dem Blickwinkel des Software-Entwicklungsprozesses betrachtet werden. Dazu soll ein kurzer Einblick über den Lebenszyklus von Software, sowie über Qualitätssicherungsmaßnahmen und ihren Zusammenhang zur Zuverlässigkeit gegeben werden.

2.1 Software Lifecycle

Im Software-Entwicklungsprozess können folgende Phasen identifiziert werden:

- Analyse-Phase
- Design-Phase
- Entwicklungsphase
- Test-Phase
- Betriebsphase (Markteinführung) [23]

Demgegenüber umfasst der Software-Lebenszyklus die Zeitspanne von der Problemstellung und dem Beginn der Konzeption, über die Entwicklung und Verwendung, bis zu dem Zeitpunkt, zu dem die Software nicht mehr zur Verfügung steht. [1]

In den folgenden Unterkapiteln werden die einzelnen Phasen näher erläutert.

2.1.1 Analyse Phase

Ziel dieser Phase ist es, die Problemstellung zu analysieren. Das heißt, es erfolgt eine Problembeschreibung, eine Abgrenzung bezüglich des Umfangs des Projekts, sowie eine Klärung der Notwendigkeit und des Nutzens der Software. [23]

Darauf aufbauend werden die Anforderungen an die Software und die Spezifikationen definiert. [23]

Dabei werden in der Anforderungsdefinition Leistungsumfang und Einschränkungen beschrieben. Sie enthält alle wichtigen Informationen, um das Projekt erfolgreich durchführen zu können, also Informationen zur gewünschten Funktionalität, der Usability, dem Verwendungszweck, dem Zeitplan, Ressourcen, einer Machbarkeitsstudie, Kosten- und Gewinnschätzungen, einer Risikoanalyse und dem Projektplan. [23]

Die Spezifikation dient dann als Präzisierung der Anforderungen für die Entwickler. In ihr werden alle relevanter Aspekte, Schnittstellen und Grenzen festgehalten. Sie ist auch jenes Dokument, das als Ausgangsbasis für die spätere Validierung dient. [23]

2.1.2 Design Phase

In der Designphase wird das Konzept der Software entwickelt. Das heißt, es wird die Systemarchitektur entworfen, sowie auf die Detaillierung (etwa die Wahl der Programmiersprache) eingegangen. [23]

Gleichzeitig wird mit der Testplanung begonnen. In dieser Phase wird das Testobjekt identifiziert (also die Frage geklärt, was getestet werden muss), die Teststrategie und Methodik festgelegt, die Testfallspezifikation erstellt, sowie Zeitpläne und Ressourcen ermittelt. [23]

2.1.3 Entwicklungsphase

In der Entwicklungsphase findet das Schreiben des Codes in einer Programmiersprache statt. Dazu werden wiederverwendbare Module identifiziert, Code erstellt und Reviews beziehungsweise Code-Prüfungen durchgeführt. [23]

Gleichzeitig wird der Testplan finalisiert. [23]

2.1.4 Test Phase

In der Testphase geht es primär darum sicherzustellen, dass die Software der geforderten Qualität entspricht. Das heißt, es sollen einerseits Fehler gefunden und korrigiert werden, andererseits soll gezeigt werden, dass das System korrekt funktioniert und die gewünschten Qualitätskriterien erfüllt sind (siehe auch Tabelle 1 aus Kapitel 1.2).

Nach [23] werden in dieser Phase des Lifecycle drei Ziele verfolgt:

1. Die Qualität des Programms soll durch das Finden und Eliminieren von Fehlern bestätigt werden
2. Die Umsetzung aller spezifizierten beziehungsweise geforderten Anforderung soll demonstriert werden
3. Die Zuverlässigkeit soll für die Betriebsphase geschätzt werden.

Erst wenn die Testphase erfolgreich durchlaufen wurde, erfolgt die Abnahme des Produkts und das System kann in die Betriebsphase wechseln.

2.1.5 Betriebsphase

Die Betriebsphase beinhaltet Installation, Training, Support und Wartung. [23]

Auch in dieser Phase können im Zuge von Wartungsarbeiten noch Änderungen am Code (und somit am Produkt selbst) vorgenommen werden. Ziel dieser Wartungsarbeiten kann es sein, Fehler zu korrigieren, die Software an neue Anforderungen anzupassen oder sie in bestehenden Funktionen zu verbessern. Dabei wird der Software-Lifecycle in ähnlicher Weise wie bei einer Neuentwicklung erneut durchlaufen. [23]

2.1.6 Software Entwicklungsmodelle

In klassischen Vorgehensmodellen werden die eben besprochenen Phasen als aufeinander folgend verstanden, so auch bei [23]. In modernen Entwicklungsprojekten werden die einzelnen Phasen allerdings in kurzen Zyklen iterativ durchlaufen, bis ein fertiges „Gesamtprodukt“ entsteht.

Gleichzeitig soll an dieser Stelle darauf hingewiesen werden, dass es sich hier um theoretische Vorgehens-Modelle handelt. So weist etwa [6] darauf hin, dass die einzelnen Phasen in der Praxis oft nicht strikt voneinander getrennt sind. Insbesondere kann dadurch nicht unbedingt davon ausgegangen werden, dass in der Testphase keine Veränderungen am oder Ergänzungen von Code vorgenommen werden.

2.2 Software Qualitätssicherungsmaßnahmen

Entsprechend dem Software-Lebenszyklus können Fehler in unterschiedlichen Phasen entstehen und somit verschiedene Ursachen haben. Beispielsweise können Anforderungen falsch interpretiert werden. Ebenso kann die Architektur der Software falsch konzipiert sein, etwa durch Fehler im Prozessfluss oder falsche Schnittstellendefinitionen. In diesem Fall können einzelne Komponenten nicht oder nicht ausreichend miteinander interagieren. Schließlich können Fehler auch bei der Umsetzung der Anforderungen in ein Stück Code entstehen. Je nach Fehlerursache sind unterschiedliche Findungs- und Korrekturmaßnahmen geeignet. So geht es etwa in der Testphase einerseits darum, die Software zu verifizieren, andererseits darum, sie zu validieren (näheres dazu in Kapitel 2.2.2). Doch nicht nur in der Testphase selbst, auch in den anderen Phasen der Softwareentwicklung finden sich Möglichkeiten und Strategien, Qualität sicherzustellen.

Dazu soll zunächst ein Blick darauf geworfen werden, inwieweit Fehler in den einzelnen Phasen entstehen und gefunden werden, bevor verschiedene Herangehensweisen zur Qualitätssicherung zusammengefasst werden.

2.2.1 Verteilung von Fehlern im Software-Lifecycle

Um zu verstehen, wie sich Qualitätssicherung in den Software-Entwicklungsprozess eingliedert, soll ein kurzer Blick darauf geworfen werden, wann typischerweise Fehler entstehen und wann sie gefunden werden.

Dazu soll Tabelle 2 nach [23] betrachtet werden. Sie stellt die Verteilung der Fehler-Einführung sowie der Fehler-Findung bezogen auf die verschiedene Phasen in der Software-Entwicklung dar.

Phase	Fehler-Einführung (%)	Fehler-Findung (%)
Analyse	55	18
Design	30	10
Entwicklung und Test	10	50
Betrieb	5	22

Tabelle 2: Verteilung der Fehler-Einführung und Fehler-Findung im Software-Entwicklungs-Prozess [23]

Es ist deutlich zu sehen, dass der Anteil an neu eingeführten Fehlern initial sehr hoch ist und im Verlauf der Softwareentwicklung abnimmt, während die Mehrzahl der Fehler erst in späteren Phasen entdeckt wird.

Gleichzeitig konnte nach [23] festgestellt werden, dass die Kosten einer Fehlerkorrektur steigen, je weiter der Entwicklungsprozess voran schreitet. Mehr noch, befindet sich das Produkt bereits in der Betriebsphase, sind die Kosten für gewöhnlich deutlich höher als während der Herstellung. Ebenso nimmt die Wahrscheinlichkeit mit dem Fertigstellungsgrad zu, dass durch die Korrektur neue Fehler eingeführt werden. [23]

Abbildung 1 nach [23] veranschaulicht diese zwei Beziehungen anhand verschiedener Teststufen. Dabei sind Design Reviews der Design-Phase zuzuordnen, Unittests der Entwicklungsphase, sowie die weiteren Tests aufeinanderfolgend in der Test-Phase (Näheres dazu in Kapitel 3.2).

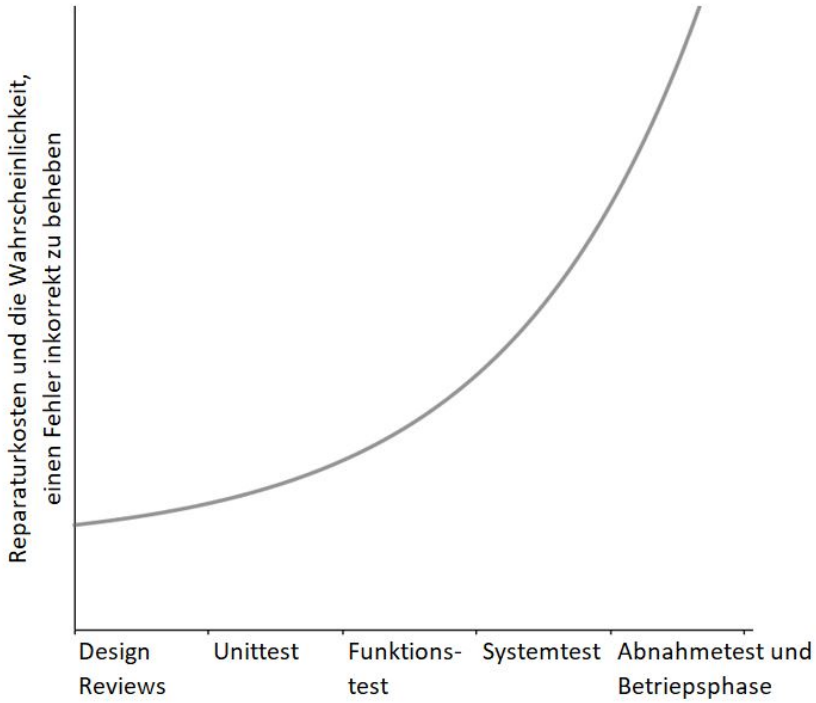

Abbildung 1: Kosten für Fehlerkorrekturen, beziehungsweise die Wahrscheinlichkeit fehlerhafter Korrekturen in Relation zum Voranschreiten im Software-Entwicklungsprozess nach [23]

Der dargestellte Zusammenhang lässt sich folgendermaßen verbalisieren:

1. Die Kosten für die Korrektur eines Fehlers steigen in den letzten Phasen des Entwicklungsprozesses schnell an.

2. Die Wahrscheinlichkeit, einen bekannten Fehler fehlerhaft zu korrigieren, nimmt kurz vor Fertigstellung des Systems rapide zu. Das heißt, es kann durch die Korrektur eines Fehlers gegen Ende des Projekts mehr Schaden als Nutzen verursacht werden.

Zu einem ähnlichen Ergebnis bezüglich der Kosten kommt auch [17].

Das Interesse ist also groß, Fehler möglichst frühzeitig zu finden und zu beheben. Ebenso

stellt sich die Frage, ob jeder Fehler korrigiert werden sollte, der gefunden wird, oder ob es gute Gründe gibt, dies zu unterlassen.

2.2.2 Verifikation versus Validierung

Eine relevante Unterscheidung ergibt sich bezüglich der Begriffe Verifikation und Validierung. Nach dem IEEE Standardglossar für Software Engineering [1] sind diese beiden Begriffe folgendermaßen definiert:

- Software verification is „the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."
- Software validation is „the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements."

Spricht man von Verifikation, meint man also das Prüfen, ob die Entwicklung den definierten Anforderungen entspricht, während Validierung zum Ziel hat festzustellen, ob die Funktionen des Produkts den Anforderungen des Kunden entsprechen. [23]

Entsprechend findet sich Verifikation nicht nur in der Testphase selbst, sondern in allen Phasen des Software-Entwicklungsprozesses. Sie kann auch dazu dienen, sicherzustellen, dass die Qualität einem notwendigen Mindestmaß entspricht, um für die nächste Phase zugelassen zu werden („Quality Gate"). Dazu wird geprüft, ob das Produkt in jeder Phase korrekt, vollständig und konsistent mit sich selbst und mit seinem Vorgängerprodukt ist. [23]

Validierung dagegen findet zumeist gegen Ende des Entwicklungsprozesses statt und stellt sicher, ob das Endprodukt mit dem übereinstimmt, was gefordert wurde und entsprechend die Bedürfnisse des Kunden erfüllt. [23]

Die Unterscheidung der beiden Begriffe finden sich bei [23] auch durch die Formulierung der Fragestellungen, ob *das Produkt richtig gemacht wird* (Verifikation) oder *das richtige Produkt gemacht wird* (Validierung).

In Bezug auf Software-Zuverlässigkeit steht typischerweise Verifikation im Vordergrund.

2.2.3 Überblick über verschiedene Herangehensweisen

Wie bereits kurz erläutert können Fehler zu unterschiedlichen Zeitpunkten im Entwicklunsprozess – mit dementsprechend verschiedenartigen Ursachen – entstehen. Gleichzeitig wurde in Kapitel 2.2.1 festgehalten, dass die Kosten für Korrekturen steigen, je später im Verlauf sie vorgenommen werden. Es liegt somit auf der Hand mit Qualitätssicherungsmaßnahmen so nah wie möglich an der Fehlerentstehung anzusetzen. Im Folgenden soll eine kurze Übersicht über verschiedene Kategorien von Maßnahmen und Methoden gegeben werden, um besser zu verstehen, wie Software-Zuverlässigkeit in diese einzuordnen ist.

Auch wenn in der Literatur unterschiedliche Begrifflichkeiten verwendet werden, soll hier versucht werden eine erste Einteilung vorzunehmen. [6, 17, 8, 4, 27]

- Formale Methoden und deterministische Nachweisverfahren

  Dazu gehören beispielsweise formale Spezifikationen, die Berechnungen beziehungsweise Korrektheitsbeweise zulassen. Gleichzeitig kann systematisches Testen (also White-Box, Black-Box oder Grey-Box Tests) zu den deterministischen Nachweisverfahren gezählt werden.

- Informale Methoden beziehungsweise informelle Qualitätssicherungsverfahren, White-Box Modelle, Mikro-Modelle, statische Modelle

  Statische Nachweisverfahren und Modelle beziehen sich auf frühe Stadien in der Software-Entwicklung und die entsprechenden Metriken und Messungen. Dazu zählen beispielsweise Anforderungs-Reviews oder Durchsichten. Sie beinhalten keine zufälligen Ereignisse.

- Statistische Methoden und Modelle beziehungsweise probabilistische Nachweisverfahren, Black-Box Modelle, Makro-Modelle, dynamische Modelle

  Dynamische Modelle sind in späteren Phasen der Software-Entwicklung angesiedelt (also in der Test- und Betriebsphase). Dabei werden das Auftreten von Ausfällen sowie die Fehlerbehebung als wahrscheinlichkeitstheoretische Ereignisse betrachtet. Darauf basierend kann dann Software-Zuverlässigkeit ermittelt werden. Da sich diese während des Software-Tests aufgrund von Korrekturen für gewöhnlich verbessert, werden diese Modelle auch gerne Software Reliability Growth Models (SRGM) genannt. In dieser Arebeit soll es hauptsächlich um diese Modelle gehen.

Eine andere beziehungsweise detailliertere Untergliederung nehmen die Autor/innen von [9] in ihrer Untersuchung vor. Sie haben Publikationen zu diesem Thema aus dem Zeitraum 2003 bis Anfang 2014 untersucht, um eine Systematisierung der zahlreichen Modelle vorzunehmen. Die Autor/innen konnten die Modelle im Kontext von Software Reliability in folgende Kategorien einteilen:

- Software Reliability Growth Models (SRGM)
- Bayesian Methods (BYM).
- Test-Based methods (TBM).
- Artificial Intelligence based techniques (AI based).
- Static and Architectural Reliability models (SARE).
- Other methods. [9]

Weiters haben sie festgestellt, dass die am meisten untersuchten und verwendeten Modelle Software Reliability Growth Models sind (wenngleich ein Rückgang an wissenschaftlichen Texten dazu festgestellt wurde). Von diesen analysieren die meisten Modelle den Fehlerfindungsprozess als eine Funktion bezogen auf die Zeit. Das heißt, im Regelfall werden dabei Messungen bezüglich der Fehlerfindungsrate pro Zeiteinheit beziehungsweise den Zeiten zwischen dem Auftreten von

Fehlern angestellt und für die Schätzung der Gesamtanzahl an Fehlern im Programm oder der Ausfallwahrscheinlichkeit des Systems verwendet. [9]

3 Testen als Basis für Software-Zuverlässigkeit (die Testphase im Detail)

Um Software-Zuverlässigkeit ermitteln beziehungsweise schätzen zu können, sind Daten über die Fehlerhäufigkeit zur Analyse notwendig. Diese können in der Testphase gewonnen werden, da hier einerseits Ausfalldaten gesammelt werden können, andererseits bereits ein relativ fertiges Produkt vorliegt, das eine sinnvolle Schätzung ermöglicht.

Um die Dialektik zwischen Testen im Sinne der Produktverbesserung und Testen zum Erheben von Daten besser zu verstehen, soll an dieser Stelle etwas näher auf diese Phase und ihre Hintergründe eingegangen werden.

3.1 Was ist Testen?

Wie wir in Kapitel 2.1.4 erfahren haben, ist ein Ziel des Testens, durch das Ausführen der Software Fehler zu finden. Das heißt – stark vereinfacht gesprochen – dass in der Software verschiedene Eingaben getätigt werden, um im Anschluss zu prüfen, ob sich das System den Eingaben entsprechend verhält beziehungsweise die korrekten Ausgabewerte produziert.

Anders formuliert ist nach [23] Software als ein „Instrument“ zu betrachten, das ein diskretes Set an Input-Werten in ein diskretes Set an Output-Werten transformiert. So enthält Software beispielsweise Anweisungen, die Ausdrücke auswerten und entscheiden, welche Operationen durchgeführt werden sollen. Die dabei entstehenden Ergebnisse können temporär oder permanent gespeichert werden.

Demzufolge kann nach [23] Software als eine Funktion f betrachtet werden, bei der jedem Element aus dem Eingabebereich ein Element aus dem Ausgabebereich zugeordnet wird.

$$f : \text{Eingabebereich} \rightarrow \text{Ausgabebereich}$$

Im Zusammenhang mit Software kann der Eingabebereich als Menge aller Input-Zustände und der Ausgabebereich als Menge aller Output-Zustände verstanden werden.

Im Zuge des Tests spricht man beim Ausführen der Software von Testfällen oder Testszenarien. Beim Funktionstest soll die Korrektheit der Applikation geprüft werden. Es soll also über die Eingabe von Input-Zuständen verifiziert werden, ob die gewünschten Output-Zustände eintreten.

Um nun das Ziel zu erfüllen Fehler zu finden, müssen folglich unterschiedliche Input-Zustände verwendet werden. Mehr noch, nach [23] sollte daher ein guter Testfall eine möglichst hohe Wahrscheinlichkeit besitzen, um einen unentdeckten Fehler zu finden.

3.2 Ausflug zu Testvorgehen, Strategien und Methodik

Es hat sich gezeigt, dass, wenn Testen – insbesondere der Funktionen – ausschließlich intuitiv und ohne vordefinierten Plan stattfindet, gewisse Bereiche der Software intensiver beziehungsweise mehrfach getestet werden, während andere Bereiche oder Input-Zustände ausgelassen oder vergessen werden. Daher haben sich verschiedene Strategien und Vorgehensweisen etabliert. Im Folgenden sollen die wichtigsten Konzepte kurz vorgestellt werden.

Grundsätzlich gilt, dass bei mehrfacher Testung desselben Input-Zustands ohne Änderung der Software der Output-Zustand unverändert gleich sein wird und nicht mehr oder weniger Fehler gefunden werden können, als es bei der ersten Durchführung der Fall ist. Es werden also nicht mehr Informationen generiert, wenn der gleiche Testfall mehrfach durchgeführt wird, was bei der Testung des gleichen Eingabewertes beziehungsweise Kombination aus Eingabewerten der Fall ist.

Daher liegt es nahe, den Eingabebereich zu analysieren (etwa durch Äquivalenzklassenanalyse und Kombinatorik), um verschiedene Input-Zustände zu identifizieren. Denn auch wenn es wie in Kapitel 1.1.2 nur theoretisch möglich ist vollständig zu testen, so ist das Bestreben dennoch, dies näherungsweise zu tun.

Wie in Kapitel 2.2.1 festgehalten wurde, sind die Kosten einer Fehlerbehebung umso höher, je später diese stattfindet. Daher ist es sinnvoll, möglichst viele beziehungsweise besonders kritische Fehler möglichst frühzeitig zu finden und zu korrigieren. Es liegt also nahe, die identifizierten Input-Zustände beziehungsweise Testfälle zur priorisieren und der entstehenden Reihenfolge entsprechend durchzuführen. Eine Priorisierung kann beispielsweise anhand des Risikos, also der Häufigkeit der Benutzung und des potentiellen Schadens, vorgenommen werden.

Gleichzeitig ist eine Strategie, das System stufenweise zu testen indem es in kleinere Einheiten zerlegt wird beziehungsweise um dem Rechnung tragen zu können, dass abgeschlossene Einheiten getestet werden können, sobald diese fertig gestellt wurden (das gilt insbesondere auch, wenn noch nicht das gesamte System zur Verfügung steht). Dabei wird im Allgemeinen zwischen Unittest, Komponententest, (System-)Integrationstest und End-to-End beziehungsweise Abnahemtest unterschieden. Während im Unittest und Komponententest kleine Einheiten unabhängig voneinander und mit dem Ziel eines hohen Detaillierungsgrades getestet werden, steht in den folgenden Teststufen das Zusammenspiel dieser sowie deren korrekte Funktionalität im Vordergrund. Im Abnahmetest schließlich steht die Validierung des Systems im Vordergrund. Durch dieses Vorgehen sollen einerseits Fehler, die bereits in kleinen (unabhängigen) Einheiten vorhanden und identifizierbar sind, frühzeitig erkannt werden, andererseits Ressourcen gespart werden, da insbesondere End-to-End-Szenarien erheblich mehr Aufwand und Durchlaufzeit bedeuten.

Schließlich stellt sich die Frage nach dem Unterschied beim Testvorgehen im Fall von Applikationen, die neu entwickelt werden und solchen, die im Zuge von Wartungsarbeiten oder agilem Vorgehen geändert werden.

Geht man von einem neuen System aus, lässt sich die Annahme nicht bestreiten, dass theoretisch das gesamte System getestet werden muss, um sicher alle Fehler zu finden. Die bereits genannten Strategien sollen dabei unterstützen. Werden nun allerdings im Zuge von Wartungs-

arbeiten (Fehlerbehebung oder geänderte beziehungsweise neue Anforderungen) nur bestimmte Bereiche des Systems modifiziert, könnte argumentiert werden, dass auch nur diese Bereiche zu testen seien. Gleichzeitig kann vorgebracht werden, dass durch die Änderungen an einer Stelle im Code aufgrund der Komplexität der Systeme auch andere Bereiche betroffen sein können, also Wechselwirkungen mit anderen Bereichen nicht ausgeschlossen werden können. Nimmt man dieses Risiko ernst, müsste – rein theoretisch – die gesamte Applikation in voller Detailtiefe erneut getestet werden. Das ist im Allgemeinen nicht notwendig, möglich und wirtschaftlich. Daher wird normalerweise ein „Regressionestest“ definiert, der alle relevanten Bereiche im Groben abdecken sollte, um die meisten Nebeneffekte identifizieren oder aber ausschließen zu können.

Schließlich sei an dieser Stelle noch auf den Begriff „Retest“ hingewiesen. Darunter ist der Test zu verstehen, der nach der Korrektur eines Fehlers vorgenommen wird, um sicher zu stellen, dass der Fehler erfolgreich aus dem System entfernt wurde.

3.3 Die Grenzen des Testens und Testendekriterien

Im Zuge der Qualitätssicherung ergibt sich die Schwierigkeit, dass initial weder bekannt ist, wie viele Fehler sich in der Software befinden, noch welcher Art oder wo sie zu finden sind. Dieser Umstand legt nahe, dass theoretisch die gesamte Software zu testen ist, um sicherzustellen, dass alle Fehler gefunden (und korrigiert) werden können. Dabei ergeben sich allerdings zwei Problemfelder.

Einerseits stellt sich die Frage, woran erkenntlich ist, ob die gesamte Applikation vollständig getestet wurde. Andererseits werden durch eine Unzahl von Input-Zuständen sehr schnell – selbst bei verhältnismäßig einfachen Programmen – die Grenzen des Machbaren erreicht.

Es herrscht somit das Bewusstsein vor, dass im Normalfall nicht vollständig getestet werden kann oder dies wirtschaftlich ist. Ebenso besteht die Erkenntnis, dass im Normalfall nicht jede mögliche Variante zu testen ist, um eine ausreichende Qualität zu gewährleisten. Daher gilt es, sinnvolle Kriterien zu definieren, um den idealen Zeitpunkt für das Testende bestimmen zu können.

Wie lange soll also getestet werden, um sicherzustellen, dass die Software der geforderten Qualität entspricht, wenn diese Zeit gleichzeitig möglichst kurz sein soll beziehungsweise mit der Einsicht, dass nicht unendlich lange getestet und korrigiert werden kann?

Wie in Kapitel 1.2 angedeutet bezieht sich Qualität auf Erwartungen an die Software, die zu Anfang definiert werden müssen. Entsprechend werden auch bei der Erstellung eines Testkonzepts beziehungsweise Testplans verschiedene Kriterien festgelegt, wann diese Anforderungen als erfüllt erachtet werden. Diese Kriterien sollen als Entscheidungsgrundlage für die formale Abnahme dienen und können verschiedene Aspekte gleichzeitig berücksichtigen.

Testendekriterien können das Erreichen eines gewissen Abdeckungsgrades (Codeabdeckung, Schnittstellenüberdeckung, Risikoabdeckung, Anwendungsfallüberdeckung etc.) beinhalten, sich auf nicht-funktionale Kriterien wie Performance, Usability, Security oder Wartbarkeit beziehen oder auf zuverlässigkeitsorientierte Maße. [28]

Gleichzeitig kann die Anzahl festgelegt sein, wie viele (bekannte) Fehler einer bestimmten

Kritikalität am Ende maximal im System verbleiben dürfen.

Für gewöhnlich bestehen Testendekriterien aus einer Kombination dieser Aspekte.

Nach [28] können sich zuverlässigkeitsorientierte Testendekriterien auf folgende Aspekte beziehen:

- Die Anzahl gefundener Fehler in Bezug auf die geschätze Gesamtfehleranzahl (etwa durch die Annäherung an diese oder das Erreichen einer vorgegebenen „Restfehleranzahl“)
- Die Ausfallrate (durch das Erreichen oder Unterschreiten eines vorgegebenen Werts)
- Die Fehlerdichte
- Gleichzeitig kann das Erreichen minimaler Kosten gemäß des verwendeten Modells mit einbezogen werden.

Das Konzept der Ausfallrate sowie die Fehlerdichte werden ab Kapitel 4 erläutert.

3.4 Software-Zuverlässigkeit in der Testphase

In Kapitel 3.3 ist Software-Zuverlässigkeit als ein Testendekriterium beschrieben worden. Andererseits wurde in Kapitel 1 Software-Zuverlässigkeit als ein Qualitätskriterium genannt, über das sichergestellt werden soll, dass das Produkt den Anforderungen der Benutzergruppen gerecht wird. Dabei sollen etwa die Anzahl der initialen Fehler, der Zuverlässigkeitslevel oder die Zeit und Ressourcen, um diese zu erreichen, geschätzt werden. Um das tun zu können, müssen geeignete Modelle gewählt und Daten gesammelt werden.

Damit Modelle sinnvoll verwendet werden können, werden Annahmen getroffen und müssen Daten entsprechend unter kontrollierten Bedingungen gesammelt werden. Das legt nahe, dass Teststrategie und Testmethodik mit den Annahmen eines Modells übereinstimmen müssen beziehungsweise bei der Wahl und Konzeption berücksichtigt werden sollten, wie auch [23] nahe legt.

Wie wir in Kapitel 3.2 erfahren haben, sollen in der Testphase möglichst schnell möglichst viele Fehler gefunden werden. Dazu werden die Strategien verfolgt, systematisch (durch Analyse der möglichen Input-Zustände), in geeigneter Reihenfolge beziehungsweise in Teststufen zu testen.

Umgekehrt soll zur validen Schätzung von Software-Zuverlässigkeit sichergestellt werden, dass so getestet wird, dass sinnvolle und auswertbare Daten gesammelt werden. Die Autoren von [23, 28] empfehlen dafür das Testen nach „operativen Profilen“ beziehungsweise Betriebsprofilen oder Userprofilen.

Die gleiche Ansicht wird auch in [17] vertreten. So schreiben die Autoren:

„Einzig ein statistischer Testansatz, der die Häufigkeit der Testfälle entsprechend des sogenannten Benutzungsprofils wählt und so den realen Betrieb des Systems simuliert, ist in der Lage, quantitative Approximationen der tatsächlichen Zuverlässigkeit des Systems zu erzeugen.“ [17]

Bei diesem Ansatz soll also das Verhalten der (späteren) User aus der Betriebsphase berücksichtigt werden. Demnach sind bestimmte Szenarien und Eingabewerte häufiger als andere. Entsprechend können den Eingabewerten unterschiedliche Wahrscheinlichkeiten zugeordnet werden, um so ein Betriebsprofil zu erstellen. [23] Die Autoren von [7] definieren dabei das operative Profil als Häufigkeitsverteilung für die relativen Wahrscheinlichkeiten in Bezug darauf, dass eine bestimmte Funktion des Programms ausgeführt wird. Der entscheidende Vorteil ergibt sich daraus, dass die Fehlerrate auf Basis dieser Daten mehr der Nutzung in Produktion entspricht und dadurch eine bessere Vorhersagekraft der Zuverlässigkeit für die Betriebsphase gewährleistet ist. [23]

Die Zielsetzung der beiden Vorgehensweisen ist demnach prinzipiell unterschiedlicher Natur. Während Testen nach den Strategien in Kapitel 3.2 im Allgemeinen zum Ziel hat, möglichst schnell möglichst viele Fehler zu finden, soll Testen nach Userprofilen theoretisch die tatsächliche Verwendung des Systems in der Betriebsphase nachstellen, um so die Software-Zuverlässigkeit für diese schätzen und vorhersagen zu können.

Insofern stellt sich die Frage, ob diese zwei Aspekte (Testen des Testens wegen und Testen zur Datengewinnung) widersprüchlich zueinander sind beziehungsweise inwieweit sie sich beeinflussen. Einerseits kann hinterfragt werden, ob Testen nach Userprofilen eine ähnliche Testabdeckung erreicht oder redundante Testfälle beinhaltet, andererseits wäre zu untersuchen, inwieweit die Reihung der Testfälle gemäß der in der Praxis verwendeten Teststrategien mit der Priorisierung im Fall von Benutzerprofilen übereinstimmt.

Die Autor/innen von [11] kritisieren jedenfalls an Vorgehensweisen, die lediglich darauf abzielen Fehler zu finden, dass die gefundenen Fehler möglicherweise unbedeutend sind, wenn sie in der Betriebsphase mit geringer Wahrscheinlichkeit auftreten. Die Fehlerkorrektur hätte dann kaum einen Einfluss auf die Zuverlässigkeit. Umgekehrt kritisieren die Autor/innen von [5] an Testen nach Betriebsprofilen, dass Ausfällen mit geringer Eintrittswahrscheinlichkeit zu wenig Beachtung geschenkt wird, so dass nach dieser Methode ab einem gewissen Zeitpunkt kaum eine Verbesserung der Zuverlässigkeit beobachtet werden kann. Gleichzeitig merken die Autor/innen an, dass das Vorgehen in der Praxis oft durch einen geringen Detaillierungsgrad der Betriebsprofile limitiert ist.

In jedem Fall wäre es sinnvoll genauer zu untersuchen, inwieweit beide Zielsetzungen gleichzeitig erreicht werden können. Derzeit scheint das noch nicht der Fall zu sein, da in [6] kritisiert wird, dass in den derzeitigen Modellen keine Informationen aus den Testfällen berücksichtigt werden, obwohl eine Tendenz Richtung Modell-basiertem Testen festzustellen ist.

Schließlich soll an dieser Stelle kurz überlegt werden, wie Software-Zuverlässigkeit in Bezug zu den verschiedenen Teststufen und -arten steht. In der zu Rate gezogenen Literatur ist zumeist davon die Rede, dass die Daten aus der Entwicklungs- oder Test-Phase bezogen werden. [6, 27, 23] Ebenso vermittelt die Empfehlung, im Zuge der Schätzung der Software-Zuverlässigkeit nach Userprofilen zu testen, dass ein möglicher Anwendungsbereich im User-Acceptance-Test angesiedelt ist. Auch wenn das die Anwendung in den anderen Teststufen nicht ausschließt, ist gewiss zwischen Nutzen und Aufwand abzuwägen. Da etwa der Unittest auf sehr granularer Ebene stattfindet, wäre eine Aussage über die Software-Zuverlässigkeit des Systems gewiss stark

begrenzt. Demgegenüber könnten im Systemintegrationstest wohl bereits relevante Schätzungen erzielt werden.

Gleichzeitig legen diese Überlegungen nahe, dass sich Software-Zuverlässigkeit tendenziell auf Funktionstests bezieht (im Gegensatz zu Performance- oder Sicherheitstests). Obgleich im Allgemeinen nichts dagegen spricht, diverse Fehler in die Schätzung der Software-Zuverlässigkeit mit einfließen zu lassen, sowie auch im Funktionstests etwa Performance-induzierte Fehler auftreten können, werden die verschiedenen Testarten für gewöhnlich getrennt durchgeführt und behandelt. Dementsprechend ist auch das Vorgehen, wie diese verschiedenen Qualitätskriterien getestet werden, unterschiedlich. Das kann ein Zusammenführen der Daten in ein gemeinsames Modell deutlich erschweren.

3.5 Aspekte zur Datenerfassung

Nach [23] werden in gängigen Software-Zuverlässigkeits-Modelle zwei Arten von Daten verwendet:

1. Zeitbereichsdaten: Aufzeichnung der Zeitpunkte, wann ein Fehler aufgetreten ist, oder der Zeit zwischen je zwei Ausfällen

2. Intervallbereichsdaten: Zählung der Fehler während eines festen Zeitraums (z.B. Testsession, Stunde, Woche, Tag).

In Bezug auf Intervallbereichsdaten handelt es sich bei den Daten um die Anzahl an Ausfällen in einem Intervall. Die Intervalle müssen dabei nicht abstandsgleich sein. Wenn beispielsweise der Zeitraum als Testsession definiert wurde, so können die einzelnen Sitzungen unterschiedlich lange dauern. [23] Nach [15] werden in manchen Modellen Ausfälle auch an der Anzahl an ausgeführten Testfällen gemessen.

Nach [23] erzielen derzeit Modelle beziehungsweise aktuelle Tools mit Zeitbereichsdaten eine höhere Genauigkeit in der Parameterschätzung. Bei diesen wiederum sind nach [27] Modelle, die sich auf Ausführungszeiten beziehen, akkurater als jene, die Zeit nach Kalendertagen messen. Allerdings ist damit ein höherer Aufwand in der Datenerfassung gegeben, insbesondere im Vergleich zu Intervallbereichsdaten. [23]

Dieser Umstand spiegelt sich auch in der Aussage von [12] wieder, wonach das Erfassen der Ausfallzeitpunkte gemäß Lehrbuch sei, während die Erfassung von Intervallbereichsdaten in der Praxis häufiger vorkommt.

4 Software Zuverlässigkeit: Mathematische Begriffe und Definitionen

Nachdem in den letzten Kapiteln der allgemeine Hintergrund bezüglich Software-Zuverlässigkeit umrissen wurde, sollen in diesem Kapitel die in diesem Zusammenhang wichtigsten wahrscheinlichkeitstheoretischen Begriffe eingeführt werden.

4.1 Wieso Zufall

„Zuverlässigkeit ist die Wahrscheinlichkeit dafür, dass das betrachtete Objekt in einem bestimmten Zeitraum unter bestimmten Einsatzbedingungen einwandfrei funktioniert. Die Zuverlässigkeit ist somit auf ein festes Zeitintervall bezogen und bedeutet die Wahrscheinlichkeit dafür, dass in diesem Zeitintervall kein Ausfall stattfindet, bei einem nicht reparierbaren Objekt ist das gleichbedeutend mit der Überlebenswahrscheinlichkeit.“[12]

Aus der Definition von Software-Zuverlässigkeit ist klar ersichtlich, dass das Auftreten von Fehlern beziehungsweise Ausfällen bei dem Versuch einer Quantifizierung als zufälliges Ereignis aufgefasst wird. Wieso ist diese Auffassung zulässig?

Betrachtet man Software, so könnte man annehmen, dass es durch Analyse des zugrundeliegenden Codes einfach und schlüssig möglich sein sollte, das Auftreten von Fehlern vorherzusagen. Theoretisch sollte bei genauerer Betrachtung klar sein – insbesondere im Fall von einfachen Programmen – wo mögliche Fehlerursachen liegen, und unter welchen Umständen die Fehlerwirkung – nämlich beim Ausführen des entsprechenden fehlerhaften Stücks Code – eintritt.

Handelt es sich um komplexere Software, wird es leichter einsehbar, dass die Vorhersage des Eintretens bestimmter Ausfälle (beziehungsweise das Erkennen ihrer Ursache im Vorhinein) kaum beziehungsweise nur unter großem Aufwand möglich ist. Zumeist wird die Fehlerursache erst erkannt, wenn die Fehlerwirkung bereits eingetreten ist. In seltenen Fällen kommt es vor, dass Fehler nicht reproduzierbar sind, so dass der genaue Auslöser unbekannt bleibt. So gibt es zwar eine Fehlerursache, diese ist jedoch aufgrund der Komplexität der Systeme und des Zusammenspiels verschiedenster Faktoren in diesem Fall nicht nachvollziehbar.

Gleichzeitig ist das konkrete User-Verhalten bei der Benutzung des Systems entscheidend für das Auftreten von Fehlern. So könnte beispielsweise ein Fehler (Fehlerursache) in einem bestimmten Bereich bekannt sein. Dennoch ließe sich nicht mit Sicherheit sagen, ob die entsprechende Fehlerwirkung eintritt beziehungsweise wann – je nachdem ob und wie der User diesen Bereich nutzt. Das ist für Software-Zuverlässigkeit insofern relevant, da es nicht allein um die Existenz von Fehlern, sondern vielmehr um das Risiko ihres Auftretens geht.

Auch wenn Software also durch einen intellektuellen Prozess entsteht und so scheinbar kontrollier- und vorhersehbar ist, kann man nach [20] auch bei Softwareausfällen von einem zufälligen Verhalten ausgehen. Softwareausfälle werden aufgrund vielfältiger unterschiedlicher Faktoren hervorgerufen, so dass nach [20] ein deterministisches Modell inpraktibel ist.

Das lässt sich nach [23] auch dadurch leicht nachzuvollziehen, dass identische Systeme, die unter ähnlichen Bedingungen betrieben werden – etwa bei Verwendung derselben Software durch verschiedene Benutzer – dennoch zu unterschiedlichen Zeitpunkten ausfallen. Mehr noch, nach [12] kann – obgleich jeder Fehler eine Ursache hat - nicht vorhergesagt werden, ob das System in einem beobachteten Zeitraum ausfällt, wann es ausfällt und – bei initialer Betrachtung - warum es ausfällt, da dies oft von dem komplexen Zusammenwirken verschiedener Faktoren abhängt.

Dennoch sollen Aussagen darüber getroffen werden, wie gut ein System funktioniert, um Risiken, Reparaturaufwände, Marktchancen etc. abschätzen zu können. Es wird also die Wahrscheinlichkeit betrachtet, dass das System über einen bestimmten Zeitraum funktioniert. [12]

4.2 Software Reliability: Mathematische Definition

Eine Möglichkeit, Software Reliability zu definieren, ist über die Wahrscheinlichkeit, dass ein System seine definierten Funktionen unter festgelegten Grenzen erfolgreich ausführen wird. [23]

Oder anders formuliert: Software Reliability bezieht sich auf die Wahrscheinlichkeit, dass eine bestimmte Software für einen definierten Zeitraum und in einer definierten Umgebung ohne Ausfälle funktioniert. [6, 23]

Wie in Kapitel 1.3 erläutert bezeichnet dabei ein Ausfall oder das Versagen des Systems (failure) das Unvermögen des Systems, die beabsichtigte Funktion auszuführen. Ein Software-Fehler (auch fault oder bug) ist dagegen der Defekt im Programmcode selbst, der zum Versagen des Programms führt, wenn dieses unter bestimmten Bedingungen ausgeführt wird. [23]

Ein Weg zur Quantifizierung der Software Reliability ist die Messung der aufgetretenen Fehler während der Entwicklung oder in der Produktion. [6, 23]

Die nun folgenden Definitionen sind gemäß [23].

Mathematisch kann Reliability als die Wahrscheinlichkeit beschrieben werden, dass ein System im Intervall von Null bis zur Zeit t erfolgreich funktioniert

$$R(t) = P(T > t), \quad t \geq 0$$

wobei T eine Zufallsvariable ist, die die Zeit bis zum (ersten) Versagen beziehungsweise Ausfall (time-to-failure) oder Versagenszeit (failure time) bezeichnet.

Die Größe $R(t)$ kann auch als Überlebenswahrscheinlichkeit oder Zuverlässigkeitsfunktion bezeichnet werden. [8]

Dabei wird vorausgesetzt, dass das System zu Beginn der Beobachtungen jedenfalls funktionsfähig ist, also

$$R(0) = P(T > 0) = 1 \tag{4.2.1}$$

Da davon auszugehen ist, dass das System nicht fehlerfrei ist, wird bei unendlich langer Betriebsdauer der Fehler jedenfalls einen Ausfall verursachen. Es kann also mit Sicherheit gesagt werden, dass das System irgendwann einmal ausfallen wird. Die Wahrscheinlichkeit für ein unendlich lange funktionsfähiges System ist somit gleich Null.

$$R(\infty) = P(T > \infty) = 0 \tag{4.2.2}$$

Demgegenüber ist die Unreliability $F(t)$, auch bezeichnet als Versagenswahrscheinlichkeit, definiert als die Wahrscheinlichkeit, dass das System bis zum Zeitpunkt t versagen wird. Also

$$F(t) = P(T \leq t), \quad t \geq 0$$

Oder auch

$$F(t) = 1 - R(t) \tag{4.2.3}$$

Entsprechend gilt auch hier die Bedingungen, dass zum Zeitpunkt $t = 0$ noch kein Ausfall stattgefunden hat, also

$$F(0) = P(T \leq 0) = 0$$

Ebenso gilt, dass das System mit Sicherheit für $t \to \infty$ ausfallen wird.

$$F(\infty) = P(T \leq \infty) = 1$$

Die Größe $F(t)$ ist somit die Verteilungsfunktion und kann auch als Versagensverteilung, Versagens- oder Ausfallwahrscheinlichkeit bezeichnet werden.

Abbildung 2 zeigt exemplarisch den Zusammenhang zwischen der Zuverlässigkeitsfunktion und der Versagensverteilung.

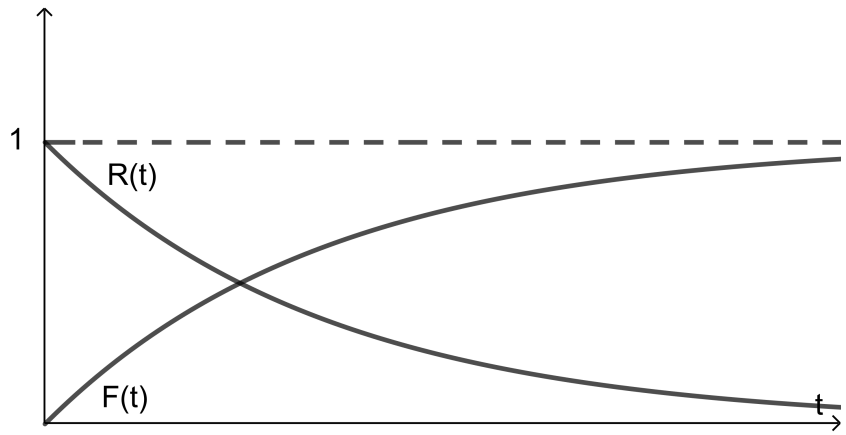

Abbildung 2: Zusammenhang zwischen der Zuverlässigkeitsfunktion und der Versagensverteilung

Ist die Verteilungsfunktion $F(t)$ differenzierbar, so ist die Wahrscheinlichkeitsdichtefunktion $f(t)$ gegeben durch deren Ableitung.

$$f(t) = F'(t)$$

Weiters gilt durch die Beziehung aus (4.2.3)

$$f(t) = -R'(t) \tag{4.2.4}$$

Weitere relevante Größen sind $MTTF$ als mittlere Zeit bis zum Ausfall (mean time to failure) beziehungsweise $MTBF$ als mittlere Zeit zwischen Ausfällen (mean time between failure). Genauere Erläuterungen dazu finden sich in Kapitel 4.8.

Auch die Begriffe der Ausfallrate oder Hazard Rate als Anzahl der Ausfälle pro Zeiteinheit, beziehungsweise der Wahrscheinlichkeit dafür, sind von großer Bedeutung. Diese werden in Kapitel 4.5 erklärt.

4.3 Jelinski-Moranda Modell: Schätzung der Versagensrate

Wie weiter oben erwähnt, kann die Zuverlässigkeit über die Messung tatsächlich aufgetretener Ausfälle geschätzt werden. Eine Möglichkeit dabei ist es, die Anzahl an Ausfällen pro Zeiteinheit zu betrachten und so die Ausfallrate zu schätzen. Das soll nun anhand eines Zuverlässigkeit-Modells veranschaulicht werden.

Eines der ältesten und einfachsten SRGM ist das Jelinski-Moranda Modell (J-M Modell). Es gehört zu jenen Modellen, die untersuchen, wie sich die Versagensrate verändert. Dabei wird eine Proportionalität zwischen der noch im Programm vorhandenen Fehleranzahl (Anzahl an Fehlerzuständen) und der Versagensrate angenommen. Das heißt, unter der Prämisse, dass ein gefundener Fehlerzustand sofort und ohne Folgefehler korrigiert wird, verringert sich die Anzahl an verbleibenden Fehlerzuständen und verändert somit auch die Versagensrate. [23, 8]

Folgende Annahmen werden für das Jelinski-Moranda Modell getroffen [23, 15]:

- Initial beinhaltet das System N Fehlerzustände (faults) – diese Anzahl ist unbekannt, wird aber als konstant angenommen
- Jeder Fehlerzustand ist unabhängig von den anderen und verursacht mit gleicher Wahrscheinlichkeit einen Ausfall (failure) während des Testens
- Die Zeitintervalle zwischen dem Auftreten zweier Ausfälle sind unabhängig voneinander
- Sobald ein Ausfall eintritt, wird der zugehörige Fehlerzustand behoben
- Durch das Beheben des Fehlerzustands wird kein neuer Fehler in das System eingeführt
- Die Versagensrate (failure rate) in dem Versagensintervall (failure interval) ist konstant und proportional zu der Anzahl an verbleibenden Fehlerzuständen im System

Das Jelinski-Moranda Modell basiert darauf, dass eine Proportionalität zwischen Fehleranzahl und Versagensrate angenommen wird. Sie bezeichnet dabei die Anzahl an gefundenen Ausfällen pro Zeiteinheit und lässt sich während der Testphase (zu verschiedenen Zeitpunkten) schätzen. Das heißt, das Modell geht davon aus, dass eine Konstante Φ bestimmt werden kann, die das Verhältnis zwischen der Anzahl N an Fehlerzuständen zur Versagensrate beschreibt. [8]

Nun soll das Modell in Anlehnung an [8] vorgestellt werden. Es wird folgende Notation verwendet.

N	anfängliche Anzahl an Fehlerzuständen im System
Φ	Proportionalitätskonstante
λ_0	anfängliche Versagensrate

Vor Beginn des Tests kann die Proportionalität somit folgendermaßen beschrieben werden

$$\lambda_0 = \Phi \cdot N,$$

Zum Zeitpunkt i gilt dann

$$\lambda_i = \Phi \cdot N_i,$$

Die Größe N_i ergibt sich aus der ursprünglichen Fehleranzahl, reduziert durch die Anzahl beseitigter Fehlerzustände:

$$N_i = N - \sum_{j=1}^{i-1} N_j$$

Die Versagensrate ab dem Zeitpunkt i kann somit auch folgendermaßen beschrieben werden:

$$\lambda_i = \Phi \cdot \left(N - \sum_{j=1}^{i-1} N_j \right)$$

Ausgangspunkt für dieses Modell ist die Schätzung der Versagensrate zu verschiedenen Zeitpunkten. Eine einfache Möglichkeit dazu ist die Beobachtung der Anzahl an Ausfällen je Zeitintervall während der Testphase, also

$$\text{Schätzwert von } \lambda_i \text{ als: } \frac{\text{Anzahl der Versagensfälle im Zeitintervall } i}{\text{Dauer des Zeitintervalls } i}$$

Hat man mehrere Schätzwerte λ_i, können damit die Proportionalitätskonstante Φ, die ursprüngliche Fehleranzahl N, sowie weitere N_i zu den Zeitpunkten i berechnet werden. Schließlich kann daraus geschätzt werden, wie viele Fehler noch im System sind und wie lange daher getestet werden muss, um diese zu beseitigen.

4.4 Jelinski-Moranda Modell: Überlebenswahrscheinlichkeit

Um nun die Zuverlässigkeit beziehungsweise Überlebenswahrscheinlichkeit R des Systems ermitteln zu können, soll diese in Zusammenhang zur Versagensrate gesetzt werden. Dazu gehe ich weiterhin nach [8] vor.

Es wird angenommen, dass die Versagensrate λ, also das Maß für die relative Abnahme der Überlebenswahrscheinlichkeit mit der Zeit, konstant ist. Weiters hängt die Überlebenswahrscheinlichkeit R von der Zeit ab. Daher gilt

$$R = R(t)$$

Da die Überlebenswahrscheinlichkeit mit der Zeit abnimmt, ist die Änderungsrate negativ und lässt sich somit als $-R'(t)$ beschreiben. Die relative Abnahme der Überlebenswahrscheinlichkeit lässt sich daher folgendermaßen beschreiben

$$-\frac{R'(t)}{R(t)}$$

Die Versagensrate λ beschreibt die Anzahl an Ausfällen pro Zeiteinheit. Gemäß der ursprünglichen Annahme in diesem Modell ist sie proportional zur verbleibenden Anzahl an Fehlern im System und konstant. Sie kann also mit der relativen Abnahme gleich gesetzt werden.

$$-\frac{R'(t)}{R(t)} = \lambda$$

Um $R(t)$ zu erhalten, soll die Gleichung nun integriert werden:

$$\int \frac{R'(t)}{R(t)}\mathrm{dt} = -\int \lambda \mathrm{d}t$$

Auf den linken Term soll nun die Substitution angewendet werden

$$\begin{aligned} u &= R(t) \\ \mathrm{d}u &= R'(t)\mathrm{d}t \end{aligned}$$

Also ist folgendes Integral zu berechnen

$$\int \frac{1}{u}\mathrm{d}u$$

Da $u = R(t) \geq 0$, folgt

$$= \log u$$

Durch Rücksubstitution erhalten wir

$$= \log R(t)$$

Da λ konstant ist, ergibt sich insgesamt

$$\log R(t) = -\lambda t + C$$

Durch Umformung anhand der Definition des Logarithmus erhalten wir schließlich

$$R(t) = e^{-\lambda t + C}$$

Um C zu bestimmen, können die Randbedingungen hinzugezogen werden. So gilt gemäß (4.2.1), dass das System am Beginn des Beobachtungszeitraums auf jeden Fall funktionsfähig ist. Daraus ergibt sich folgende Gleichung

$$R(t = 0) = 1 = e^{-\lambda \cdot 0 + C}$$

Daher muss gelten

$$C = 0$$

Weiters gilt gemäß (4.2.2), dass das System mit Sicherheit – jedenfalls nach unendlich langer Zeit – versagen wird. Also

$$R(t = \infty) = 0 = e^{-\lambda \cdot \infty + C}$$

Setzt man $C = 0$ in diese zweite Bedingung ein, so ergibt sich folgende Gleichung

$$R(t = \infty) = 0 = e^{-\lambda \cdot \infty + 0}$$

Das entspricht einer wahren Aussage. Somit ist die Überlebenswahrscheinlichkeit R im Fall einer konstanten Ausfallrate

$$R(t) = e^{-\lambda t} \tag{4.4.1}$$

Die Versagenswahrscheinlichkeit lässt sich durch die Verteilungsfunktion F als Exponentialverteilung beschreiben:

$$F(t) = 1 - e^{-\lambda t}$$

4.5 Ausfallrate und Hazard Rate

In Kapitel 4.3 konnte die Idee der Ausfallrate anhand eines einfachen Modells veranschaulicht werden. Dabei wurde aufgrund der vorgeschlagenen Annahmen von einer konstanten Ausfallrate ausgegangen. Gleichzeitig konnte in Kapitel 4.4 gezeigt werden, wie anhand dieses Schätzwertes die Zuverlässigkeit berechnet werden kann.

Im Folgenden sollen diese Konzepte verallgemeinert betrachtet werden, also wenn nicht davon ausgegangen werden kann, dass die Ausfallrate konstant ist. Gleichzeitig soll eine Differenzierung zwischen den Begriffen Hazard Rate und Ausfallrate stattfinden.

Dabei ist neben der Möglichkeit, über eine Schätzung der Ausfallrate die Zuverlässigkeit bestimmen zu können, nach [12] die Motivation für die Ausfallrate und die Hazard Rate jene, dass man die Zuverlässigkeit beziehungsweise Überlebenswahrscheinlichkeit von Objekten beurteilen möchte, die bereits eine gewisse Zeit lang funktionsfähig waren.

Nach [23] liegt die Wichtigkeit der Hazard Rate Funktion darin, dass sie die Veränderung der Failure Rate über die Zeit angibt. So kann die Zuverlässigkeit zweier verschiedener Systeme zu einem bestimmten Zeitpunkt identisch sein. Und dennoch können die Failure Rates bis zu diesem Zeitpunkt einen unterschiedlichen Verlauf annehmen.

Was ist also die Hazard Rate oder Failure Rate?

Für die folgenden Erläuterungen gehe ich nach [23] und [4] vor.

Wie in Kapitel 4.2 eingeführt, beschreibt die Zufallsvariable $T > 0$ die Zeit bis zum (ersten)

Ausfall und die Zuverlässigkeit $R(t) = P(T > t), \quad t > 0$ die Wahrscheinlichkeit dafür, dass bis zu diesem Zeitpunkt kein Ausfall stattfindet. Unter der Voraussetzung, dass die Ausfallwahrscheinlichkeit $F(t) = 1 - R(t)$ differenzierbar ist, erhalten wir außerdem die Ausfalldichte $f(t) = F'(t)$.

Die Wahrscheinlichkeit, dass ein Ausfall im gegebenen Intervall $[t_1, t_2]$ stattfindet, kann demnach durch die Ausfallwahrscheinlichkeit ausgedrückt werden.

$$P(t_1 \leq T < t_2) = \int_{t_1}^{t_2} f(t)\mathrm{d}t = F(t_2) - F(t_1)$$

Durch die Beziehung $F(t) = 1 - R(t)$ aus (4.2.3) ergibt sich

$$P(t_1 \leq T < t_2) = R(t_1) - R(t_2)$$

Die Wahrscheinlichkeit eines Ausfalls in diesem Zeitintervall, unter der Bedingung, dass das Objekt bis zum Zeitpunkt t_1 nicht ausgefallen ist, kann also wie folgt angegeben werden.

$$P(t_1 \leq T < t_2 | T > t_1) = \frac{P(t_1 \leq T < t_2)}{P(T > t_1)} = \frac{R(t_1) - R(t_2)}{R(t_1)}$$

Soll nun diese bedingte Wahrscheinlichkeit pro Zeiteinheit ermittelt werden, erhalten wir die Ausfallrate.

$$\frac{R(t_1) - R(t_2)}{(t_2 - t_1)R(t_1)}$$

Wird das Intervall $[t_1, t_2]$ durch $[t, t + \Delta t]$ angegeben, können wir die Ausfallrate über die Zuverlässigkeit darstellen,

$$\frac{R(t) - R(t + \Delta t)}{\Delta t R(t)}$$

beziehungsweise über die Ausfallwahrscheinlichkeit.

$$\frac{F(t + \Delta t) - F(t)}{\Delta t R(t)}$$

Demnach ist die Ausfallrate definiert als „die Wahrscheinlichkeit, mit der sich ein Ausfall pro Zeiteinheit in einem Zeitintervall $[t, t + \Delta t]$ ereignet, unter der Voraussetzung, daß [sic!] sich bis zum Zeitpunkt t kein Ausfall ereignet hat. Dies ist also die Häufigkeit, mit der sich Ausfälle im Intervall $[t, t + \Delta t]$ ereignen.“ [4]

Weiters ist die Hazard Rate (auch hazard function, instantaneous failure rate oder Ausfallintensität) definiert als der Grenzwert der Ausfallrate für $\Delta t \to 0$, also

$$h(t) = \lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)}$$

Betrachtet man den Ausdruck ohne $R(t)$, so entspricht dieser der Ableitung von $F(t)$, also

$$\lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t} = F'(t)$$

Somit können wir die Hazard Rate auch wie folgt darstellen.

$$h(t) = \frac{F'(t)}{R(t)}$$

Oder auch

$$h(t) = \frac{f(t)}{R(t)} \tag{4.5.1}$$

Entsprechend kann nach [4] die Hazard Rate auch als bedingte Ausfalldichte verstanden werden.

Während also die Ausfallrate die Rate an Ausfällen in einem Zeitintervall angibt, kann die Hazard Rate als ihr Grenzwert als die momentane Ausfallrate beschrieben werden. In manchen Texten werden die Begriffe allerdings zumindest punktuell synonym verwendet (etwa bei [12, 18]).

Nach [23] müssen alle Hazard Rate Funktionen zwei Bedingungen erfüllen:

$$h(t) \geq 0 \text{ für alle } t \geq 0$$
$$\int_0^\infty h(t)\mathrm{d}t = \infty$$

4.6 Hazard-Rate: Umformung zur Verteilungsfunktion

Ähnlich wie in Kapitel 4.4 kann nun auch von einer allgemeinen Form der Hazard Rate eine Umformung auf die Zuverlässigkeitsfunktion beziehungsweise die Verteilungsfunktion oder Dichtefunktion stattfinden. Nachfolgend sollen diese Umformung in Anlehnung an [18] vorgenommen werden.

Voraussetzung ist, sich anhand von Annahmen für eine Form von $h(x)$ zu entscheiden. Ebenso wird neben den Definitionen aus Kapitel 4.2 vorausgesetzt, dass die Verteilungsfunktion $F(x)$ differenzierbar ist.

Gemäß Gleichung (4.5.1) hat die Hazard Rate folgende Form.

$$h(x) = \frac{f(x)}{R(x)}$$

Aufgrund der Beziehung $R(x) = 1 - F(x)$ gemäß (4.2.3) und $f(x) = F'(x)$ gilt

$$h(x) = \frac{F'(x)}{1 - F(x)}$$

Um eine Umformung nach F vorzunehmen, soll die Gleichung integriert werden.

$$\int_0^t h(x)\mathrm{d}x = \int_0^t \frac{F'(x)}{1-F(x)}\mathrm{d}x$$

Wie in Kapitel 4.4 soll zur Auflösung des Integrals die Substitutionsmethode verwendet werden. Dazu wird folgende Substitution auf den rechten Term angewendet.

$$\begin{aligned} u &= F(x) \\ \mathrm{d}u &= F'(x)\mathrm{d}x \end{aligned}$$

Daher gilt für obiges Integral

$$= \int \frac{1}{1-u}\mathrm{d}u$$

Nun soll eine weitere Substitution angewendet werden

$$\begin{aligned} v &= 1-u \\ \mathrm{d}v &= -\mathrm{d}u \end{aligned}$$

Für obiges Integral erhalten wir also

$$-\int \frac{1}{v}\mathrm{d}v = -\log|v|$$

Durch Rücksubstitution erhalten wir

$$= -\log|1-u|$$

Da $0 \leq F(x) = u \leq 1$ gilt, können die Beträge weg gelassen werden.

$$= -\log(1-u)$$

Durch erneute Rücksubstitution erhalten wir schließlich

$$\int_0^t h(x)\mathrm{d}x = -\log[1-F(x)]|_0^t$$

Gemäß den Rechenregeln des Logarithmus können wir folgende Umformung vornehmen.

$$\log\frac{1-F(t)}{1-F(0)} = -\int_0^t h(x)\mathrm{d}x$$

Aufgrund der Annahme $F(0) = 0$ können wir den linken Term vereinfachen und erhalten somit

$$\log(1 - F(t)) = -\int_0^t h(x)\mathrm{d}x$$

Daraus ergibt sich weiters

$$1 - F(t) = R(t) = e^{-\int_0^t h(x)\mathrm{d}x}$$

Ist eine Umformung nach der Dichtefunktion gewünscht, so erhalten wir diese durch Ableiten gemäß der Kettenregel

$$f(t) = -R' = h(t)e^{-\int_0^t h(x)\mathrm{d}x}$$

4.7 Hazard-Rate für reparierbare Objekte

In Kapitel 4.5 basieren die Überlegungen in Bezug auf den Begriff Hazard Rate auf der Zuverlässigkeit im Sinn der Überlebenswahrscheinlichkeit. Das Konzept dahinter ist dann stimmig, wenn von nicht-reparierbaren Systemen ausgegangen wird. In der Software-Zuverlässigkeit sind aber auch reparierbare Systeme interessant. Das heißt, wenn ein ausgefallenes System vom Status nicht-funktionsfähig wieder zurück in den Status funktionsfähig wechseln kann. In diesem Fall interessiert also nicht sosehr die Zeit bis zum ersten Ausfall, sondern vielmehr die Zeit zwischen Ausfällen oder auch die Anzahl an Ausfällen in Intervallen. Genauere Überlegungen und Vergleiche zwischen nicht-reparierbaren und reparierbaren Systemen werden in Kapitel 5 angestellt.

In diesem Kapitel soll die Hazard Rate für reparierbare Systeme betrachtet werden. Dabei wird in der Literatur vermehrt der Begriff Failure Intensity oder auch Ausfallintensität verwendet. In dieser Arbeit werden die Begriffe gleichwertig verwendet.

Für die nachfolgenden Erläuterungen gehe ich vorwiegend nach [4] vor.

Im Zentrum der Betrachtung steht der Zufallsprozess der kumulativen Anzahl an Ausfällen in Abhängigkeit zur Zeit t. Dieser wird mit $N(t)$ bezeichnet. Der Erwartungswert beziehungsweise Mittelwert ist dann

$$m(t) = E(N(t))$$

Ebenso wie für den Begriff Hazard Rate bei nicht-reparierbaren Objekten (siehe Kapitel 4.5) wird die Ausfallintensität $\lambda(t)$ beziehungsweise $h(t)$ als augenblickliche Änderungsrate der erwarteten Anzahl an Ausfälle zur Zeit t bezeichnet. Also

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{P[N(t + \Delta t) - N(t) > 0]}{\Delta t}$$

Ist $m(t)$ differenzierbar, gilt die Beziehung

$$\lambda(t) = m'(t) = E'(N(t))$$

Umgekehrt kann der Mittelwert dann über die Ausfallintensität ausgedrückt werden.

$$m(t) = \int_0^t \lambda(s)\mathrm{d}s$$

Weiterhin gelten, wie in Kapitel 4.5 festgehalten, folgende Bedingungen für die Ausfallintensität von nicht-reparierbaren Systemen.

$$\lambda(t) \geq 0, \quad \text{für } t \geq 0$$
$$\int_0^\infty \lambda(t)\mathrm{d}t = \infty$$

Die Größe $m(t)$ kann auch als Erneuerungsfunktion, cumulative mean function (CMF) oder (kumulative) Mittelwertfunktion bezeichnet werden. [12] Wie wir im späteren Verlauf dieser Arbeit sehen werden, ist die Mittelwertfunktion durch ihren engen Zusammenhang mit der Ausfallintensität von zentraler Bedeutung.

4.8 MTTF

Unter $MTTF$ (mean time to failure) versteht man die mittlere Zeit bis zum ersten Ausfall. Analog dazu ist $MTBF$ (mean time between failure) definiert als die mittlere Zeit zwischen zwei Ausfällen. Für gewöhnlich ist nur einer der beiden Begriff notwendig, je nachdem, ob man von einem reparierbaren oder nicht-reparierbaren System ausgeht (siehe Kapitel 5.2).

Im Folgenden sollen die wichtigsten Überlegungen diesbezüglich in Anlehnung nach [23] vorgestellt werden. Demnach wird zur Berechnung der $MTTF$ vorausgesetzt, dass die Wahrscheinlichkeitsdichtefunktion gegeben ist. Die mittlere Zeit bis zum Ausfall entspricht dann dem Erwartungswert und kann über diesen berechnet werden.

Allgemein ist der Erwartungswert $E(X)$ der Zufallsvariablen X mit integrierbarer Dichtefunktion $f(x)$ gegeben durch

$$E(X) = \int_{-\infty}^{\infty} xf(x)\mathrm{d}x$$

Somit ist $MTTF$ definiert als

$$MTTF = \int_{-\infty}^{\infty} tf(t)\mathrm{d}t$$

Da die Funktion für $t < 0$ nicht definiert ist, gilt

$$MTTF = \int_0^\infty t f(t) \mathrm{d}t \tag{4.8.1}$$

Aus Gleichung (4.2.4) kennen wir die Beziehung der Dichtefunktion $f(t)$ zur Überlebenswahrscheinlichkeit $R(t)$

$$f(t) = -R'(t)$$

Setzten wir diese in die Definition der $MTTF$ ein, also Gleichung (4.8.1), so erhalten wir

$$MTTF = -\int_0^\infty t R'(t) \mathrm{d}t$$

Es kann die Regel zur partiellen Integration angewendet werden,

$$\int u' \cdot v = u \cdot v - \int u \cdot v'$$

wobei

$$u' = R'(t)$$

und

$$v = t$$

Daher erhalten wir

$$MTTF = [-tR(t)]|_0^\infty + \int_0^\infty R(t) \mathrm{d}t \tag{4.8.2}$$

Aus Gleichung (4.2.2) wissen wir, dass das System nach einer endlichen Zeitspanne jedenfalls ausfallen wird, insbesondere also $R(\infty) = 0$ gilt. Zusätzlich setzen wir voraus, dass

$$\lim_{t \to \infty} tR(t) = 0$$

Weiters wissen wir aus Gleichung (4.2.1), dass das System zum Zeitpunkt $t = 0$ auf jeden Fall funktionsfähig ist, also $R(0) = 1$ ist. Daher gilt

$$\lim_{t \to 0} tR(t) = 0$$

Dadurch ergibt sich für (4.8.2), dass die MTTF dem Integral der Zuverlässigkeitsfunktion ent-

spricht.

$$MTTF = \int_0^\infty R(t)\mathrm{d}t \tag{4.8.3}$$

Nach [23] sollte $MTTF$ verwendet werden, wenn die Verteilungsfunktion spezifiziert wurde, da der ermittelte Zuverlässigkeitsgrad von dieser abhängt. Dabei sollte berücksichtigt werden, dass für zwei Verteilungsfunktionen die gleiche $MTTF$ mit dennoch unterschiedlichen Graden an Zuverlässigkeit ermittelt werden kann. Das kann sich etwa daraus ergeben, wenn bei gleicher $MTTF$ die Verteilungsfunktionen durch eine Normalverteilung und eine Exponentialverteilung gegeben sind. [23]

Hingewiesen sei an dieser Stelle außerdem, dass die mittlere Zeit bis zum Ausfall – entgegen gängiger Fehlinterpretation – keine garantierte Lebensdauer darstellt, sondern der durchschnittlichen Zeit bis zum Ausfall entspricht. [23]

4.9 MTTF und MTBF in Zusammenhang mit der Ausfallrate

In Kapitel 4.3 beziehungsweise 4.4 haben wir ein Modell mit einer konstanten Ausfallrate kennengelernt. Dabei hat sich für die Zuverlässigkeitsfunktion in Gleichung (4.4.1) eine Exponentialfunktion ergeben.

$$R(t) = e^{-\lambda t}$$

Setzt man diese in Gleichung (4.8.3) ein, erhalten wir

$$\begin{aligned} MTTF = \int_0^\infty R(t)\mathrm{d}t &= \int_0^\infty e^{-\lambda t}\mathrm{d}t \\ &= \frac{1}{-\lambda} e^{-\lambda t}\Big|_0^\infty = 0 + \frac{1}{\lambda} \end{aligned}$$

Im Fall einer konstanten Ausfallrate gilt also

$$MTTF = \frac{1}{\lambda}$$

In Kapitel 6.1 werden wir zeigen, dass im Fall einer konstanten Ausfallrate beziehungsweise der Exponentialverteilung das System gedächtnislos ist. Diese Eigenschaft bedeutet, dass ein bereits „benutzes“ System so gut wie neu funktioniert. Die mittlere Zeit zwischen zwei Ausfällen entspricht in diesem Fall der mittleren Zeit bis zum ersten Ausfall. Die Größe $MTBF$ hat dann den gleichen Wert wie $MTTF$. Für λ konstant gilt also

$$MTBF = MTTF = \frac{1}{\lambda}$$

Ist die Ausfallrate $\lambda(t)$ allerdings als Failure Rate Funktion definiert, dann gilt nach [23] bezie-

hungsweise per Definition eine Ungleichheit. Für $\lambda(t)$ nicht konstant gilt somit

$$MTTF \neq \frac{1}{\lambda}$$

4.10 Mathematische Begriffe und Definitionen: Übersicht

In Tabelle 3 werden die Begriffe, die in diesem Kapitel definiert worden sind, zusammenfassend veranschaulicht.

5 Zuverlässigkeitskonzepte in unterschiedlichen Systemen

Viele gängige Software-Zuverlässigkeitsmodelle haben ihren Ursprung in Hardware-Zuverlässigkeitsmodellen. [6] Daher soll an dieser Stelle näher auf Gemeinsamkeiten und Unterschiede zwischen Hardware- und Software-Zuverlässigkeit eingegangen werden.

Gleichzeitig ergibt sich bei dieser Betrachtung die Relevanz, zwischen reparierbaren und nicht-reparierbaren Systemen zu differenzieren. Daher sollen in diesem Kapitel auch diese Konzepte beleuchtet werden.

5.1 Unterschiede Software Reliability und Hardware Reliability

Damit ein Computersystem funktionsfähig ist, müssen sowohl Hardware als auch Software korrekt funktionieren. Bei beiden Komponenten kann das Konzept der Zuverlässigkeit betrachtet werden, wobei gleichermaßen Qualität angestrebt wird. [23] In beiden Fällen sollen Schwachstellen erkannt und reduziert werden, um so eine Verminderung der Leistung zu verhindern.

Dennoch gibt es gravierende Unterschiede. Diese sind in der Software beziehungsweise Hardware selbst zu finden, also der Technik, so wie in weiterer Folge deren Testkonzepten, der Zuverlässigkeit und den eingesetzten Modellen. [23]

Ein Aspekt etwa ist, dass Software im Gegensatz zu Hardware nicht von sich aus mit der Zeit schlechter wird. Ausfälle in Hardware können beispielsweise aufgrund von Materialalterung entstehen, während die Ursachen von Softwareausfällen von Anfang an vorhanden sind und erst bei entsprechender Ausführung des Programms in Erscheinung treten. [23]

In folgenden Bereichen können Unterschiede festgehalten werden:

- Fehler- beziehungsweise Ausfallursache

  Bei Hardware können Ausfälle aufgrund von Materialalterung, Belastung, Missbrauch oder Umwelteinflüssen einerseits oder aufgrund von Konstruktions- oder Designfehleren auftreten. [6, 23]

  Dagegen treten bei Software Ausfälle (ähnlich wie bei Designfehlern eines komplexen Hardwaresystems) aufgrund inkorrekter Anforderungen, Fehlern im Design, der Logik, dem Code, der Kommunikation mit anderen Systemen oder ungeeigneten Eingabedaten auf. [6, 23]

Math. Ausdruck	Bezeichnung (Englisch)	Bezeichnung (Deutsch)	Kommentar
$R(t) = P(T > t)$, $T > 0$	software reliability	Software-Zuverlässigkeit, Zuverlässigkeitsfunktion	im Fall von nicht-reparierbaren Objekten auch Überlebenswahrscheinlichkeit
$F(t) = P(T < t)$, $T > 0$	cumulative distribution function (CDF)	Ausfallwahrscheinlichkeit, Versagenswahrscheinlichkeit, Wahrscheinlichkeitsvertreilung, Versagensverteilung, Verteilungsfunktion	für nicht-reparierbare Objekte auch Lebensdauerverteilung [12]
$f(t) = F'(t)$	probability density function (PDF), fault density	(Ausfall-) Wahrscheinlichkeitsdichte, Verteilungsdichte	
$MTTF$	mean time to failure	mittlere Zeit bis zum ersten Ausfall	relevant für nicht reparierbare Objekte
$MTBF$	mean time between failure	mittlere Zeit zwischen zwei Ausfällen	relevant für reparierbare Objekte
$\lambda(t)$	failure rate	Ausfallrate, Versagensrate	$\lambda(t)\mathrm{d}t$ entspricht der Wahrscheinlichkeit eines Ausfalls pro Zeiteinheit, im kleinen aber endlichen Intervall $\mathrm{d}t$ und unter der Bedingung, dass das System bis zum Zeitpunkt t funktioniert hat [12]; Häufigkeit, mit der Ausfälle im Intervall $\mathrm{d}t$ auftreten [4]; Oft wird der Buchstabe λ zur Unterscheidung im Fall einer konstanten Versagensrate verwendet, etwa bei [18, 8]
$h(t) = \lim_{\Delta t \to 0} \lambda(t)$	hazard rate, hazard rate function, hazard function, failure rate function, instantaneous failure rate, failure intensity, failure intensity function	Ausfallintensität	Grenzwert der Ausfallrate [12, 18, 23]; bedingte Ausfalldichte (bis zum Zeitpunkt t ist kein Ausfall aufgetreten) [4]
$h(t) = m' = \mu'(t)$	failure intensity	Ausfallintensität	augenblickliche Rate der Änderung der erwarteten Anzahl an Ausfällen zur Zeit t [4]; im Fall von reparierbaren Objekten auch Erneuerungsdichte [12]

Tabelle 3: Zusammenfassung Mathematische Begriffe

- Auftreten von Ausfällen

  Während bei Hardware Ausfälle – etwa aufgrund von Materialalterung – auch bei Nicht-Verwendung auftreten können, kann Software nur versagen, wenn sie verwendet wird, nicht jedoch, wenn das System ausgeschaltet ist. [23]

- Warnungen

  Vor Hardware-Ausfällen treten in der Regel Warnungen auf, was bei Software-Ausfällen normalerweise nicht der Fall ist. [23]

- Standards

  Nach [23] können Hardware-Komponenten standardisiert sein, während das bei Software-Komponenten selten der Fall ist.

  Erklärbar ist das möglicherweise durch höhere Normierungsanforderungen bei Hardware beziehungsweise einer geringeren Flexibilität diese zusammen zu schließen, sowie aufgrund eines größeren Spielraums wenn es darum geht nicht standardisierte Software-Komponenten in ein System zu integriert.

- Verbesserung der Zuverlässigkeit

  Hardware kann durch besseres Design, besseres Material, sowie die Anwendung von Redundanzen und beschleunigtem Lebenstest (accelerated life testing) verbessert werden. [23]

  Software kann durch vermehrtes Testen und die Korrektur so erkannter Fehler optimiert werden. Dadurch ändert sich die Zuverlässigkeit während des Testens kontinuierlich: Änderungen können Fehler eliminieren oder neue Fehler einführen. [23]

- Reparaturen

  Spricht man bei Hardware von Reparatur, so ist damit gemeint, dass der ursprüngliche Zustand wieder hergestellt wird, um die bisherige Funktion in gleicher Weise ausführen zu können, etwa durch das Ersetzen eines Bauteils durch eine gleichartige Komponente. Demgegenüber sollen bei der Reparatur von Software Fehler behoben werden, indem die Funktionsweise des Programms – also der Code – verändert beziehungsweise korrigiert wird. In diesem Sinn wird hier ein neues Produkt erzeugt. [23]

- Testen

  Gemäß [23] kann Hardware im Allgemeinen vollständig getestet werden. Es können also prinzipiell für alle in der Stichprobe enthaltenen Objekte Ausfälle beobachtet werden. Für Software ist theoretisch unendlich langes Testen notwendig.

- Wachstumsverhalten der Ausfallrate

  Typischerweise kann die Ausfallrate bei Hardware mit einer Badewannenkurve beschrieben werden. Initial ist die Ausfallrate durch eine Einbrennphase (ähnlich wie beim Software-Debugging) hoch. Nach dieser ersten Reduktion von Ausfällen folgt die Alterung des Systems. [23]

Dagegen ist die Ausfallrate von Software ohne die Berücksichtigung der Programmentwicklung nicht steigend. [23]

- Wachstumsverhalten der Zuverlässigkeit

  Bei Hardware nimmt die Zuverlässigkeit (abgesehen von initialen Hardware-Fehlern) mit der Zeit ab. [6]

  Bei Software nimmt die Zuverlässigkeit unter Berücksichtigung der Fehlerbehebung mit der Zeit zu.[6]

- Zuverlässigkeits-Modell

  Bei Hardware wird die Wahl einer Verteilungsfunktion aufgrund der Analyse der Ausfalldaten, sowie durch Erfahrung getroffen. Mithilfe von Anpassungstests wird eine geeignete Zuverlässigkeitsverteilung ermittelt. Die Ausfalldaten werden an die Verteilungsfunktion angepasst. Der Schwerpunkt liegt auf der Analyse der Ausfalldaten. [23]

  Demgegenüber wird bei Software das Modell und somit die Verteilungsfunktion anhand von Annahmen abgeleitet. Über die beobachteten Ausfalldaten werden dann die für das Modell notwendigen Parameter geschätzt. Der Schwerpunkt liegt auf der theoretischen Entwicklung eines passenden Modells, der Interpretation der Modellannahmen und der physikalischen Bedeutung der Parameter. [23]

  Die Wahl eines geeigneten Modells verläuft somit in den beiden Kontexten klassischerweise gemäß unterschiedlichen Prinzipien. Entsprechend kommen tendenziell verschiedene Schätzmethoden zum Einsatz. Die wichtigsten Prinzipien bezüglich Schätzungen in Zusammenhang mit Software-Zuverlässigkeit werden in Kapitel 7 vorgestellt.

Tabelle 4 fasst die eben genannten Aspekte zusammen.

5.2 Reparierbare und nicht-reparierbare Objekte

Mehr noch als die Unterscheidung zur Hardware-Zuverlässigkeit spielen Überlegungen zur Reparierbarkeit eines Systems eine Rolle. So ist es essentiell für die Wahl des richtigen Modells zu entscheiden, ob ein System reparierbar oder nicht reparierbar ist. Es muss also die Frage beantwortet werden, ob ein Ausfall bedingt, dass das System ab diesem Zeitpunkt nicht mehr funktionsfähig ist und daher vollständig ersetzt werden muss , oder ob etwa durch Tausch von Einzelteilen das System zurück in den Zustand der Funktionsfähigkeit versetzt werden kann.

Ein Objekt heißt nicht-reparierbar, wenn es vom Zustand $\bar{A}$ funktionsfähig, in den Zustand A nicht funktionsfähig wechseln kann, nicht aber umgekehrt.[12]

Es gilt also

$$\bar{A} \to A, \quad \text{aber nicht } A \to \bar{A}$$

Das trifft typischerweise auf einzelne Elemente oder Komponenten zu. [12]

Merkmal	Software	Hardware
Fehlerursache	inkorrekten Anforderungen, Design, Logik, Code, Schnittstellendefinitionen oder Eingabedaten	Materialalterung, Belastung, zufällige Ausfälle, Missbrauch, Umwelteinflüsse, Designfehler
Auftreten von Ausfällen	nur bei Verwendung	auch bei Nicht-Verwendung
Warnungen	in der Regel keine Warnung vor einem Softwareausfall	in der Regel Warnungen vor einem Hardwareausfall
Standards	selten standardisiert	kann standardisiert sein
Verbesserung der Zuverlässigkeit	Vermehrtes Testen und Korrektur erkannter Fehler	besseres Design, besseres Material, Anwendung von Redundanzen
Reparatur	produziert neue Software	stellt ursprünglichen Zustand wieder her
Testen	kann nicht vollständig getestet werden	kann in der Regel vollständig getestet werden
Wachstumsverhalten der Ausfallrate	nicht steigend	Badewannenkurve
Wachstumsverhalten der Zuverlässigkeit	R nimmt mit der Zeit zu	R nimmt mit der Zeit ab
Wahl des Zuverlässigkeitsmodells	aufgrund von Annahmen (analytisch)	Anpassung der Ausfalldaten an eine Verteilungsfunktion
Schätzmethoden	vornehmlich Punktschätzungen	zunächst Anpassungstests

Tabelle 4: Vergleich von Software und Hardware in Bezug auf Zuverlässigkeit [6, 23]

In diesem Fall ist Zuverlässigkeit (wie anfangs in Kapitel 4.2 definiert) als Wahrscheinlichkeit zu verstehen, dass ein Objekt bis zum Zeitpunkt $t > 0$ funktionsfähig bleibt. Sie wird dann auch als Überlebenswahrscheinlichkeit bezeichnet. Die Zufallsgröße betrifft die Zeit bis zum Ausfall (TTF, time to failure).

In Bezug auf ein Zufallsexperiment (Lebensdauertest) mit $n > 1$ Objekten kann $R(t)$ auch als Anteil der bis zu t nicht ausgefallenen Objekte verstanden werden. $F(t)$ kann somit als Anteil der bis zu t ausgefallenen Objekte verstanden werden.

Ein System heißt reparierbar, wenn es vom Zustand $\bar{A}$ funktionsfähig, in den Zustand A nicht funktionsfähig wechseln kann, und durch Reparatur vom Zustand A nach $\bar{A}$. Es kann also zu jedem beliebigen Zeitpunkt funktionsfähig oder ausgefallen sein. [12]

Es gilt also

$$\bar{A} \to A, \quad \text{sowie } A \to \bar{A}$$

Das trifft typischerweise auf den Zusammenschluss von Komponenten zu. Diese können dabei

parallel oder seriell geschaltet sein. [12]

Zuverlässigkeit ist hier als Wahrscheinlichkeit zu verstehen, dass ein Objekt über einen bestimmten Zeitraum funktionsfähig bleibt. Die betrachtete Zufallsgröße ist somit die Zeit zwischen Ausfällen (TBF, time between failures). Sie hängt sowohl von der Systemstruktur, als auch den Ausfallwahrscheinlichkeiten der Komponenten ab. [12]

Für nicht-reparierbare und reparierbare Systeme gilt gleichermaßen die Annahme, dass das Objekt zum Zeitpunkt $t = 0$ funktionsfähig ist (Es gilt also Gleichung (4.2.1)). [12]

Im Folgenden soll näher auf die beiden Konzepte eingegangen werden.

5.3 Datenerfassung für Reparierbare und nicht-reparierbare Objekte

Wie in Kapitel 3.5 erläutert, können Ausfalldaten über den jeweiligen Ausfallzeitpunkt, oder die Anzahl der Ausfälle für ein definiertes Zeitintervall gesammelt werden. Das gilt für reparierbare und nicht-reparierbare Objekte. Dabei werden für nicht-reparierbare Objekte $n > 1$ Objekte beobachtet und für reparierbare Objekte $n \geq 1$ Objekte. [12]

Da man sich außerdem in beiden Fällen für die Zeitspannen interessiert, in denen die Objekte funktionsfähig sind, müssen für reparierbare Systeme die Korrekturzeiten eliminiert werden. Fällt ein reparierbares System aus, wird für gewöhnlich Zeit benötigt, um es wieder funktionsfähig zu machen. Für die Anwendung eines Modells wird diese Reparaturzeit ausgenommen. Rechnerisch kehrt das System also sofort in den funktionsfähigen Zustand zurück. Das Modell wird sodann nur auf diese Funktionsdauer angewendet. [12]

Fällt die Wahl auf ein Modell für Zeitbereichsdaten, werden die Ausfallzeitpunkte dokumentiert, wobei $1 \leq r \leq n$ Ausfälle zu den Zeitpunkten $0 < t_1 \leq t_2 \leq \cdots \leq t_r \leq t^*$ beobachtet werden und $t^* > 0$ das Ende des Beobachtungszeitraums ist. [12]

Fällt die Wahl auf ein Modell für Intervallbereichsdaten, wird die Häufigkeit der Ausfälle in festen Zeitintervallen gezählt, wobei $0 \leq x_j$ die Anzahl der Ausfälle in aneinandergrenzenden, sich nicht überschneidende Intervalle $\Delta_j > 0$, mit j = 1, 2, . . . , m, ist. [12]

5.4 Wahrscheinlichkeitsmodelle für reparierbare und nicht-reparierbare Objekte

Im Fall von nicht-reparierbaren Objekten ist das einfachste Modell, das zum Einsatz kommt, die Exponentialverteilung. Dieses kann angewendet werden, wenn von einer konstanten Ausfallrate ausgegangen wird. Ist die Ausfallrate zeitabhängig, so wird laut [12] am häufigsten eine Weibull-Verteilung angewendeten. Die Weibull Verteilung ist eine Verallgemeinerung der Exponentialverteilung. [12]

Bei reparierbaren Systemen handelt es sich um einen alternierenden stochastischen Prozess, da das System vom Zustand funktionsfähig in den Zustand nicht-funktionsfähig springen kann und zurück. Es geht um einen Zählprozess beziehungsweise einen Punktprozess. Die Reparaturzeiten müssen dabei rechnerisch eliminiert werden. Nach [12] ist das bekannteste Modell ein Erneuerungsprozess. [12]

Wird für den alternierenden Prozess von einer konstanten Ausfallrate ausgegangen, kann ein homogener Poisson Prozess als Modell verwendet werden. In diesem Fall ist die Anzahl der Ausfälle Poisson-verteilt, während die Zeit zwischen zwei Ausfällen einer Exponentialverteilung folgt (siehe Kapitel 6.3.2). Ist die Ausfallintensität allerdings zeitabhängig, also $\lambda(t) > 0$, so ist das häufigste verwendete Modelle ein nicht-homogener Poisson Prozess (NHPP). [12]

Gehen wir nun davon aus, dass es sich bei Software um reparierbare Systeme handelt und dass die Ausfallrate nicht unbedingt konstant ist, sind somit NHPP Modelle für Software-Zuverlässigkeit von besonderem Interesse. Das spiegelt sich auch in der untersuchten Literatur wieder.

In Kapitel 6 werden die vier genannten Verteilungen näher beschrieben.

5.5 Zusammenfassung reparierbare und nicht-reparierbare Objekte

In Tabelle 5 sollen die Unterschiede zwischen nicht-reparierbaren und reparierbaren Objekten, beziehungsweise Komponenten und Systemen zusammengefasst dargestellt werden.

Komponente	System
nicht reparierbar	reparierbar
Element (Unit)	Zusammenschluss (parallel und/oder seriell) von Komponenten
Die Zuverlässigkeit ist die Wahrscheinlichkeit, dass ein Objekt bis zum Zeitpunkt $t > 0$ funktionsfähig bleibt (Überlebenswahrscheinlichkeit).	Die Zuverlässigkeit ist die Wahrscheinlichkeit, dass ein Objekt in einem bestimmten Zeitraum funktionsfähig ist.
Die Zufallsgröße ist die Zeit bis zum Ausfall.	Die Zufallsgröße ist die Zeit zwischen Ausfällen.
Komponenten lassen sich bezüglich R nicht weiter zerlegen.	Die Versagenswahrscheinlichkeit hängt von der Systemstruktur sowie der Ausfallwahrscheinlichkeit der Komponenten ab.
Bei konstanter Ausfallrate wird typischerweise die Exponentialverteilung verwendet.	Bei konstanter Ausfallrate wird typischerweise ein homogener Poisson Prozess (HPP) angenommen.
Bei einer zeitabhängigen Ausfallrate ist die häufigste Vereilung die Weibull-Verteilung.	Bei einer zeitabhängigen Ausfallrate wird ein nicht homogener Poisson Prozess (NHPP) angenommen.

Tabelle 5: Vergleich zwischen Komponente und System

6 Verteilungen

Im letzten Kapitel konnten wir feststellen, dass für verschiedene Objekte, Systeme und Annahmen verschiedene Modelltypen anwendbar sind. In diesem Kapitel sollen daher die vier wichtigs-

ten Wahrscheinlichkeitsverteilungen im Zusammenhang mit Zuverlässigkeit vorgestellt werden.

Zunächst werden die Verteilungen im Zusammenhang mit nicht-reparierbaren Objekten betrachtet, also die Exponentialverteilung für den Fall einer konstanten Ausfallrate, und die Weibull Verteilung für eine zeitabhängige Ausfallrate. Analog dazu werden im Anschluss zwei Verteilungen für reparierbare Systeme näher vorgestellt. Das heißt der homogene Poisson Prozess für eine konstante Ausfallrate, sowie der nicht homogene Poisson Prozess für den Fall einer zeitabhängigen Ausfallrate.

6.1 Exponentialverteilung

In Kapitel 4.3 und 4.4 haben wir das Jelinski-Moranda Modell kennengelernt. Dieses basiert auf der Exponentialverteilung. In diesem Kapitel soll noch einmal kurz auf ihre Kenngrößen und Eigenschaften eingegangen werden.

Dabei ist vorweg zu nehmen, dass die Exponentialverteilung in der Zuverlässigkeitstechnik aufgrund der konstanten Ausfallrate eine wichtige Rolle spielt. Laut [18] ist ihre Bedeutung in diesem Bereich mit der Relevanz der Normalverteilung in anderen Bereichen der Statistik vergleichbar. Allerdings gilt als wichtige Voraussetzung für die sinnvolle Verwendung dieser Verteilung, dass eine benutzte Komponente, die bisher noch nicht ausgefallen ist, so gut wie eine neue Komponente funktionieren soll. Das System soll also gedächtnislos sein. [23]

Für die folgenden Erläuterungen zur Exponentialverteilung in der Zuverlässigkeits-Analyse gehe ich nach [23] vor.

Wie bereits in Gleichung (4.4.1) festgehalten, ist die Zuverlässigkeit $R(t)$ im Fall einer Exponentialverteilung gegeben durch

$$R(t) = e^{-\lambda t}, \quad t \geq 0$$

Die Dichtefunktion ist demnach durch die Beziehung (4.2.4) $f(t) = -R'(T)$ gegeben als

$$f(t) = \lambda e^{-\lambda t}, \quad t \geq 0$$

Über die Definition der Hazard Rate Funktion aus Gleichung (4.5.1) kann die Ausfallrate ermittelt werden.

$$\begin{aligned} h(t) &= \frac{f(t)}{R(t)} \\ &= \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} \\ &= \lambda \end{aligned}$$

Dabei ist $\lambda > 0$ somit eine konstante Ausfallrate.

Teilweise wird die Schreibweise $\theta = 1/\lambda > 0$ bevorzugt, also

$$f(t) = \frac{1}{\theta} e^{-\frac{t}{\theta}}$$
$$R(t) = e^{-\frac{t}{\theta}}$$

Weiters gilt, wie in Kapitel 4.9 gezeigt, folgende Beziehung zur $MTTF$ beziehungsweise zum Erwartungswert

$$MTTF = \frac{1}{\lambda} = \theta$$

Eine weitere wichtige Eigenschaft ist die Gedächtnislosigkeit der Exponentialverteilung. Das heißt, dass die bedingte Wahrscheinlichkeit für die Lebensdauer einer Komponente, die bis zum Zeitpunkt s überlebt hat, identisch zu der einer neuen Komponente ist. Es wird auch der Ausdruck „used-as-good-as-new“ verwendet, basierend auf der Annahme, dass keine Alterung des Systems vorhanden ist.

Die Exponentialverteilung ist die einzige stetige Verteilungsfunktion, die gedächntislos beziehungsweise stochastisch unabhängig ist.

Es gilt folgende Gleichung

$$P(T \geq t) = P(T \geq t + s | T \geq s), \quad \text{für } t > 0, s > 0$$

Das kann folgendermaßen gezeigt werden.

$$\begin{aligned} P(T \geq t + s | T \geq s) &= \frac{P(T \geq t + s)}{P(T \geq s)} \\ &= \frac{R(t + s)}{R(s)} \\ &= \frac{e^{-\lambda(t+s)}}{e^{-\lambda s}} \\ &= e^{-\lambda(t+s)-(-\lambda s)} \\ &= e^{-\lambda t} = R(t) \\ &= P(T \geq t) \end{aligned}$$

6.2 Weibull Verteilung

Die Exponentialverteilung ist durch die Eigenschaft der Gedächtnislosigkeit limitiert und auf eine konstante Ausfallrate beschränkt. Die Weibull-Verteilung ist eine Verallgemeinerung der Exponentialverteilung mit bis zu drei Parametern und kann auch steigende und fallende Hazard Rates berücksichtigen. Sie ist also geeignet, um die Lebensdauer von Komponenten zu modellieren, die zeitabhängige Hazard Rate Funktionen haben. [23, 18]

Das ist beispielsweise dann der Fall, wenn Alterungserscheinungen nicht nur in Abhängigkeit von der Zeit sondern auch der Nutzungsintensität zu berücksichtigen sind. In Abbildung 4 findet

sich eine Darstellung für verschiedene Hazard Rate Funktionen.

Die Weibull-Verteilung kann aus dem Hazard Rate Konzept oder als asymptotische Extremwertverteilung hergeleitet werden. [18, 12]

Im Fall einer drei-parametrigen Weibull-Verteilung ist die Zuverlässigkeit $R(t)$ für $t \geq \gamma$ gegeben durch

$$R(t) = e^{-(\frac{t-\gamma}{\theta})^{\beta}}, \quad \text{für } t \geq \gamma \geq 0, \quad \beta > 0, \quad \theta > 0$$

Dabei handelt es sich bei γ um den Lageparameter, bei θ um den Skalierungsparameter und bei β um den Formparameter. Alle drei Parameter sind immer positiv. Durch die Verwendung verschiedener Parameter kann die Verteilung verschiedenen Verteilungsfunktionen folgen, wie der Exponentialverteilung oder der Normalverteilung. [23]

Die Dichtefunktion der drei-parametrigen Weibull-Verteilung ist gegeben durch

$$f(t) = \frac{\beta(t-\gamma)^{\beta-1}}{\theta^{\beta}} e^{-(\frac{t-\gamma}{\theta})^{\beta}}, \quad t \geq \gamma \geq 0$$

Abbildung 3 nach [18] zeigt verschiedene Weibull Dichtefunktionen für verschiedene Werte von β, mit den fixierten Parametern $\gamma = 0$ und $\theta = 1$.

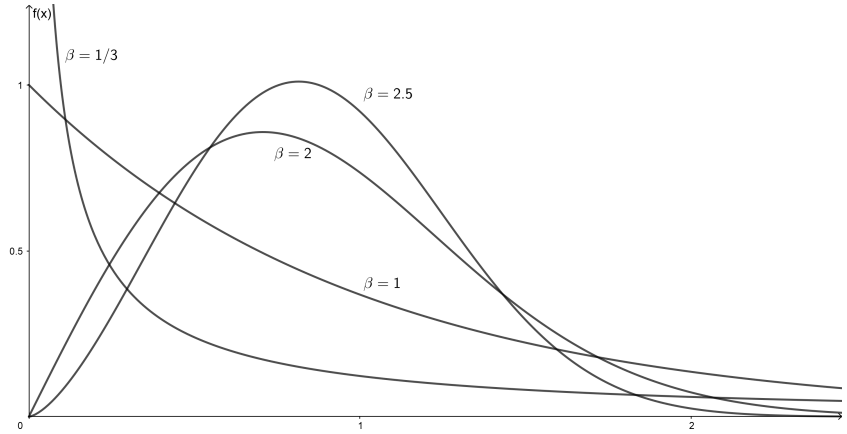

Abbildung 3: Weibull Dichtefunktionen nach [18]

Die Hazard Rate ist gegeben durch

$$h(t) = \frac{\beta(t-\gamma)^{\beta-1}}{\theta^{\beta}}, \quad \text{für } t \geq \gamma \geq 0, \quad \beta > 0, \quad \theta > 0$$
$$h(t) = 0, \quad \text{für } t < \gamma$$

Sie ist für $\beta < 1$ abnehmend, für $\beta > 1$ zunehmend und für $\beta = 1$ konstant. Für $\beta = 1$ folgt die Weibull-Verteilung somit der zwei-parametrigen Exponentialverteilung. [23] Abbildung 4 nach [18] veranschaulicht die verschiedenen Wertebereiche von β im Fall von fixierten Parametern $\gamma = 0$ und $\theta = 1$.

Wird die zwei-parametrige Weibull-Verteilung mit dem Lageparameter $\gamma = 0$ betrachtet, so

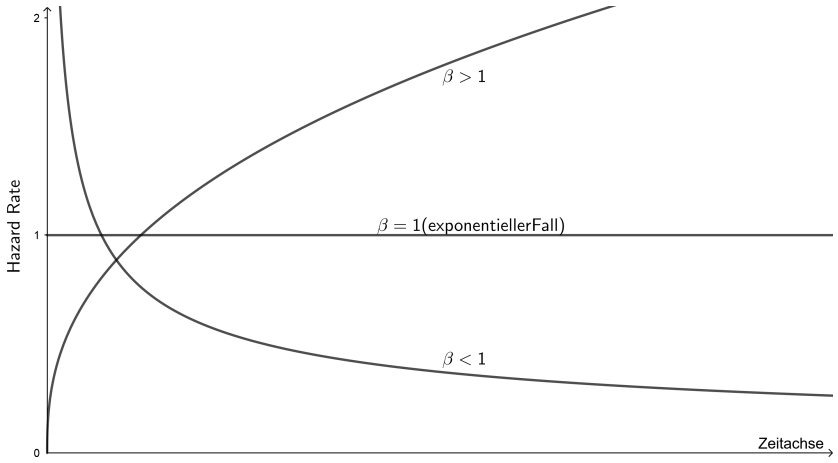

Abbildung 4: Verschiedene Hazard-Rate Funktionen der Weibull-Verteilung nach [18]

ist die Zuverlässigkeit gegeben durch

$$R(t) = e^{-(\frac{t}{\theta})^{\beta}}, \quad \text{für } t \geq 0, \quad \beta > 0, \quad \theta > 0$$

Besonders ist hier die Stelle $t = \theta$, da hier die Funktion unabhängig von den Parametern immer den gleichen Wert annimmt. [12]

$$R(t = \theta) = \frac{1}{e}$$

Daher wird θ auch als charakteristischer Wert bezeichnet. [12]

6.3 Poisson Prozess

Entgegen den bisherigen Verteilungen behandelt der Poisson Prozess eine diskrete Zufallsvariable. Während etwa die Exponentialverteilung die Zeit bis zu einem Ausfall betrachtet, wird beim Poisson Prozess die Anzahl an Ausfällen in einem Intervall gezählt. Er ist dafür geeignet den Punktprozess in Zusammenhang mit reparierbaren Systemen – also die augenblicklichen Änderungen der Ausfälle mit deren sofortiger Reparatur – zu beschreiben. Im Bereich der Zuverlässigkeitsanalyse ist der Prozess gut geeignet, da er Ereignisse mit kleinen Wahrscheinlichkeiten behandelt. [12] Eine Bezeichnung dafür ist daher auch Verteilung der seltenen Ereignisse.

Laut [23] wird der Poisson Prozess auch dann verwendet, wenn die Größe der Stichprobe unbekannt ist.

Ähnlich wie bei der Exponentialverteilung ist die Anzahl der Ausfälle nicht von der Zeit t, sondern der Länge s des Intervalls $(t, t + s]$ abhängig. Sie besitzt stationäre unabhängige Zuwächse. Die Ausfallintensität λ ist also auch hier konstant. Daher wird dieses Modell auch als homogener Poisson Prozess bezeichnet. [23]

Gleichzeitig steht die Poissonverteilung auch in Zusammenhang mit anderen Verteilungen.

Je kleiner λ, desto asymetrischer ist die Verteilung. Bei größeren λ nähert sie sich der Normalverteilung an. [12]

Die Wahrscheinlichkeitsdichtefunktion ist gegeben durch

$$P_m(t) = \frac{(\lambda t)^m e^{-\lambda t}}{m!}, \quad \text{für } m = 0, 1, 2, \ldots$$

Dabei ist λ eine konstante Ausfallrate und m die Anzahl der Ausfälle. Die Größe $P_m(t)$ ist dabei die Wahrscheinlichkeit von exakt m Ausfällen im Intervall t.

Die Zuverlässigkeit der Poissonverteilung $R(k)$ ist dann die Wahrscheinlichkeit von maximal k Ausfällen, gegeben durch

$$R_k(t) = \sum_{m=0}^{k} \frac{(\lambda t)^m e^{-\lambda t}}{m!}$$

Der Erwartungswert ist dann

$$\mu(t) = \lambda s$$

6.3.1 Herleitung

Im Folgenden soll die Herleitung des Poisson Prozesses nach [18] gezeigt werden.

Für die Poissonverteilung wird ein reparierbares System vorausgesetzt. Das heißt, das System ist augenblicklichen Änderungen – und zwar Ausfällen und deren sofortiger Reparatur – ausgesetzt. Es interessiert dabei die Gesamtzahl der Ausfälle. [18]

Die Wahrscheinlichkeit von m Ausfällen (beziehungsweise Reparaturen) im Intervall t soll als $P_m(t)$ bezeichnet werden. Dann ist $P_0(t)$ die Wahrscheinlichkeit von Null Ausfällen im Intervall t und $1-P_0(t)$ die Wahrscheinlichkeit von mindestens einem Ausfall in diesem Intervall. Wird eine konstante Ausfallrate $\lambda > 0$ angenommen, so kann diese für $t \to 0$ folgendermaßen beschrieben werden

$$\lambda = \frac{1 - P_0(t)}{t}$$

Die Wahrscheinlichkeit für mindestens einen Ausfall im kleinen Intervall h kann somit auch wie folgt beschrieben werden.

$$1 - P_0(h) = \lambda h + o(h)$$

Dabei stellt $o(h)$ eine Funktion $g(h)$ für $h > 0$ mit folgender Eigenschaft dar.

$$\lim_{h \to 0} \frac{g(h)}{h} = 0$$

Es soll nun von folgenden zwei Annahmen ausgegangen werden.

1. Der Prozess ist homogen und das Auftreten von Ereignissen ist unabhängig vom Auftreten

vergangener Ereignisse. Das System ist also gedächtnislos und die Wahrscheinlichkeit von mindestens einem Ausfall im Intervall h kann beschrieben werden als

$$p(h) = \lambda h + o(h), \quad \lambda > 0, \quad h \to 0.$$

2. Die Wahrscheinlichkeit von mindestens zwei Ausfällen im kleinen Intervall h ist vernachlässigbar, also $o(h)$.

Diese Annahmen führen zu einem Differentialgleichungssystem für $P_m(h)$.

Die Wahrscheinlichkeit von mindestens einem Ausfall im Intervall h kann als Summe der Wahrscheinlichkeiten dargestellt werden, also

$$p(h) = P_1(h) + P_2(h) + P_3(h) + \dots$$

Oder anders ausgedrückt

$$p(h) = P_1(h) + \sum_{m=2}^{\infty} P_m(h)$$

Aufgrund der zweiten Annahme gilt

$$\sum_{m=2}^{\infty} P_m(h) = o(h)$$

Somit ist

$$p(h) = P_1(h) + o(h)$$

Im Folgenden soll $P_m(h)$ für jedes m berechnet werden. Dazu wird $P_m(t+h)$ betrachtet. Die Wahrscheinlichkeit für genau m Ausfälle im Intervall $t+h$ lässt sich über die Kombination der Wahrscheinlichkeiten in den einzelnen Intervallen ermitteln.

$$P_m(t+h) = P_m(t)P_0(h) + P_{m-1}(t)P_1(h) + \sum_{k=2}^{m} P_{m-k}(t)P_k(h)$$

Dabei gilt

$$P_0(h) = 1 - p(h)$$

sowie

$$P_1(h) = p(h) - o(h)$$

Da die Wahrscheinlichkeit $P_{m-k} \leq 1$ ist, muss außerdem folgende Ungleichung gelten

$$\sum_{k=2}^{m} P_{m-k}(t)P_k(h) \leq \sum_{k=2}^{m} P_k(h) = o(h)$$

Somit ergibt sich für die Gleichung

$$P_m(t+h) = P_m(t)[1-p(h)] + P_{m-1}(t)[p(h)-o(h)] + o(h)$$

Nun soll die Beziehung $p(h) = \lambda h + o(h)$ aus der ersten Annahme verwendet werden. Damit erhalten wir

$$P_m(t+h) = P_m(t)[1-\lambda h - o(h)] + P_{m-1}(t)[\lambda h] + o(h)$$

Man betrachte nun den Differenzenquotienten. Wir erhalten

$$\begin{aligned}\frac{P_m(t+h)-P_m(t)}{h} &= \frac{P_m(t) - P_m(t)\lambda h - P_m(t)o(h) + P_{m-1}(t)\lambda h + o(h) - P_m(t)}{h} \\ &= -\lambda P_m(t) + \lambda P_{m-1}(t) + \frac{o(h)}{h}\end{aligned}$$

Betrachtet man diese Gleichung für das infinitesimale Intervall $h \to 0$, also den Differentialquotienten, so muss gelten

$$P'_m(t) = \lim_{h\to 0} \frac{P_m(t+h)-P_m(t)}{h} = -\lambda P_m(t) + \lambda P_{m-1}(t), \quad m = 1, 2, \ldots \tag{6.3.1}$$

Für den Fall $m = 0$ gilt

$$P_0(t+h) = P_0(t)P_0(h) = P_0(t)[1-p(h)]$$

Also gilt für den Differentialquotienten

$$\begin{aligned}\lim_{h\to 0} \frac{P_0(t+h)-P_0(t)}{h} &= \lim_{h\to 0} \frac{P_0(t)[1-p(h)] - P_0(t)}{h} \\ &= -\lim_{h\to 0} P_0(t)\frac{p(h)}{h}\end{aligned}$$

Aufgrund der ersten Annahme gilt

$$\lim_{h\to 0} \frac{p(h)}{h} = \lambda$$

Somit erhalten wir

$$P'_0(t) = -\lambda P_0(t). \tag{6.3.2}$$

Die Gleichungen (6.3.1) und (6.3.2) sind Differentialgleichungen, die über die Verwendung von

Laplace Transformationen gelöst werden können.

Sei $\tilde{P}(s)$ die Laplace Transformation von $P(t)$ definiert als

$$\tilde{P}(s) = L[P(t)] = \int_0^\infty e^{-st} P(t) \mathrm{d}t$$

sowie

$$\tilde{P}'(s) = L[P'(t)] = s\tilde{P}(s) - P(0).$$

Nun soll die Definition der Laplace Transformation auf Gleichung (6.3.2) angewendet werden. Wir erhalten daher

$$s\tilde{P}_0(s) - P_0(0) = -\lambda \tilde{P}_0(s).$$

Da das System zum Zeitpunkt $t = 0$ jedenfalls funktionsfähig ist, gilt laut Anfangsbedingung $P_0(0) = 1$. Wir erhalten somit durch Einsetzen und Umformen

$$\tilde{P}_0(s) = \frac{1}{s + \lambda}$$

Um $P_0(t)$ zu erhalten, wird die inverse Laplace Transformation auf $\tilde{P}_0(s)$ angewendet. Diese erhält man aus einer Tabelle von inversen Laplace-Transformationen, etwa [25]

$$P_0(t) = L^{-1}\{\tilde{P}_0(s)\} = e^{-\lambda}$$

Nun soll die Laplace Transformation auf Gleichung (6.3.1) $P'_m(t) = -\lambda P_m(t) + \lambda P_{m-1}(t)$ angewendet werden. Wir erhalten

$$s\tilde{P}_m(s) - P_m(0) = -\lambda \tilde{P}_m(s) + \lambda \tilde{P}_{m-1}(s)$$

Aufgrund der Anfangsbedingung $P_m(0) = 0$, für $m = 1, 2, \ldots$ ergibt sich

$$s\tilde{P}_m(s) = -\lambda \tilde{P}_m(s) + \lambda \tilde{P}_{m-1}(s)$$

Durch Umformung erhalten wir

$$\tilde{P}_m(s) = \frac{\lambda \tilde{P}_{m-1}(s)}{s + \lambda}$$

Löst man diese Gleichung durch Rekursion, erhalten wir

$$\tilde{P}_m(s) = \frac{\lambda^m}{(s + \lambda)^{m+1}}$$

Über die inverse Laplace Transformation [25, 26] dieser Gleichung erhalten wir die gesuchte

Größe $P_m(t)$

$$P_m(t) = \frac{e^{-\lambda t}(\lambda t)^m}{m!}, \quad \text{für } m = 0, 1, 2, \ldots \tag{6.3.3}$$

6.3.2 Zusammenhang zur Exponentialverteilung

Wie zu Beginn dieses Kapitels erwähnt stellt die Exponentialverteilung die Zeit bis zu einem Ausfall dar. Beim Poisson Prozess interessiert dagegen die Anzahl an Ausfällen in einem Intervall. Nun können wir für diesen Prozess Überlegungen zu den Zeitintervallen zwischen zwei Ausfällen anstellen. Es interessiert also die Wahrscheinlichkeit von Null Ausfällen im Intervall t. Das heißt

$$\begin{aligned} R_0(T \geq t) &= \sum_{m=0}^{0} \frac{(\lambda t)^m e^{-\lambda t}}{m!} = \\ &= e^{-\lambda t} \end{aligned}$$

Die Zeit zwischen Ausfällen im Poisson Prozess folgt somit der Exponentialverteilung. [12] Die beiden Verteilungen beschreiben den gleichen Ausfallprozess lediglich in Hinblick auf zwei verschiedene Aspekte.

6.4 Nicht homogener Poisson Prozess (NHPP)

Beim homogenen Poisson Prozess ist – ähnlich wie bei der Exponentialverteilung – eine Limitierung die Annahme einer konstanten Ausfallrate. Im Gegensatz dazu wird beim nicht homogenen Poisson Prozess (NHPP) die Ausfallintensität durch eine Funktion beschrieben und kann somit zeitabhängig sein. Wie in Kapitel 4.7 festgestellt steht die Ausfallintensität in engem Zusammenhang mit der Mittelwertfunktion. Je nach Modellannahmen kann diese unterschiedliche Formen annehmen. Sie beschreibt die erwartete Anzahl an Ausfällen bis zu einem definierten Zeitpunkt. Bei NHPP Modellen liegt daher ein Hauptmerkmal auf der Findung einer geeigneten Mittelwertfunktion. [23]

Wie auch der homogene Poisson Prozess ist der NHPP für reparierbare Systeme geeignet. Auch hier gilt die Annahme, dass die Ursache eines erkannten Ausfalls umgehend und mit Sicherheit behoben wird. Das System kehrt also laut Annahme sofort in einen funktionsfähigen Zustand zurück. [15]

Gleichzeitig sind NHPP Modelle sowohl für kontinuierliche als auch diskrete stochastische Prozesse geeignet. So basieren zwar die meisten Modelle auf Kalenderzeit oder Maschinenausführungszeit, doch einige verwenden etwa auch die Anzahl an Testfällen oder Fehlererfassungsperioden als Einheiten. [15]

Im Folgenden wird der nicht homogene Poisson Prozess in Bezug auf Software-Zuverlässigkeit in Anlehnung an [23] und [15] vorgestellt. Es wird von folgenden Annahmen ausgegangen.

- Die Zeitintervalle während des Ausfallvorgangs sind unabhängig voneinander. Das heißt, die Anzahl der Ausfälle im Intervall $(t, t + s]$ hängt vom aktuellen Zeitpunkt t, sowie der Länge s, nicht aber von der Vergangenheit ab.

- Die Ausfallrate ist gegeben als die Wahrscheinlichkeit von genau einem Ausfalls im Intervall $(t, t + \Delta t]$, also

$$P\{\text{genau ein Ausfall in } (t, t + \Delta t)\} = P\{N(t + \Delta t) - N(t) = 1\} = \lambda(t)\Delta t + o(\Delta t)$$

  Dabei bezeichnet $N(t)$ die Anzahl an beobachteten Ausfällen bis zum Zeitpunkt t und $\lambda(t)$ die Ausfallintensität.

- Die Wahrscheinlichkeit von mehr als einem Ausfall im kleinen Intervalls Δt ist vernachlässigbar, das heißt

$$P\{\text{mindestens zwei Ausfälle in } (t, t + \Delta t)\} = P\{N(t + \Delta t) - N(t) > 1\} = o(\Delta t)$$

Aufgrund der obigen Annahmen kann die Wahrscheinlichkeit für genau n Ausfälle im Zeitintervalls $(0, t)$ gegeben werden durch

$$P\{N(t) = n\} = \frac{[m(t)]^n}{n!} e^{-m(t)} \quad n = 0, 1, 2, \ldots$$

Wie in Kapitel 4.7 erwähnt, betrachtet die Mittelwertfunktion $m(t)$ den Erwartungswert der Summe der Ausfälle und kann durch die Ausfallintensitätsfunktion $\lambda(t)$ ausgedrückt werden.

$$m(t) = E[N(t)] = \int_0^t \lambda(s)\mathrm{d}s$$

Die Zuverlässigkeit $R(t)$ als Wahrscheinlichkeit, dass im Intervall $(0, t)$ keine Ausfälle eintreten, ist dann folgendermaßen gegeben

$$R(t) = P\{N(t) = 0\} = e^{-m(t)} = e^{-\int_0^t \lambda(s)\mathrm{d}s}$$

Weiters kann die Zuverlässigkeit $R(x|t)$ betrachtet werden. Sie bezeichnet die Wahrscheinlichkeit, dass im Intervall $(t, t + x)$ keine Ausfälle auftreten, unter der Bedingung, dass der letzte Ausfall zur Zeit t aufgetreten ist. Sie ist gegeben durch

$$\begin{aligned} R(x|t) &= P\{N(t + x) - N(t) = 0\} \\ &= \frac{R(t + x)}{R(t)} \\ &= e^{-[m(t+x)-m(t)]} \end{aligned}$$

Die Dichtefunktion ist sodann

$$\begin{aligned} f(x) &= -R'(x|t) \\ &= \lambda(t + x)e^{-[m(t+x)-m(t)]} \end{aligned}$$

Nach [15] ist ein Vorteil von NHPP Modellen, dass zwei oder mehrere bestehende NHPP Modelle leicht miteinander integriert werden können, indem die Mittelwertfunktionen zusammengefasst werden. Die Versagensintensität des überlagerten Prozesses entspricht dann der Summe der Versagensintensitäten der zugrunde liegenden Prozesse. [15] Demnach ist es relativ einfach mehrere Modelle zu kombinieren.

6.5 Besonderheiten der Wahrscheinlichkeitsmodelle in der Software-Zuverlässigkeit

Wie vielleicht bereits an der Auswahl der beschriebenen Verteilungen ersichtlich wurde, entsprechen die Wahrscheinlichkeitsmodelle in der Zuverlässigkeitsanalyse nicht den typischen Anwendungsgebieten der Stochastik. Laut [12] ist das wohl bekannteste und möglicherweise meistverwendete Modell die Normalverteilung. Diese ist für die gesamten reellen Zahlen definiert, die Dichtefunktion ist symmetrisch um den Erwartungswert, an dem sie auch ihr Maximum hat. Weiters werden normalerweise vollständige Stichproben mit $n > 1$ Beobachtungswerten und einem möglichst großen Stichprobenumfang ausgewertet. [12]

Demgegenüber ist nach [12] die in der Zuverlässigkeitstechnik am häufigsten angewendete Verteilung die Exponentialverteilung. Typischerweise werden nur die positiven reellen Zahlen betrachtet, die Dichtefunktion ist nicht symmetrisch und weist ihr Maximum bei Null auf. Im Allgemeinen wird versucht, den Stichprobenumfang n möglichst klein zu halten, da es sich insbesondere bei Hardware-Zuverlässigkeit um eine zerstörende Prüfung handelt. Typischerweise handelt es sich daher auch um eine zensierte Stichprobe, da etwa der Zeitraum auf $t^* > 0$ begrenzt wird und $r < n$ Ausfälle beobachtet werden. Gleichzeitig ist die Ausfallwahrscheinlichkeit sehr klein, wodurch die Anzahl an Ausfälle auf die Anzahl der Stichprobenelemente gering ausfällt. In der Hardware-Zuverlässigkeit werden daher beschleunigte Lebenstests angewendet. Das heißt, dass durch Überlastung der Ausfallprozess beschleunigt wird, um in einem verhältnismäßig kurzen Beobachtungszeitraum dennoch ausreichend Ausfälle beobachten zu können. Im Anschluss werden die beobachteten Werte auf eine Normalbelastung zurück gerechnet. [12]

7 Schätzmethoden

Wie in Kapitel 5.1 erwähnt, ist eine der Differenzen zwischen der Hardware-Zuverlässigkeitstechnik und der Software-Zuverlässigkeitstechnik die Herangehensweise bei der Wahl des Modells. So wird für die Ermittlung der Hardware-Zuverlässigkeit die Verteilungsfunktion aufgrund der beobachteten Ausfalldaten ermittelt, während in der Software-Zuverlässigkeitsanalyse das Modell anhand von Annahmen gewählt wird. Entsprechend werden verschiedene Konzepte des Schätzens verwendet.

Betrachtet man nun die verschiedenen Schätzmethoden in der Statistik im Allgemeinen, so können drei wichtige Konzepte unterschieden werden. Einerseits können die Parameter eines bereits bekannten oder angenommenen Modells geschätzt werden (Punktschätzung). Für diese können dann in weiterer Folge die Konfidenzintervalle geschätzt werden (Intervallschätzung), also

Intervalle, die mit einer gewissen Wahrscheinlichkeit den tatsächlich gesuchten Wert beinhalten. Schließlich können bei unbekannten Verteilungsfunktionen die beobachteten Daten mit Modellen verglichen und so geprüft werden, ob sich die Daten durch diese Verteilungen erklären lassen (Anpassungstests).

In der Hardware-Zuverlässigkeit kommen vornehmlich Anpassungstests zum Einsatz, während in der Software-Zuverlässigkeit Punktschätzungen von besonderer Bedeutung sind. Im Folgenden soll daher näher auf Punktschätzungen eingegangen werden. Das heißt, dass einerseits wichtige Eigenschaften von Punktschätzungen aufgezeigt werden. Andererseits werden die zwei Methoden, die in der Software-Zuverlässigkeit vornehmlich zum Einsatz kommen – nämlich das Maximum Likelihood Prinzip sowie die Methode der kleinsten Quadrate – und ihre Beziehung zueinander vorgestellt.

7.1 Punktschätzungen

Bei einer Punktschätzung wird bereits von einer bestimmten Verteilungsfunktion ausgegangen. Über die gesammelten Ausfalldaten sollen die unbekannten Parameter der Verteilung geschätzt werden. [23]

Da es verschiedene Möglichkeiten gibt, um das zu erreichen, ist es notwendig zu überlegen, welche Eigenschaften eine Punktschätzung aufweisen sollte. Im Folgenden werden diese gemäß [23] erläutert.

Dabei sind $x_1, x_2, \ldots, x_n$ die beobachteten Werte von $X_1, X_2, \ldots, X_n$. Es soll eine Funktion $h(X_1, X_2, \ldots, X_n)$ gefunden werden, so dass der Wert $h(x_1, x_2, \ldots, x_n)$ eine gute Punktschätzung des gesuchten Parameters ist.

Folgende Eigenschaften sind Qualitätskritierien für eine gute Punktschätzung.

- Erwartungstreue
- Konsistenz
- Effizient (minimale Varianz)
- Suffizienz

Definition Erwartungstreue

Für eine gegebene positive ganze Zahl n heißt die Größe $Y = h(X_1, X_2, \ldots, X_n)$ erwartungstreuer Schätzer des Parameters Θ, wenn der Erwartungswert von Y gleich dem Parameter Θ ist, also

$$E(Y) = \Theta$$

Definition Konsistenz

Die Größe Y heißt konsistenter Schätzer des Parameters Θ, wenn Y für $n \to \infty$ stochastisch gegen den Parameter Θ konvergiert. Das heißt für jede beliebig kleine positive Zahl ϵ gilt

$$\lim_{n\to\infty} P(|Y - \Theta| \leq \epsilon) = 1$$

Definition Effizienz

Ein erwartungstreuer Schätzer Y des Parameters Θ heißt effizient, wenn die Varianz von Y kleiner oder gleich der Varianz jedes anderen erwarungstreuen Schätzers von Θ ist.

Definition Suffizienz

Die Größe Y heißt suffizient für Θ, wenn die bedingte Verteilung von X für $Y = y$ unabhängig von Θ ist.

7.2 Maximum Likelihood Schätzung (MLS)

Nach [12] ist eine der bekanntesten Methoden zum Schätzen von Parametern von Verteilungsfunktionen das Maximum Likelihood Prinzip beziehungsweise die Maximum Likelihood Schätzung (MLS). Die Idee dahinter ist, jene Werte für die gesuchten Parameter zu ermitteln, die die tatsächlichen Beobachtungswerte mit der höchsten Wahrscheinlichkeit erzeugen würden. Dabei ist zu beachten, dass die gesuchten Parameter nicht etwa zufällig, sondern nur unbekannt, also fest, sind.

Die Maximum Likelihood Schätzung ist eine Punktschätzung mit den Eigenschaften, dass die Schätzer asymptotisch erwartungstreu, konsistent und für große Stichproben asymptotisch normalverteilt sind. [12]

Für die Beschreibung der Methode gehe ich nach [12] und [23] vor. Es sollen nur kontinuierliche Zufallsvariablen X betrachtet werden. Folgende Notation wird verwendet.

$\Theta = (\Theta_1, \ldots, \Theta_m)$	zu schätzender Parameter
$F(x\|\Theta)$	Wahrscheinlichkeitsmodell
$f(x\|\Theta)$	Verteilungsdichte
$x_1, \ldots, x_n$, $n > 1$	unabhängige Beobachtungswerte
$f(x_i\|\Theta)$	zum i-ten Beobachtungswert gehörende Verteilungsdichte

Die Likelihood-Funktion ist das Produkt der Verteilungsdichten je Beobachtungswert, also

$$L(\Theta|x_1, \ldots, x_n) = \prod_{i=1}^{n} f(x_i|\Theta) \tag{7.2.1}$$

Der Schätzwert $\hat{\Theta}$ ist dann der plausibelste Parameterwert, also das Maximum dieses Funktion. Um diesen zu erhalten, kann die Funktion (7.2.1) nach Θ differenziert und gleich Null gesetzt werden.

$$\frac{d}{d\Theta} L(\Theta|x_1, \ldots, x_n) = 0$$

Für den Schätzwert $\hat{\Theta}$ erhält die Stichprobe die größte Plausibilität.

Die Berechnung der Schätzwerte vereinfacht sich zumeist, wenn man die Likelhood-Funktion logarithmiert und erst danach differenziert. Dadurch erhält man die Summe der logarithmierten

Verteilungsdichten anstatt des Produkts.

$$\log L(\Theta|x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \log f(x_i|\Theta) \tag{7.2.2}$$

Dieses Verfahren ist deshalb möglich, da der Logarithmus eine streng monoton steigende Funktion ist und daher keinen Einfluss auf die Position des Maximum hat. Die Gleichungen (7.2.1) und (7.2.2) haben ihre Maxima also an der gleichen Stelle.

Im Allgemeinen kann ein Maximum Likelihood Schätzer $\hat{\Theta}$ mithilfe folgender Schritte gefunden werden.

1. Man finde die mehrdimensionale Dichtefunktion $L(\Theta|X)$
2. Man logarithmiere die Likelifunktion und erhalte $\log L$
3. Man berechne die partiellen Ableitungen von $\log L$ je Parameter
4. Man setze die partiellen Ableitungen gleich Null
5. Man löse das erhaltene Gleichungssystem, um die beziehungsweise den Parameter zu erhalten.

In Kapitel 8 wird dieses Vorgehen für verschiedene Verteilungen und Modelle beispielhaft veranschaulicht.

7.3 Methode der kleinsten Quadrate

Neben dem Maximum Likelihood Prinzip ist nach [23] auch die Methode der kleinsten Quadrate ein beliebtes Schätzverfahren im Bereich der Software Zuverlässigkeit. Bei diesem Verfahren soll der optimale Schätzwert beziehungsweise die optimale Schätzlinie dadurch ermittelt werden, dass der quadratische Abstand der Datenpunkte zu dieser Schätzung (beziehungsweise den bedingten Erwartungswerten) minimiert wird.

Es handelt sich um eine Punktschätzung für geordnete Stichproben. [18]

Zumeist wird diese Schätzmethode zur Kurvenanpassung in Regressionsmodellen verwendet. Dabei soll der (lineare) Zusammenhang zwischen dem Erwartungswert der abhängigen Zufallsvariablen Y und einer oder mehrerer Einflussgrößen $X = (x_1, x_2, \ldots, x_m)$ betrachtet werden. [12]

Voraussetzung ist, dass der Messfehler selbst normalverteilt ist mit Erwartungswert gleich Null. Das heißt insbesondere, dass vorausgesetzt wird, dass es keine Ausreißer bei den beobachteten Datensätzen gibt.

Für die folgende Darstellung der Methode gehe ich vornehmlich nach [12] vor.

Für die lineare Regression gemäß dem Gauß-Markov-Modell wird eine Gleichung des folgenden Typs vorausgesetzt

$$E[Y|x] = \Theta_1 x_1 + \Theta_2 x_2 + \cdots + \Theta_m x_m$$

- $E[Y|x]$ ist der Spaltenvektor der bedingten Erwartungswerte für die abhängige Zufallsvariable Y
- $X = (x_1, \ldots, x_m)$ ist Zeilenvektor, wobei x_j auch einfache Funktionen sein können (zum Beispiel x^2, $\log x$, $\sqrt{x}$)
- Der Spaltenvektor $\Theta = (\Theta_1, \ldots, \Theta_m)'$ mit $j = 1, \ldots, m$ beinhaltet die zu schätzenden Modellparameter

Da die beobachteten Werte y_i, $i = 1, \ldots, n$ vom erwarten Wert abweichen, kann ein zufälliger Fehler ε_i angenommen werden. Also

$$\varepsilon_i = y_i - E[Y|x]$$

Es ergibt sich also für jedes y_i, $i = 1, \ldots, n$

$$y_i = \Theta_1 x_1 + \cdots + \Theta_m x_m + \varepsilon_i.$$

Der Schätzer $\hat{\Theta}$ für den gesuchten Parameter Θ soll dann so bestimmt werden, dass die Summe der quadratischen Abweichungen zum Erwartungswert minimal ist, also die Funktion Q ihr Minimum hat.

$$Q = \sum_{i=1}^{n} [y_i - Y]^2 \tag{7.3.1}$$

Dazu können die partiellen Ableitungen ermittelt und Null gesetzt werden.

Für stochastisch unabhängige y_i ist die Varianz $Var[y_i] = \sigma^2$ und die Kovarianz $Cov[y_i, y_j] = 0$, $i \neq j$. Die Kovarianzmatrix hat die Gestalt

$$\sum_n = \sigma^2 I$$

Die Größe I bezeichnet die $n \times n$-Einheitsmatrix.

Über die partiellen Ableitungen der Gleichung (7.3.1) und das Nullsetzen dieser erhält man für diesen Fall folgende Normalgleichung

$$S = (Y - X\Theta)'(Y - X\Theta)$$

Der Schätzwert $\hat{\Theta}$ ist somit

$$\hat{\Theta} = (X'X)^{-1}X'Y$$

Um zu prüfen, ob es sich bei dem Schätzwert um ein globales Minimum handelt, soll die Funktion Q am „Rand“ betrachtet werden. Da die Kovarianzmatrix im Fall von unabhängigen y_i diagonalisierbar ist, kann auch $y_i - Y$ in der Form $A\Lambda A^{-1}$ dargestellt werden, wobei $(A\Lambda A^{-1})^2 = A\Lambda^2 A^{-1}$.

Die Diagonalmatrix Λ^2 ist positiv definit, wenn $\sigma \neq 0$. Damit ist auch $A\Lambda^2 A^{-1} > 0$. Insbesondere

$$\lim_{\Theta \to \infty} Q = \lim_{\Theta \to \infty} \sum_{i=1}^{n} [y_i - Y]^2 = +\infty$$

Da $\hat{\Theta} < \infty$, ist der Schätzwert globales Minimum.

Sind die beobachteten Werte y_i stochastisch abhängig beziehungsweise korreliert, so gilt für die Varianz $Var[y_i] = \sigma^2$ und für die Kovarianz $Cov[y_i, y_j] = G, i \neq j$. Dabei ist die Matrix G symmetrisch, positiv definit und nicht-singulär. Die Kovarianzmatrix hat dann die Gestalt

$$\sum_{n} = \sigma^2 G$$

Erneut wird das Minimum der Gleichung (7.3.1) gesucht. Man erhält die folgende Normalgleichung

$$S = (Y - X\Theta)' G^{-1} (Y - X\Theta)$$

Der Schätzwert $\hat{\Theta}$ lässt sich somit folgendermaßen berechnen

$$\hat{\Theta} = (X' G^{-1} X)^{-1} X' G^{-1} Y$$

Es ist festzuhalten, dass die Schätzer erwartungstreu sind. Im Fall von unkorrelierten Datensätzen ist außerdem die Eigenschaft der Effizienz erfüllt. [12]

7.4 Überschneidung der Methode der kleinsten Quadrate mit dem Maximum Likelihood Prinzip

Voraussetzung für die Anwendung der Methode der kleinsten Quadrate ist, dass der Fehler ε als symmetrisch normalverteilt angenommen wird. In diesem Fall ergibt die Maximum Likelihood Schätzung den gleichen Schätzwert wie die Methode der kleinsten Quadrate. Kann für den Fehler keine Normalverteilung vorausgesetzt werden, ist die Maximum Likelihood Methode zu bevorzugen. [12]

Im Folgenden soll die Methode der kleinsten Quadrate anhand eines einfachen Beispiels demonstriert, sowie veranschaulicht werden, inwieweit sich Übereinstimmungen zur Maximum Likelihood Schätzung ergeben.

7.4.1 Methode der kleinsten Quadrate für zwei Schätzwerte

Zunächst betrachten wir die Methode der kleinsten Quadrate für zwei Parameter. Für dieses Beispiel gehe ich nach [23] vor.

Angenommen, es existiert ein linearer Zusammenhang zwischen Z und $E(X|z)$, also

$$E(X|z) = \alpha + \beta z$$

Es sollen die zwei Koeffizienten α und β so geschätzt werden, dass die Summe der (vertikalen) quadratischen Abweichungen minimal ist. Die Parameter müssen also so gewählt werden, dass die Funktion Q ihr Minimum annimmt.

$$Q = \sum_{i=1}^{n} [x_i - E(X|z_i)]^2$$

Für die beobachteten Werte (x_i, z_i) wird also die lineare Beziehung angenommen.

$$X_i = \alpha + \beta Z_i$$

Die Funktion Q ist somit gegeben als

$$Q = \sum_{i=1}^{n} (x_i - \alpha - \beta z_i)^2$$

Um die Parameter α und β zu erhalten, soll das Minimum von Q gefunden werden. Dazu werden die partiellen Ableitungen berechnet und Null gesetzt. Wir erhalten

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} Q = -2 \sum_{i=1}^{n} (x_i - \alpha - \beta z_i) = 0$$
$$\frac{\mathrm{d}}{\mathrm{d}\beta} Q = -2 \sum_{i=1}^{n} z_i (x_i - \alpha - \beta z_i) = 0$$

Betrachten wir die erste Gleichung, so erhalten wir den Schätzer für den Parameter α.

$$\begin{aligned}
-2 \sum_{i=1}^{n} (x_i - \alpha - \beta z_i) &= 0 \\
\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \alpha - \sum_{i=1}^{n} \beta z_i &= 0 \\
\sum_{i=1}^{n} \alpha &= \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \beta z_i \\
n\alpha &= \sum_{i=1}^{n} x_i - \beta \sum_{i=1}^{n} z_i \\
\alpha &= \frac{1}{n} \sum_{i=1}^{n} x_i - \beta \frac{1}{n} \sum_{i=1}^{n} z_i
\end{aligned}$$

In der Gleichung sollen die Mittelwerte durch die gängige Schreibweise ersetzt werden.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i$$

Wir erhalten somit den Schätzwert $\hat{\alpha}$.

$$\hat{\alpha} = \bar{x} - \beta\bar{z}$$

Betrachten wir die zweite Gleichung, können wir den Schätzer für β ermitteln.

$$\begin{aligned}
-2\sum_{i=1}^{n} z_i(x_i - \alpha - \beta z_i) &= 0 \\
-2\sum_{i=1}^{n} z_i[x_i - (\bar{x} - \beta\bar{z}) - \beta z_i] &= 0 \\
\sum_{i=1}^{n} z_i(x_i - \bar{x} + \beta\bar{z} - \beta z_i) &= 0 \\
\sum_{i=1}^{n} z_i[(x_i - \bar{x}) + \beta(\bar{z} - z_i)] &= 0 \\
\sum_{i=1}^{n} z_i(x_i - \bar{x}) + \sum_{i=1}^{n} \beta z_i(\bar{z} - z_i) &= 0 \\
-\beta\sum_{i=1}^{n} z_i(\bar{z} - z_i) &= \sum_{i=1}^{n} z_i(x_i - \bar{x}) \\
\beta\sum_{i=1}^{n} z_i(z_i - \bar{z}) &= \sum_{i=1}^{n} z_i(x_i - \bar{x})
\end{aligned}$$

Als Ergebnis erhalten wir

$$\hat{\beta} = \frac{\sum_{i=1}^{n} z_i(x_i - \bar{x})}{\sum_{i=1}^{n} z_i(z_i - \bar{z})}$$

Somit kennen wir für diesen Fall die Schätzwerte für die zwei unbekannten Parameter, die sich bei Anwendung der Methode der kleinsten Quadrate ergeben. Nun können wir zum Vergleich den normalverteilten Fall mithilfe des Maximum Likelihood Prinzips betrachten. Dadurch soll der Zusammenhang zwischen diesen beiden Methoden veranschaulicht werden.

7.4.2 Maximum Likelihood Schätzung für eine bivariate Normalverteilung

Im Folgenden soll das Maximum Likelihood Prinzip in Anlehnung an [23] auf die Normalverteilung angewendet werden.

Es soll eine bivariate Normalverteilung betrachtet werden, wobei die Verteilung von X als

Funktion fixierter Werte Z angenommen wird mit dem linearen Zusammenhang $x = \alpha + \beta z$.

Somit ist die bedingte Dichtefunktion gegeben durch

$$f(x|Z) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} e^{-\frac{1}{2\sigma^2}(x-\alpha-\beta z)^2}$$

Seien (x_i, z_i) die tatsächlichen Beobachtungswerte mit dem Stichprobenumfang $0 < i \leq n$. Dann kann die Likelihood Funktion gegeben werden durch

$$\begin{aligned} L(\alpha, \beta|x_i) = f(x_1, x_2, \ldots, x_n) &= \prod_{i=1}^{n} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} e^{-\frac{1}{2\sigma^2}(x_i-\alpha-\beta z_i)^2} \\ &= \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\alpha-\beta z_i)^2} \end{aligned}$$

Die Likelihood-Funktion soll zur Vereinfachung logarithmiert werden. Wir erhalten

$$\ln L = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}[x_i - \alpha - \beta z_i]^2 \tag{7.4.1}$$

Um das Maximum zu finden, werden von Gleichung (7.4.1) die partiellen Ableitungen nach α und β gebildet und gleich Null gesetzt. Wir erhalten

$$\begin{aligned} \frac{\partial \ln L}{\partial \alpha} &= -2\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \alpha - \beta z_i) = 0 \\ \frac{\partial \ln L}{\partial \beta} &= -2\frac{1}{2\sigma^2}\sum_{i=1}^{n} z_i(x_i - \alpha - \beta z_i) = 0 \end{aligned}$$

Oder vereinfacht ausgedrückt

$$\begin{aligned} \sum_{i=1}^{n}(x_i - \alpha - \beta z_i) &= 0 \\ \sum_{i=1}^{n} z_i(x_i - \alpha - \beta z_i) &= 0 \end{aligned}$$

Als Ergebnisse erhalten wir die Schätzwerte $\hat{\alpha}$ und $\hat{\beta}$.

$$\begin{aligned} \hat{\alpha} &= \overline{X} - \beta\overline{Z} \\ \hat{\beta} &= \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Z_i - \overline{Z})}{\sum_{i=1}^{n}(Z_i - \overline{Z})^2} \end{aligned}$$

Nun soll geprüft werden, ob es sich bei den Schätzwerten tatsächlich um das Maximum handelt. Dazu betrachten wir die Likelihood Funktion an ihren Rändern, also $\lim_{\alpha\to\pm\infty} L$ beziehungsweise $\lim_{\beta\to\pm\infty} L$, wobei der jeweils andere Parameter als fest angenommen wird. Beide Parameter stehen lediglich im Exponenten und werden quadriert. Der Ausdruck geht daher gegen plus unendlich,

der Exponent insgesamt gegen minus unendlich und die Exponentialfunktion gegen Null. Daher gilt

$$\lim_{\alpha\to\infty} L = \lim_{\alpha\to\infty} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\alpha-\beta z_i)^2} = 0$$

Alle anderen Fälle sind analog. Da die Likelihood Funktion das Produkt von Verteilungsdichten ist, ist sie jedenfalls größer Null und wir können $\hat{\alpha}$ und $\hat{\beta}$ als Maximum Likelihood Schätzer akzeptieren.

Wie dieses vereinfachte Beispiel veranschaulicht entsprechen die geschätzten Parameter der Maximum Likelihood Schätzung im Fall einer Normalverteilung den Schätzern, die über die Methode der kleinsten Quadrate ermittelt wurden.

Wenngleich beide Methoden in der Zuverlässigkeitstechnik Anwendung finden, so konnte doch festgestellt werden, dass in den meisten untersuchten Texten das Maximum Likelihood Prinzip zur Schätzung der Parameter herangezogen wurde.

8 Anwendung des Maximum Likelihood Prinzips

In diesem Kapitel soll das Maximum Likelihood Prinzip auf einige der Verteilungsfunktionen aus Kapitel 6, sowie das Jelinski-Moranda Modell angewendet werden. Im Fall des NHPP wird dabei unterschieden, in welcher Form die Ausfalldaten vorliegen, also ob die Zeit bis zum Ausfall erfasst wird, oder die Anzahl an Ausfällen in Intervallen.

8.1 Maximum Likelihood Schätzung für Exponentialverteilungen

Zunächst soll das Maximum Likelihood Prinzip auf die Exponentialverteilung angewendet werden, um einen Schätzer für den Parameter λ zu ermitteln. Dazu gehe ich nach [23] vor.

Sei $X_1, X_2, \ldots, X_n$ eine Zufallsstichprobe mit einer exponentiellen Verteilungsdichte.

$$f(x, \lambda) = \lambda e^{-\lambda x}, \quad x > 0,\, \lambda > 0$$

Die Likelihood-Funktion ist gegeben durch das Produkt der Dichtefunktionen je Beobachtungswert $X_1, X_2, \ldots, X_n$.

$$L(\lambda|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$

Also

$$L(\lambda|x_1, x_2, \ldots, x_n) = \lambda^n e^{-\lambda \sum_{i=1}^{n} x_i}$$

Die logarithmierte Likelihood-Funktion ist sodann

$$\log L(\lambda|X) = n \log \lambda - \lambda \sum_{i=1}^{n} x_i$$

Um das Maximum zu ermitteln, wird die Gleichung nach dem zu schätzenden Parameter λ differenziert und gleich Null gesetzt.

$$(\log L)' = \frac{n}{\lambda} - \sum_{i=1}^{n} x_i = 0$$

Durch Umformung erhalten wir den Schätzer $\hat{\lambda}$.

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^{n} x_i}$$

Um zu prüfen, ob es sich um ein Maximum handelt, betrachten wir die Ränder der Funktion, also $\lim_{\lambda \to \infty} L$, sowie $\lim_{\lambda \to 0^+} L$, da $\lambda > 0$. Zur Vereinfachung wird für die Summe der Wert s angenommen, wobei $s > 0$ gilt. Da die Exponentialfunktion wesentlich stärker wächst als die Potenzfunktion, also $e^{s\lambda} > \lambda^n$ für genügend große λ gilt, erhalten wir

$$\lim_{\lambda \to \infty} L = \lim_{\lambda \to \infty} \lambda^n e^{-\lambda s} = \lim_{\lambda \to \infty} \frac{\lambda^n}{e^{s\lambda}} = 0$$

Außerdem gilt

$$\lim_{\lambda \to 0^+} L = \lim_{\lambda \to \infty} \lambda^n e^{-\lambda s} = 0^n e^{-0s} = 0 \cdot 1 = 0$$

Da jede Dichtefunktion größer Null ist, ist auch die Likelihood Funktion größer Null. Somit ist $\hat{\lambda}$ der Maximum Likelihood Schätzer von λ.

8.2 Maximum Likelihood Schätzung für das J-M Modell

Nun soll das Maximum Likelihood Prinzip auf das Jelinski-Moranda Modell angewendet werden, wie wir es in Kapitel 4.3 kennengelernt haben. Dabei soll nun nicht nur wie im vorherigen Kapitel der Parameter λ geschätzt werden, sondern die Gesamtfehleranzahl N, sowie die Proportionalitätskonstante Φ.

Im Folgenden gehe ich nach [23] vor, unter der Verwendung folgender Notation.

Φ	Proportionalitätskonstante
N	Anzahl an initial vorhandenen Fehlern
t_i	Zeit zwischen dem $(i-1)$-ten und i-ten Ausfall

Analog zu Kapitel 4.3 ist die Ausfallrate gegeben durch

$$\lambda(t_i) = \Phi[N - (i-1)], \quad i = 1, 2, \ldots, N$$

Weiters ist die Dichtefunktion gegeben durch

$$f(t_i) = \lambda(t_i)e^{-\lambda(t_i)t_i}$$

Oder bei Verwendung der zu schätzenden Parameter N und Φ

$$f(t_i) = \Phi[N-(i-1)]e^{-\Phi[N-(i-1)]t_i}$$

Es seien $t_1, t_2, \ldots, t_n$ die beobachteten Ausfallzeiten. Die Maximum Likelihood Funktion für die Parameter N und Φ ist dann gegeben durch

$$\begin{aligned} L(N,\Phi|t_i) &= \prod_{i=1}^{n} f(t_i) \\ &= \prod_{i=1}^{n} [\Phi(N-(i-1))e^{-\Phi(N-(i-1))t_i}] \\ &= \Phi^n \prod_{i=1}^{n} [N-(i-1)]e^{-\Phi\sum_{i=1}^{n}[N-(i-1)]t_i} \end{aligned}$$

Die logarithmierte Likelihood-Funktion ist somit

$$\ln L(N,\Phi|t_i) = n\ln\Phi + \sum_{i=1}^{n}\ln[N-(i-1)] - \Phi\sum_{i=1}^{n}[N-(i-1)]t_i \tag{8.2.1}$$

Um die Gesamtfehleranzahl N sowie die Proportionalitätskonstante Φ zu schätzen, soll die logarithmierte Likelihood-Funktion (8.2.1) nach diesen Parametern partiell differenziert und Null gesetzt werden.

Zunächst berechnet man die erste partielle Ableitung dieser Funktion nach N. So erhält man

$$\frac{\partial}{\partial N}\ln L(N,\Phi|t_i) = \sum_{i=1}^{n}\frac{1}{N-(i-1)} - \Phi\sum_{i=1}^{n} t_i$$

Um das Maximum zu finden, setzt man diesen Ausdruck gleich Null.

$$\frac{\partial}{\partial N}\ln L(N,\Phi|t_i) = 0$$

Daraus ergibt sich

$$\sum_{i=1}^{n}\frac{1}{N-(i-1)} = \Phi\sum_{i=1}^{n} t_i \tag{8.2.2}$$

Nun soll die partielle Ableitung nach Φ der Gleichung (8.2.1) berechnet werden. Man erhält

$$\frac{\partial}{\partial \Phi}\ln L(N,\Phi|t_i) = \frac{n}{\Phi} - \sum_{i=1}^{n}[N-(i-1)]t_i$$

Auch diese Gleichung wird Null gesetzt, also erhalten wir

$$\frac{n}{\Phi} = \sum_{i=1}^{n}[N-(i-1)]t_i \tag{8.2.3}$$

Durch das Lösen der Gleichungen (8.2.2) und (8.2.3) erhalten wir

$$\Phi = \frac{\sum_{i=1}^{n} \frac{1}{N-(i-1)}}{\sum_{i=1}^{n} t_i}$$

Sowie

$$n\sum_{i=1}^{n} t_i = \left[\sum_{i=1}^{n}[N-(i-1)]t_i\right]\left[\sum_{i=1}^{n}\frac{1}{N-(i-1)}\right]$$

Die Größe N kann durch numerische Lösungsverfahren, etwa mithilfe des Newton-Verfahrens oder dem Regula-Falsi-Verfahren, bestimmt werden. [4]

Nun soll die Likelihood Funktion an ihren Rändern betrachtet werden, wobei $N > 0$, $\Phi > 0$ und $n \leq N$ gilt.

$$\begin{aligned}\lim_{N\to\infty} L &= \lim_{N\to\infty} \Phi^n \prod_{i=1}^{n}[N-(i-1)]e^{-\Phi\sum_{i=1}^{n}[N-(i-1)]t_i} \\ &= \lim_{N\to\infty} \Phi^n \frac{\prod_{i=1}^{n} N-(i-1)}{e^{\Phi\sum_{i=1}^{n}[N-(i-1)]t_i}} = 0\end{aligned}$$

da der Ausdruck im Zähler kleiner N^n, einer Potenzfunktion, ist, und deren Wachstum deutlich geringer als das der Exponentialfunktion ausfällt.

Für $\Phi \to \infty$ kann in ähnlicher Weise vorgegangen und argumentiert werden. Auch hier geht der Grenzwert Likelihood Funktion gegen Null.

Betrachten wir nun das Verhalten für $N \to 0^+$, sowie $\phi \to 0^+$. Es ist leicht ersichtlich, dass auch hier die Grenzwerte gleich Null sind. Gleichzeitig gilt $L > 0$ für den gesamten Definitionsbereich. Somit sind $\hat{\Phi}$ und $\hat{N}$ Maximum Likelihood Schätzer.

8.3 Maximum Likelihood Schätzung für NHPP

Wie in Kapitel 6.4 angemerkt wurde, können NHPP Modelle sowohl für diskrete als auch kontinuierliche Prozesse verwendet werden. Je nach Art der Datenerfassung werden laut [23] typischerweise folgende zwei Herangehensweisen genutzt, um die Parameter mithilfe des Maximum Likelihood Prinzips zu schätzen. Für die entsprechenden Beschreibungen in den Kapiteln 8.3.1 und 8.3.2 gehe ich somit nach [23] vor.

8.3.1 Maximum Likelihood Schätzung im Fall von Zeitbereichsdaten

Für Modelle, die auf Zeitbereichsdaten basieren, werden die Ausfallzeitpunkte beobachtet. Entsprechend kann die Schreibweise aus Kapitel 6.4 für die Dichtefunktion von NHPP-Modelle

verwendet werden, also

$$f(x) = \lambda(t+x)e^{-[m(t+x)-m(t)]}$$

Sei nun S_j mit $j = 1, 2, \ldots, n$ die Zufallsvariable der beobachtbaren aufeinanderfolgenden Zeitpunkte, zu denen der j-te Ausfall eintritt und s_j mit $0 \leq s_1 \leq s_2 \cdots \leq s_n$ die Realisierung dieser, also die tatsächlich beobachteten Zeitpunkte, sowie n die beobachtete Anzahl an Ausfällen. So kann die Dichtefunktion mit dem gesuchten Parameter auch folgendermaßen angegeben werden

$$f(s_i|\Theta) = \lambda(s_i)e^{-[m(s_i)-m(s_{i-1})]}$$

Die Likelihood-Funktion kann also gegeben werden durch

$$\begin{aligned} L(\Theta|s_i) &= \prod_{i=1}^{n} \left[\lambda(s_i)e^{-[m(s_i)-m(s_{i-1})]}\right] \\ &= \prod_{i=1}^{n} \lambda(s_i) \cdot \prod_{i=1}^{n} e^{-[m(s_i)-m(s_{i-1})]} \\ &= \prod_{i=1}^{n} [\lambda(s_i)] \cdot e^{-\sum_{i=1}^{n}[m(s_i)-m(s_{i-1})]} \\ &= \prod_{i=1}^{n} [\lambda(s_i)] \cdot e^{-[m(s_n)-m(s_0)]} \end{aligned}$$

Per Definition muss $m(s_0)$ gleich Null sein, also

$$L(\Theta|s_i) = \prod_{i=1}^{n} [\lambda(s_i)] \cdot e^{-m(s_n)}$$

Die logarithmierte Likelihood-Funktion ist somit

$$\log L(s_i|\Theta) = \sum_{i=1}^{n} \log[\lambda(s_i)] - m(s_n)$$

Um die unbekannten Parameter $\Theta = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ zu schätzen, wird die Gleichung nach den einzelnen Parametern partiell differenziert und gleich Null gesetzt. Dabei gilt weiterhin die Beziehung

$$\lambda(t) = m'(t)$$

Daher können die Schätzer des unbekannten Paramters $\Theta = (\Theta_1, \Theta_2, \ldots, \Theta_n)$ durch das Lösen folgenden Gleichungssystems ermittelt werden

$$0 = \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta}\lambda(S_i)}{\lambda(S_i)} - \frac{\mathrm{d}}{\mathrm{d}\Theta}m(S_n) \tag{8.3.1}$$

Für Θ ist jeder der unbekannten Parameter einzusetzen.

Erneut können Überlegungen bezüglich der Randwerte der Likelihood Funktion angestellt werden. Wir betrachten dazu $\lambda > 0$. Da die Beziehung $m = \int \lambda \mathrm{d}s$ gilt, können wir das Wachstumsverhalten der Exponentialfunktion in Vergleich zum Produkt setzen. Es gilt also

$$\lim_{\lambda \to \infty} L = \lim_{\lambda \to \infty} \prod_{i=1}^{n} [\lambda(s_i)] \cdot e^{-m(s_n)} = 0$$

Auch für $\lambda \to 0^+$ geht die Likelihood Funktion gegen Null. Die Schätzer, die über das Gleichungssystem (8.3.1) gefunden werden können, sind somit ein globales Maximum.

Die zu lösenden Gleichungen für die Schätzwerte sind nichtlinear. Erneut können numerische Näherungsverfahren angewendet werden. Da es etwa für das Newton-Verfahren notwendig sein kann, die ersten und zweiten Ableitungen der Funktionen $m(t)$ und $\lambda(t)$ zu ermitteln, ist es nach [23] empfehlenswert, geeignete Computerprogramme zur Berechnung in Betracht zu ziehen.

8.3.2 Maximum Likelihood Schätzung im Fall von Intervallbereichsdaten

Im Fall von Modellen für Intervallbereichsdaten wird die Anzahl an beobachteten Ausfällen je Intervall beobachtet und erfasst.

Dabei sei y_i die kumulierte Anzahl an beobachteten Ausfällen in einem gegebenen Zeitintervall $(0, t_i)$ für $i = 1, 2, \ldots, n$ und $0 < t_1 < t_2 < \cdots < t_n$. Die Größe $y_i := y(t_i)$ ist also die Realisierung von $N(t)$ zu den Zeitpunkten t_i. Somit betrachten wir die Wahrscheinlichkeit $P\{N(t_i) = y_i\}$.

Gemäß Kapitel 6.4 ist die Wahrscheinlichkeit für n Ausfälle im Intervall $(0, t)$ von folgender Form

$$P\{N(t) = n\} = \frac{[m(t)]^n}{n!} e^{-m(t)}$$

Wir sind nun an der Wahrscheinlichkeit $P\{N(t_i) = y_i\}$ im Intervall (t_{i-1}, t_i) interessiert. Diese ist gegeben durch

$$P\{N(t_i) = y_i\} = \frac{[m(t_i) - m(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-[m(t_i) - m(t_{i-1})]}$$

Die Likelihood Funktion ist demnach

$$L(\Theta | (t_i, y_i)) = \prod_{i=1}^{n} \frac{[m(t_i) - m(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-[m(t_i) - m(t_{i-1})]}$$

Wie in den Kapiteln zuvor soll die Likelihood Funktion logarithmiert werden. Sie nimmt folgende Form an

$$\begin{aligned}\log L(\Theta|(t_i, y_i)) &= \sum_{i=1}^{n}(y_i - y_{i-1})\log[m(t_i) - m(t_{i-1})] - \log[(y_i - y_{i-1})!] - [m(t_i) - m(t_{i-1})] \\ &= \sum_{i=1}^{n}(y_i - y_{i-1})\log[m(t_i) - m(t_{i-1})] - \log[(y_i - y_{i-1})!] - m(t_n)\end{aligned}$$

Durch das partielle Ableiten nach Θ, sowie das Nullsetzen, können wir das Maximum dieser Funktion über folgendes Gleichungssystem bestimmen

$$0 = \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta}m(t_i) - \frac{\mathrm{d}}{\mathrm{d}\Theta}m(t_{i-1})}{m(t_i) - m(t_{i-1})}(y_i - y_{i-1}) - \frac{\mathrm{d}}{\mathrm{d}\Theta}m(t_n) \tag{8.3.2}$$

Für Θ ist jeder der unbekannten Parameter einzusetzen.

Mithilfe der beobachteten Fehlerdaten (t_1, y_1) mit $i = 1, 2, \ldots, n$ und der Mittelwertfunktion $m(t_i)$ kann dann in weiterer Folge die erwartete Anzahl an Ausfällen für die Zeitpunkte t_i für $i = n + 1, n + 2, \ldots$ ermittelt werden. [23]

Betrachten wir nun noch die Randbereiche der Likelihood Funktion für $m \to \infty$ und $m \to 0^+$. Da m die erwartete Anzahl an bis zur Zeit t beobachteten Ausfällen beschreibt, können wir davon ausgehen, dass $m(t_i) \geq m(t_{i-1})$ gilt. Ähnlich können wir davon ausgehen, dass $y_i - y_{i-1}$ einen positiven (festen) Wert annimmt. Außerdem ist erneut das Wachstum der Exponentialfunktion wesentlich stärker als jenes der Potenzfunktion. Somit erhalten wir

$$\lim_{m\to\infty} L = \lim_{m\to\infty} \prod_{i=1}^{n} \frac{[m(t_i) - m(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-[m(t_i) - m(t_{i-1})]} = 0$$

Für $m \to 0$ ist leicht zu sehen, dass die Likelihood Funktion auch an diesem Rand gegen Null geht, wobei die Funktion selbst für den gesamten Definitionsbereich größer Null ist. Daher handelt es sich bei dem Schätzer erneut um das global Maximum.

8.4 Maximum Likelikhood Schätzung für Weibull-Verteilungen

Wie in Kapitel 6.2 ist die Dichtefunktion der dreiparametrigen Weibull-Verteilung gegeben durch

$$f(t) = \frac{\beta(t - \gamma)^{\beta - 1}}{\theta^{\beta}} e^{-(\frac{t-\gamma}{\theta})^{\beta}}, \quad t \geq \gamma \geq 0$$

Sei nun t_i mit $i = 1, 2, \ldots, n$ und $0 < t_1 \leq t_2 \leq \cdots \leq n$ der beobachtete Zeitpunkt des j-ten Ausfalls.

Dann ist die Likelihood Funktion folgendermaßen gegeben

$$\begin{aligned} L(\beta,\gamma,\theta|t_i) &= \prod_{i=1}^{n} f(t_i|\beta,\gamma,\theta) \\ &= \prod_{i=1}^{n} \frac{\beta(t_i-\gamma)^{\beta-1}}{\theta^{\beta}} e^{-(\frac{t_i-\gamma}{\theta})^{\beta}} \\ &= \frac{\beta^n}{\theta^{n\beta}} \prod_{i=1}^{n} (t_i-\gamma)^{\beta-1} \prod_{i=1}^{n} e^{-(\frac{t_i-\gamma}{\theta})^{\beta}} \\ &= \frac{\beta^n}{\theta^{n\beta}} e^{-\sum_{i=1}^{n}(\frac{t_i-\gamma}{\theta})^{\beta}} \prod_{i=1}^{n} (t_i-\gamma)^{\beta-1} \end{aligned}$$

Die logarithmierte Likelihood Funktion kann dann wie folgt dargestellt werden.

$$\log L(\beta,\gamma,\theta|t_i) = n\log\beta - n\beta\log\theta - \sum_{i=1}^{n}\left(\frac{t_i-\gamma}{\theta}\right)^{\beta} + (\beta-1)\sum_{i=1}^{n}\log(t_i-\gamma)$$

Das Maximum erhalten wir durch Nullsetzen der partiellen Ableitungen nach den gesuchten Parametern β, γ und θ.

$$\begin{aligned} \frac{\mathrm{d}\log L}{\mathrm{d}\beta} &= \frac{n}{\beta} - n\log\theta - \sum_{i=1}^{n}\left(\frac{t_i-\gamma}{\theta}\right)^{\beta}\log\left(\frac{t_i-\gamma}{\theta}\right) + \sum_{i=1}^{n}\log(t_i-\gamma) = 0 \\ \frac{\mathrm{d}\log L}{\mathrm{d}\theta} &= -\frac{n\beta}{\theta} + \beta\theta^{-\beta-1}\sum_{i=1}^{n}(t_i-\gamma)^{\beta} = 0 \\ \frac{\mathrm{d}\log L}{\mathrm{d}\gamma} &= \theta^{-\beta}\sum_{i=1}^{n}(t_i-\gamma)^{\beta-1} - (\beta-1)\sum_{i=1}^{n}\frac{1}{t_i-\gamma} = 0 \end{aligned}$$

Das Lösen des Gleichungssystems nach β, γ und θ ist komplex und bedarf nach [23] entweder einer grafischen oder numerischen Lösungsmethode.

Über Betrachtung des Randes können wir feststellen, dass der Schätzer wirklich ein globales Maximum ist.

9 Verschiedene Modelle und Überlegungen bezüglich ihrer Anwendung

Wie in Kapitel 5.1 festgehalten, werden im Bereich der Software-Zuverlässigkeits-Analyse geeignete Wahrscheinlichkeitsmodelle aufgrund von Annahmen abgeleitet und gewählt.

Der erste Schritt im Vorgehen zur Ermittlung der Zuverlässigkeit ist somit konzeptioneller Natur und beschäftigt sich mit der Analyse der Eigenschaften des Objekts und der Art der Ausfälle – nicht zuletzt mit der Frage, was ein Ausfall ist beziehungsweise als Ausfall gezählt wird, aber auch mit einer Einschätzung über die Veränderung der Ausfallhäufigkeit mit der Zeit. [12]

Erst in einem zweiten Schritt werden dann mithilfe von Testdaten die Wahrscheinlichkeiten beziehungsweise Parameter des Modells geschätzt und getestet. [12]

Im Folgenden soll ein kurzer Überblick über verschiedene Einteilungen von Modellen gegeben werden, bevor im Anschluss eine kleine Auswahl an Modellen vorgestellt wird. Schließlich sollen noch weitere Aspekte bezüglich ihrer Anwendung betrachtet werden.

9.1 Überblick über unterschiedliche Software-Zuverlässigkeits Modelle

In Kapitel 2.2.3 konnten von einem sehr allgemeinen Standpunkt aus verschiedene Herangehensweisen in Bezug auf Qualitätssicherung und mögliche Unterteilungen der Verfahren vorgestellt werden. Im weiteren Verlauf konnte in Kapitel 5 auf die wichtigsten Unterschiede zwischen Hardware- und Software-Zuverlässigkeit, sowie reparierbare und nicht-reparierbare Systeme eingegangen werden, mit daraus folgenden Implikationen für die Wahl und Konzeption von Zuverlässigkeitsmodellen.

In diesem Kapitel soll kurz darauf eingegangen werden, inwieweit detailliertere Einteilungen der Modelle vorgenommen werden können.

So unterteilt [23] die probabilistischen Modelle in folgende sechs Untergruppen:

- Error Seeding Modelle

  Das Grundprinzip bei Error Seeding Modellen besteht darin, dass in den Code bewusst Fehler induziert werden. Die unbekannte Anzahl an inhärenten Fehlern wird über das Verhältnis dieser zur bekannten Anzahl induzierter Fehler ermittelt. Dabei wird das Verhältnis der beiden Fehlerkategorien durch debugging-Daten erhalten. Es werden mehrstufige Stichprobenverfahren angewendet. [23]

- Ausfallraten Modelle

  Modelle bezüglich der Ausfallrate untersuchen die Änderungen der Ausfallrate zu den Ausfallzeitpunkten in Ausfallintervallen. Bei diesen Modellen wird davon ausgegangen, dass sich die Ausfallrate gemäß der Anzahl an verbleibenden Fehlern im Programm ändert. Das Jelinski-Moranda Modell fällt in diese Kategorie von Modellen. [23]

- Kurvenanpassungsmodelle

  Kurvenanpassungsmodelle verwenden die statistische Regressionsanalyse, um den Zusammenhang zwischen Software-Komplexität, der Anzahl an Änderungen, der Anzahl an Ausfällen oder der Ausfallrate zu untersuchen. Dabei werden lineare Regression, nicht-lineare Regression oder Zeitreihenanalyse verwendet, um funktionale Beziehungen zwischen abhängigen Variablen (etwa der Anzahl an Ausfällen) und unabhängigen Variablen (beispielsweise der Anzahl an Änderungen, der Programmgröße oder den Fähigkeiten der Programmierer) zu finden. [23]

- Zuverlässigkeitswachstumsmodelle (Software Reliability Growth Models, SRGM)

  Diese Modelle gehen davon aus, dass sich die Zuverlässigkeit mit der Zeit verbessert. Das ist darauf zurück zu führen, dass angenommen wird, dass während des Softwaretests (und der

Betriebsphase) Fehler gefunden und behoben werden. SRGM messen und prognostizieren also die Zuverlässigkeit während der Testphase. Die Ausfallrate oder Zuverlässigkeit werden als Funktion der Zeit oder der Anzahl an Testfällen modelliert. [23]

- NHPP Modelle

  Auch bei NHPP Modellen werden Software-Ausfälle in der Testphase beschrieben. Dabei liegt das Hauptaugenmerk auf der Schätzung der Mittelwertsfunktion der kumulativen Anzahl an beobachten Ausfällen bis zu einem bestimmten Zeitpunkt. In Kapitel 6.4 wurden bereits die wichtigsten Eigenschaften dieser Verteilung vorgestellt. [23]

- Markov Modelle

  Der Grundgedanke bei Markov Prozessen ist, dass das zukünftige Verhalten nur vom jetzigen Zustand, nicht aber von der Vergangenheit abhängt. Bei diese Gruppe handelt es sich um einen generellen Ansatz zur Beschreibung von Fehlerprozessen. [23]

Demgegenüber unterscheidet [15] zwischen folgenden drei Kategorien an Modellen:

- Bayes Modelle

  In der klassischen Statistik sind die gesuchten Parameter unbekannt. Es wird also davon ausgegangen, dass es sich um Konstanten handelt. In Bayes Modellen wird der Parameter selbst auch als Zufallsvariable verstanden. [23]

  Die Grundannahme ist, dass das Eintreten eines Ereignisses kontextabhängig und die Wahrscheinlichkeit dafür somit immer eine bedingte Wahrscheinlichkeit ist. Diese ist dann als Maß für das Wissen im Vergleich zum Nicht-Wissen zu verstehen. [12]

  Auf die Modellierung bezogen ist die Idee, das a-priori Wissen zu verwenden und mithilfe aktueller Daten die Schätzung zu verbessern. [23]

- Markov Modelle

- NHPP Modelle

Abgesehen von den beiden vorgestellten Kategorisierungen von Software-Zuverlässigkeits-Modellen gibt es verschiedene Möglichkeiten Einteilungen vorzunehmen. Es ist allerdings nicht Ziel dieser Arbeit, hier einen umfassenden Vergleich vorzunehmen. Vielmehr soll über die zwei vorgestellten ein Eindruck über verschiedene Ansätze und relevante Kategorien vermittelt werden. So soll an dieser Stelle festgehalten werden, dass zumindest eine Überschneidung von zwei Kategorien vorhanden ist. Gleichzeitig soll darauf hingewiesen werden, dass in [15] alle Modelle, die in der Testphase verwendet werden, als SRGM bezeichnet werden, während [6] dynamische Modelle in Zusammenhang mit SRGM setzt. Weiters werden NHPP in [24] als große Gruppe der SRGM bezeichnet. Mehr noch, die Autorin von [3] geht davon aus, dass NHPP Modelle aufgrund ihrer realistischeren Annahmen wohl die gebräuchlichsten Software-Zuverlässigkeits-Modelle sind. Insgesamt konnte beobachtet werden, dass sowohl SRGM also auch NHPP entscheidende Begrifflichkeiten in Zusammenhang mit Software-Zuverlässigkeit darstellen. Auch in

Kapitel 2.2.3 konnte bereits festgehalten werden, dass nach [9] die am meisten untersuchte sowie verwendete Gruppe jene der SRGM ist.

9.2 NHPP Modelle im Allgemeinen

Wie soeben besprochen ist eine bedeutende Kategorie an Modellen jene der NHPP. In Kapitel 6.4 wurden die Verteilungsfunktion sowie Überlegungen bezüglich der Anwendbarkeit vorgestellt. Ebenso wurde festgehalten, dass sich Modelle dieser Gruppe bezüglich ihrer Mittelwertfunktion unterscheiden. Die Form und Realisierung der Mittelwertfunktion wird aufgrund verschiedener Annahmen über den Test- und Entwicklungsprozess bestimmt.

Nach [6] basieren die meisten Modellen auf folgenden allgemeinen Annahmen.

- Das zu testende System bleibt abgesehen von der Entfernung gefundener Fehler während des Testens unverändert. Lediglich manche Modelle berücksichtigen, dass durch die Korrektur von Fehlern neue Fehler entstehen können.
- Alle Fehler sind bezüglich ihres Auftretens unabhängig voneinander. Das heißt auch, dass sich die Wahrscheinlichkeit einen Fehler zu finden nicht durch die Korrektur eines anderen Fehlers verändert.
- Der Testaufwand ist bezüglich der gemessenen Zeit konstant beziehungsweise annähernd gleichförmig.
- Die zukünftige Entwicklung des Testprozesses beziehungsweise die Beobachtung von Ausfällen hängt vom aktuellen Zustand des Systems ab, also dem aktuellen Zeitpunkt, der Anzahl gefundener und verbleibender Fehler und den allgemeinen Parametern des Modells. Sie hängt allerdings nicht vom vergangen Testprozess ab und ist somit Markovian.
- Alle Fehler sind gleich wichtig und werden in der Ausfallrate gleichermaßen berücksichtigt.
- Zu Beginn des Testens gibt es eine endliche Anzahl an Fehlern im System, die fix oder zufällig ist. Alternativ wird eine unendliche Anzahl an Fehlern angenommen.
- Zwischen Ausfällen folgt die Ausfallintensität einer bekannten Funktionsform. Diese wird oft zur Vereinfachung als konstant angenommen.

Abgesehen von diesen allgemeinen Annahmen werden üblicherweise weitere spezifischere Annahmen über den Entwicklungsprozess und die Testumgebung getroffen. Darauf basierend wird dann ein geeignetes Modell gewählt. Nach [23] ist es daher notwendig, mit den verschiedenen Modellen vertraut zu sein, um fundierte Entscheidungen treffen zu können. Ebenso ist nach [6] jeweils zu prüfen, ob die Modellannahmen tatsächlich erfüllt werden. Einige Problematiken diesbezüglich werden in Kapitel 9.7 erläutert.

Gleichzeitig bemerkt [6], dass unterschiedliche Sets an Annahmen zu äquivalenten Modellen führen können. Demnach gibt es nach [6] wenig Erkenntnisse darüber, wie verschiedene Modelle unterschieden werden können.

9.3 Ein NHPP exponential Modell: Goel-Okumoto NHPP

Eine einfache Realisierung eines NHPP Modells ist das Goel-Okumoto NHPP. Es ähnelt dem J-M Modell, mit dem Unterschied, dass die Ausfallrate abhängig von der Zeit ist und abnimmt. [15] Ansonsten wird auch hier angenommen, dass erkannte Fehler auf jeden Fall korrigiert werden, ohne neue Fehler im Code einzuführen. Diese Annahme wird auch als „perfect debugging“ bezeichnet.

In diesem Kapitel gehe ich nach [23] vor. Es soll folgende Notation verwendet werden.

$m(t)$	erwartete Anzahl an bis zur Zeit t erkannten Fehlern
a	Gesamtanzahl an Fehlern
b	Fehlererkennungsrate
$N(t)$	die Zufallsvariable der kumulativen Anzahl an erkannten Fehlern zur Zeit t
$y(t)$	beobachteter Wert von $N(t)$, wobei $y_i := y(t_i)$
S_j	beobachtete Ausfallzeit des j-ten Fehlers
$R(s\|t)$	Zuverlässigkeit im Intervall $(t, t+s]$, unter der Voraussetzung, dass der letzte Ausfall zur Zeit t stattgefunden hat.

Es werden folgende Annahmen vorausgesetzt

- Alle Fehler im Programm sind aus Sicht der Ausfallerkennung unabhängig voneinander.
- Die Anzahl an gefundenen Ausfällen ist zu jeder Zeit proportional zur aktuellen Anzahl an Fehlern im Programm. Das heißt, dass die Wahrscheinlichkeit dafür, dass ein Fehler tatsächlich einen Ausfall verursacht, konstant ist.
- Erkannte Fehler werden vor dem nächsten Testlauf entfernt.
- Nach jedem Ausfall wird der verursachende Fehler sofort und mit Sicherheit entfernt, ohne dabei neue Fehler einzuführen.

Diese Annahmen werden durch die folgende Differentialgleichung veranschaulicht.

$$m'(t) = b[a - m(t)] \tag{9.3.1}$$

Dabei stellt die Größe a die erwartete initiale Fehleranzahl und b die Ausfallintensität eines einzelnen Fehlers dar. Wie im J-M Modell kann die Ausfallrate $m'(t)$ also als Produkt der konstanten Ausfallintensität eines einzelnen Fehlers und der erwarteten Anzahl an in der Software verbleibenden Fehlern dargestellt werden.

Die Differentialgleichung (9.3.1) ist eine inhomogene lineare Differentialgleichung erster Ordnung und kann über die Methode der Variation der Konstanten gelöst werden.

Durch Umformung der Gleichung (9.3.1) erhalten wir

$$\begin{aligned} m'(t) &= ab - bm(t) \\ m'(t) + bm(t) &= ab \end{aligned}$$

Nun betrachten wir die homogene Differentialgleichung

$$m_0'(t) + bm_0(t) = 0$$

Somit gilt

$$m_0'(t) = -bm_0(t)$$

Die Gleichung kann auch folgendermaßen dargestellt werden

$$\frac{\mathrm{d}m_0}{m_0(t)} = -b\mathrm{d}t$$

Nun soll die Gleichung integriert werden. Also

$$\int \frac{\mathrm{d}m_0}{m_0(t)} = \int -b\mathrm{d}t$$

Wir erhalten

$$\log|m_0| = -bt + \tilde{c}$$

Somit ergibt sich als Lösung der homogenen Differentialgleichung

$$\begin{aligned} m_0 &= e^{-bt+\tilde{c}} \\ &= e^{-bt}e^{\tilde{c}} \end{aligned}$$

Also

$$m_0 = ce^{-bt}$$

Gemäß der Methode der Variation der Konstanten können wir schreiben

$$\begin{aligned} c &\to c(t) \\ m_0 &\to m = c(t)e^{-bt} \end{aligned}$$

Wird $m(t)$ abgeleitet, so erhalten wir gemäß der Produktregel

$$m'(t) = c'(t)e^{-bt} - bc(t)e^{-bt}$$

Nun sollen $m(t)$ und $m'(t)$ in die Differentialgleichung (9.3.1) eingesetzt werden.

$$c'(t)e^{-bt} - bc(t)e^{-bt} = b[a - c(t)e^{-bt}]$$

Nach Auflösen der Klammern erhalten wir

$$c'(t)e^{-bt} - bc(t)e^{-bt} = ab - bc(t)e^{-bt}$$

Beziehungsweise nach dem Kürzen

$$c'(t)e^{-bt} = ab$$

Also erhalten wir

$$c'(t) = abe^{bt}$$

Um $c(t)$ zu finden, wird diese Gleichung integriert

$$\begin{aligned} c(t) &= \int abe^{bt}\mathrm{d}t \\ &= ab\frac{1}{b}e^{bt} + c_1 \\ &= ae^{bt} + c_1 \end{aligned}$$

Setzen wir diesen Ausdruck in die Gleichung von $m(t)$ ein, so erhalten wir

$$\begin{aligned} m(t) &= (ae^{bt} + c_1)e^{-bt} \\ &= ae^{bt}e^{-bt} + ce^{-bt} \end{aligned}$$

Die Mittelwertfunktion hat also folgende Form

$$m(t) = a + ce^{-bt}$$

Es gilt die Bedingung, dass zu Beginn des Beobachtungszeitraums noch keine Fehler gefunden wurden, also

$$m(0) = 0$$

Daher erhalten wir

$$\begin{aligned} 0 &= a + c \\ c &= -a \end{aligned}$$

Somit ergibt sich

$$m(t) = a - ae^{-bt}$$

Das Ergebnis der Differentialgleichung ist also (9.3.1)

$$m(t) = a(1 - e^{-bt}) \tag{9.3.2}$$

In Kapitel 8.3 konnten wir die allgemeinen Formen der zu lösenden Gleichungssysteme ermitteln, um Schätzwerte für a und b mittels der Maximum Likelihood Schätzung zu erhalten. Nun sollen diese verwendet werden, um die Schätzer für die gefundene Mittelwertfunktion in diesem Modell zu erhalten.

9.3.1 Maximum Likelihood Schätzung im Fall von Intervallbereichsdaten

Im Fall von Intervallbereichsdaten wird die kumulative Anzahl an Ausfällen je Zeitintervall beobachtet. Daher ist die soeben ermittelte Form von $m(t)$ in die Gleichung (8.3.2) einzusetzen, sowie sind für Θ die gesuchten Parameter a und b zu verwenden. Das Gleichungssystem hat also folgende Form

$$\begin{aligned} 0 &= \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta} m(t_i) - \frac{\mathrm{d}}{\mathrm{d}\Theta} m(t_{i-1})}{m(t_i) - m(t_{i-1})} (y_i - y_{i-1}) - \frac{\mathrm{d}}{\mathrm{d}\Theta} m(t_n) \\ \frac{\mathrm{d}}{\mathrm{d}\Theta} m(t_n) &= \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta} m(t_i) - \frac{\mathrm{d}}{\mathrm{d}\Theta} m(t_{i-1})}{m(t_i) - m(t_{i-1})} (y_i - y_{i-1}) \end{aligned}$$

Dabei sind y_i mit $i = 1, 2, \ldots, n$ die Realisierungen der kumulativen Anzahl an Ausfällen zu den Zeitpunkten t_i. Setzen wir $m(t)$ aus Gleichung (9.3.2) ein, so erhalten wir

$$\frac{\mathrm{d}}{\mathrm{d}\Theta} a(1 - e^{-bt_n}) = \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta} a(1 - e^{-bt_i}) - \frac{\mathrm{d}}{\mathrm{d}\Theta} a(1 - e^{-bt_{i-1}})}{a(1 - e^{-bt_i}) - a(1 - e^{-bt_{i-1}})} (y_i - y_{i-1}) \tag{9.3.3}$$

Zunächst soll die partielle Ableitung nach a ermittelt werden. Wir erhalten

$$\begin{aligned} 1 - e^{-bt_n} &= \sum_{i=1}^{n} \frac{(1 - e^{-bt_i}) - (1 - e^{-bt_{i-1}})}{a(1 - e^{-bt_i}) - a(1 - e^{-bt_{i-1}})} (y_i - y_{i-1}) \\ &= \sum_{i=1}^{n} \frac{(-e^{-bt_i} + e^{-bt_{i-1}})}{a(-e^{-bt_i} + e^{-bt_{i-1}})} (y_i - y_{i-1}) \\ &= \sum_{i=1}^{n} \frac{1}{a} (y_i - y_{i-1}) \\ &= \frac{1}{a} \sum_{i=1}^{n} (y_i - y_{i-1}) \\ &= \frac{1}{a} (y_n - y_0) \end{aligned}$$

Zum Zeitpunkt $t = 0$ ist die Anzahl an gefundenen Fehlern gleich Null. Daher erhalten wir

$$1 - e^{-bt_n} = \frac{1}{a} y_n$$

Durch Umformung erhalten wir folgende Gleichung für a

$$a = \frac{y_n}{(1 - e^{-bt_n})} \tag{9.3.4}$$

Nun soll für Gleichung (9.3.3) die partielle Ableitung nach b ermittelt werden. Wir erhalten

$$\begin{aligned} at_n e^{-bt_n} &= \sum_{i=1}^{n} \frac{at_i e^{-bt_i} - at_{i-1} e^{-bt_{i-1}}}{a(-e^{-bt_i} + e^{-bt_{i-1}})}(y_i - y_{i-1}) \\ &= \sum_{i=1}^{n} \frac{a(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{a(-e^{-bt_i} + e^{-bt_{i-1}})}(y_i - y_{i-1}) \\ &= \sum_{i=1}^{n} \frac{(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{(-e^{-bt_i} + e^{-bt_{i-1}})}(y_i - y_{i-1}) \end{aligned}$$

Setzen wir für a den Ausdruck aus (9.3.4) ein, so erhalten wir

$$\frac{y_n t_n e^{-bt_n}}{(1 - e^{-bt_n})} = \sum_{i=1}^{n} \frac{(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{(-e^{-bt_i} + e^{-bt_{i-1}})}(y_i - y_{i-1}) \tag{9.3.5}$$

Schätzungen für a und b können somit für Intervallbereichsdaten ermittelt werden, indem die Gleichungen (9.3.4) und (9.3.5) gleichzeitig gelöst werden.

9.3.2 Maximum Likelihood Schätzung im Fall von Zeitbereichsdaten

Im Fall von Zeitbereichsdaten werden die Zeitpunkte der Ausfälle beobachtet. Wie zuvor können wir Erkenntnisse aus Kapitel 8.3, in diesem Fall das Gleichungssystem (8.3.1), verwenden, um Schätzungen für die initiale Gesamtfehleranzahl a und die Fehlererkennungsrate b zu erhalten. Dieses hat folgende Form

$$\begin{aligned} 0 &= \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta}\lambda(s_i)}{\lambda(s_i)} - \frac{\mathrm{d}}{\mathrm{d}\Theta} m(s_n) \\ \frac{\mathrm{d}}{\mathrm{d}\Theta} m(s_n) &= \sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta}\lambda(s_i)}{\lambda(s_i)} \end{aligned}$$

Erneut kann die Form der Mittelwertfunktion $m(t)$ aus Gleichung (9.3.2) eingesetzt werden. Außerdem wird die Ausfallrate $\lambda(t)$, also die erste Ableitung der Mittelwertfunktion nach t verwendet. Diese ist

$$\begin{aligned} m'(t) &= \left[a(1 - e^{-bt})\right]' \\ &= abe^{-bt} \end{aligned}$$

Wir erhalten also folgendes Gleichungssystem

$$\frac{\mathrm{d}}{\mathrm{d}\Theta}a(1-e^{-bs_n}) = \sum_{i=1}^{n}\frac{\frac{\mathrm{d}}{\mathrm{d}\Theta}abe^{-bs_i}}{abe^{-bs_i}} \tag{9.3.6}$$

Dabei sind s_i mit $i = 1, 2, \ldots, n$ die beobachteten aufeinander folgenden Zeitpunkte der Ausfälle. Nun soll diese Gleichung nach a partiell differenziert werden.

$$\begin{aligned} 1-e^{-bs_n} &= \sum_{i=1}^{n}\frac{be^{-bs_i}}{abe^{-bs_i}} \\ &= \sum_{i=1}^{n}\frac{1}{a} \\ &= \frac{n}{a} \end{aligned}$$

Somit erhalten wir durch Umformen

$$a = \frac{n}{(1-e^{-bs_n})} \tag{9.3.7}$$

Auf gleiche Weise soll nun Gleichung (9.3.6) nach b differenziert werden. Also

$$\frac{\mathrm{d}}{\mathrm{d}b}a(1-e^{-bs_n}) = \sum_{i=1}^{n}\frac{\frac{\mathrm{d}}{\mathrm{d}b}abe^{-bs_i}}{abe^{-bs_i}}$$

Der Ausdruck im rechten Term kann mithilfe der Produktregel differenziert werden.

$$\begin{aligned} \left[\frac{\mathrm{d}}{\mathrm{d}b}abe^{-bs_i}\right]' &= ae^{-bs_i} + ab(-s_i)e^{-bs_i} \\ &= ae^{-bs_i}(1-bs_i) \end{aligned}$$

Wir erhalten somit

$$\begin{aligned} as_ne^{-bs_n} &= \sum_{i=1}^{n}\frac{(1-bs_i)ae^{-bs_i}}{abe^{-bs_i}} \\ &= \sum_{i=1}^{n}\frac{(1-bs_i)}{b} \\ &= \sum_{i=1}^{n}\left(\frac{1}{b}-s_i\right) \\ &= \frac{n}{b} - \sum_{i=1}^{n}s_i \end{aligned}$$

Nun soll für a der Ausdruck aus Gleichung (9.3.7) eingesetzt werden.

$$\frac{ns_n e^{-bs_n}}{(1-e^{-bs_n})} = \frac{n}{b} - \sum_{i=1}^{n} s_i$$

Durch Umformung erhalten wir folgende Gleichung

$$\frac{n}{b} = \sum_{i=1}^{n} s_i + \frac{ns_n e^{-bs_n}}{(1-e^{-bs_n})} \tag{9.3.8}$$

Durch das gleichzeitige Lösen der Gleichungen (9.3.7) und (9.3.8) können Schätzwerte für a und b ermittelt werden.

Seien nun $\hat{a}$ und $\hat{b}$ die Schätzwerte von a und b. Dann können wir schließlich die Maximum Likelihood Schätzung der Mittelwertfunktion und der Zuverlässigkeit folgendermaßen ausdrücken.

$$\hat{m}(t) = \hat{a}[1 - e^{-\hat{b}t}]$$
$$\hat{R}(x|t) = e^{-\hat{a}[e^{-\hat{b}t} - e^{-\hat{b}(t+x)}]}$$

9.4 Ein NHPP Imperfect Debugging Modell

Bisher sind Modelle besprochen worden, die davon ausgehen, dass Fehler sofort und mit Sicherheit korrigiert werden sobald ein Ausfall eintritt. Die Anzahl der verbleibenden Fehler ist daher eine fallende Funktion bezüglich der Zeit. In diesem Fall spricht man auch von einer vollständigen Fehlerbehebung oder „perfect debugging“. [23]

In der Realität kommt es allerdings auch vor, dass die Behebung des Fehlers misslingt. Das ist der Fall, wenn etwa der Fehler nicht oder unvollständig entfernt oder aber neue Fehler eingeführt werden. Die Anzahl der verbleibenden Fehler ist daher nicht unbedingt fallend, sondern zeitabhängig. Man spricht auch von unvollständiger Fehlerbehebung oder „imperfect debugging“. [23]

In diesem Kapitel soll ein imperfect debugging Modell vorgestellt werden. Ich gehe dazu nach [23] mit folgender Notation vor.

a	erwartete Anzahl an anfänglichen Fehlern
b_j	Fehlererkennungsrate je Fehlerkategorie, $j = 1, 2, 3$; $0 < b_1 < b_2 < b_3 < 1$
p_j	Anteil je Fehlerkategorie j
$\lambda(t)$	Ausfallintensität oder Fehlererkennungsrate
$N_j(t)$	kumulative Anzahl an erkannten Fehlern je Fehlerkategorie i
$n(t)$	erwartete Anzahl an anfänglichen Fehlern und neu eingeführten Fehlern zur Zeit t
β_j	Fehlereinführungsrate je Fehlerkategorie i, mit $0 \leq \beta_j < 1$
$m(t)$	erwartete Anzahl an bis zur Zeit t erkannten Fehlern je Fehlerkategorie j

Das Modell geht von folgenden Annahmen aus:

- Im Zuge der Fehlerbehebung von beobachteten Ausfällen können neue Fehler entstehen.
- Die Wahrscheinlichkeit einen Ausfall zu beobachten, also einen Fehler zu finden, ist proportional zur Anzahl an verbleibenden Fehlern.
- Die Wahrscheinlichkeit einen neuen Fehler einzuführen ist konstant.
- Es wird von drei Kategorien von Fehlern ausgegangen: kritisch, schwer und einfach. Die Unterscheidung erfolgt anhand der Schwere, den Fehler zu beobachten und zu korrigieren.
- Die Parameter a und b_j für $j = 1, 2, 3$ sind unbekannte Konstanten.
- Das Finden von Fehlern folgt einem nicht homogenen Poisson Prozess.

Die Annahmen spiegeln sich in folgenden Differentialgleichungen wieder.

Die Fehlererkennungsrate zur Zeit t ist proportional zur Anzahl an im System verbleibenden Fehlern, also

$$m_j'(t) = b_j[n_j(t) - m_j(t)] \tag{9.4.1}$$

Die Fehlererkennungsrate ist außerdem proportional zur Veränderung der Gesamtanzahl an Fehlern zur Zeit t, also

$$n_j'(t) = \beta_j m_j'(t) \tag{9.4.2}$$

Gleichzeitig setzt sich die erwartete Anzahl an erkannten Fehlern zur Zeit t aus der Anzahl an erkannten Fehlern je Fehlerkategorie zusammen.

$$m(t) = \sum_{i=1}^{3} m_j(t) \tag{9.4.3}$$

Zu Beginn des Beobachtungszeitraums entspricht die erwartete Anzahl an Fehlern je Kategorie ihrem Anteil an der anfänglich erwarteten Gesamtanzahl an Fehlern.

$$n_j(0) = ap_j \tag{9.4.4}$$

Außerdem wurden zur Zeit $t = 0$ noch keine Fehler – und somit keine Fehler je Kategorie – gefunden.

$$m_j(0) = 0 \tag{9.4.5}$$

Nun soll das Differentialgleichungssystem gelöst werden. Dazu betrachten wir zunächst Gleichung (9.4.2). Diese kann auch geschrieben werden als

$$n_j'(t) - \beta_j m_j'(t) = 0$$

Also

$$(n_j(t) - \beta_j m_j(t))' = 0$$

Aufgrund des Mittelwertsatzes muss es eine Konstante c_0 geben, so dass

$$n_j(t) - \beta_j m_j(t) = c_0$$

Es erfolgt die Umformung nach $n_j(t)$.

$$n_j(t) = \beta_j m_j(t) + c_0 \tag{9.4.6}$$

Dieser Ausdruck soll nun in die Differentialgleichung (9.4.1) eingesetzt werden.

$$m_j'(t) = b_j[(\beta_j m_j(t) + c_0) - m_j(t)]$$

Durch das Auflösen der Klammern und Herausheben erhalten wir folgende inhomogene Differentialgleichung

$$m_j'(t) = b_j(\beta_j - 1)m_j(t) + b_j c_0 \tag{9.4.7}$$

Die zugehörige homogene Differentialgleichung ist somit

$$m_j'(t) = b_j(\beta_j - 1)m_j(t)$$

Diese kann wie die Differentialgleichung in Kapitel 9.3 gelöst werden. Also

$$\begin{aligned}
\frac{\mathrm{d}m_{j_0}}{m_{j_0}(t)} &= b_j(\beta_j - 1)\mathrm{d}t \\
\int \frac{\mathrm{d}m_{j_0}}{m_{j_0}(t)} &= \int b_j(\beta_j - 1)\mathrm{d}t \\
\log m_{j_0} &= b_j(\beta_j - 1)t + \log c_1
\end{aligned}$$

Wir erhalten als Lösung für das homogenen System

$$m_{j_0}(t) = c_1 e^{b_j(\beta_j - 1)t}$$

Nun soll die Methode der Variation der Konstanten angewendet werden, also

$$\begin{aligned}
c &\to c(t) \\
m_{j_0}(t) &\to m_j(t)
\end{aligned}$$

Somit hat $m_j(t)$ folgende Form

$$m_j(t) = c_1(t)e^{b_j(\beta_j-1)t} \tag{9.4.8}$$

Diese Gleichung soll nun gemäß der Produktregel abgeleitet werden.

$$m_j'(t) = c_1'(t)e^{b_j(\beta_j-1)t} + b_j(\beta_j-1)c_1(t)e^{b_j(\beta_j-1)t}$$

Gleichzeitig kann der Ausdruck aus (9.4.8) in die Differentialgleichung (9.4.7) eingesetzt werden. Also

$$m_j'(t) = b_j(\beta_j-1)c_1(t)e^{b_j(\beta_j-1)t} + b_jc_0$$

Die ermittelten Gleichungen können gleichgesetzt werden. Durch Umformung erhalten wir dann die Ableitung von $c_1'(t)$.

$$\begin{aligned} c_1'(t)e^{b_j(\beta_j-1)t} + b_j(\beta_j-1)c_1(t)e^{b_j(\beta_j-1)t} &= b_j(\beta_j-1)c_1(t)e^{b_j(\beta_j-1)t} + c_0 \\ c_1'(t)e^{b_j(\beta_j-1)t} &= b_jc_0 \\ c_1'(t) &= b_jc_0e^{-b_j(\beta_j-1)t} \end{aligned}$$

Durch Integration kann $c_1(t)$ ermittelt werden.

$$\begin{aligned} c_1'(t) &= \int b_jc_0e^{-b_j(\beta_j-1)t}\mathrm{d}t \\ &= -\frac{b_jc_0}{b_j(\beta_j-1)}e^{-b_j(\beta_j-1)t} + c_2 \\ &= \frac{c_0}{1-\beta_j}e^{-b_j(\beta_j-1)t} + c_2 \end{aligned}$$

Diese Information kann für Gleichung (9.4.8) verwendet werden.

$$\begin{aligned} m_j(t) &= \left[\frac{c_0}{1-\beta_j}e^{-b_j(\beta_j-1)t} + c_2\right]e^{b_j(\beta_j-1)t} \\ &= \frac{c_0}{1-\beta_j} + c_2e^{b_j(\beta_j-1)t} \end{aligned}$$

Betrachten wir nun die Anfangsbedingung $m_j(0) = 0$, so können wir c_2 ermitteln.

$$\begin{aligned} m_j(0) &= \frac{c_0}{1-\beta_j} + c_2 = 0 \\ c_2 &= -\frac{c_0}{1-\beta_j} \end{aligned}$$

Gleichzeitig kann diese Anfangsbedingung in Gleichung (9.4.6) verwendet werden, um c_0 zu

erhalten.

$$\begin{aligned} n_j(t) &= \beta_j m_j(t) + c_0 \\ n_j(0) &= c_0 \end{aligned}$$

Da gemäß der ursprünglichen Annahme $n_j(0) = ap_j$ gilt, kennen wir somit auch c_0

$$c_0 = ap_j$$

Nun können wir c_2 sowie c_0 in die Gleichung für $m_j(t)$ einsetzen.

$$\begin{aligned} m_j(t) &= \frac{c_0}{1-\beta_j} - \frac{c_0}{1-\beta_j} e^{b_j(\beta_j - 1)t} \\ &= \frac{ap_j}{1-\beta_j} - \frac{ap_j}{1-\beta_j} e^{b_j(\beta_j - 1)t} \end{aligned}$$

Durch Herausheben erhalten wir als Ergebnis für die Mittelwertfunktion $m_1(t)$ folgenden Ausdruck.

$$m_j(t) = \frac{ap_j}{1-\beta_j}[1 - e^{-(1-\beta_j)b_j t}]$$

Aufgrund von Gleichung (9.4.6) können wir nun auch $n_j(t)$ ermitteln.

$$\begin{aligned} n_j(t) &= \beta_j m_j(t) + c_0 \\ &= \frac{ap_j\beta_j}{1-\beta_j}[1 - e^{-(1-\beta_j)b_j t}] + ap_j \\ &= ap_j \left[\frac{\beta_j}{1-\beta_j} - \frac{\beta_j}{1-\beta_j} e^{-(1-\beta_j)b_j t} + 1\right] \\ &= ap_j \left[\frac{\beta_j}{1-\beta_j} - \frac{\beta_j}{1-\beta_j} e^{-(1-\beta_j)b_j t} + \frac{1-\beta_j}{1-\beta_j}\right] \\ &= \frac{ap_j}{1-\beta_j} \left[\beta_j - \beta_j e^{-(1-\beta_j)b_j t} + 1 - \beta_j\right] \end{aligned}$$

Somit erhalten wir

$$n_j(t) = \frac{ap_j}{1-\beta_j} \left[1 - \beta_j e^{-(1-\beta_j)b_j t}\right]$$

Die Ausfallintensität $\lambda_j(t)$ ist dann

$$\lambda_j(t) = m_j'(t) = ap_j b_j e^{-(1-\beta_j)b_j t}$$

Die bedingte Zuverlässigkeit hat gemäß Kapitel 6.4 die Form

$$R(x|t) = e^{-[m(t+m) - m(t)]}$$

Setzt man die Mittelwertfunktion ein, erhalten wir also

$$R(x|t) = e^{-\left[\sum_{i=1}^{3} \frac{ap_j}{1-\beta_j}(e^{-(1-\beta_j)b_j t})(1-e^{-(1-\beta_j)b_j x})\right]}$$

Nun können wir die Parameter a und b_j über die Maximum Likelihood Schätzung ermitteln.

9.4.1 Maximum Likelihood Schätzung im Fall von Intervallbereichsdaten

Wie bereits zuvor wird im Fall von Intervallbereichsdaten die kumulative Anzahl an Ausfällen – für dieses Modell je Fehlerkategorie – in Zeitintervallen beobachtet und dokumentiert. Die Daten haben dann die Form $(t_i, y_{i,j})$, mit $y_{i,j}$ Ausfällen des Typs j, $j = 1, 2, 3$, zur Zeit t_i, $i = 1, 2, \ldots, n$.

Analog zu Kapitel 8.3 interessieren wir uns in diesem Fall für $P\{N(t_i) = y_i\}$ beziehungsweise $P\{N_j(t_n) = y_{n,j}\}$. Somit ist die Likelihood Funktion für diesen Fall gegeben durch

$$L(a, b_1, b_2, b_3|(t_i, y_i)) = \prod_{j=1}^{3}\prod_{i=1}^{n} \frac{[m_j(t_i) - m_j(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} e^{-[m_j(t_i) - m_j(t_{i-1})]}$$

Die logarithmierte Funktion ist dann

$$\begin{aligned}\log L(a, b_1, b_2, b_3|(t_i, y_i) = &\sum_{j=1}^{3}\sum_{i=1}^{n}[(y_{i,j} - y_{i-1,j}) \log[m_j(t_i) - m_j(t_{i-1})] \\ &- \log[(y_{i,j} - y_{i-1,j})!] - [m_j(t_i) - m_j(t_{i-1})]]\end{aligned}$$

Analog zu Kapitel 8.3 erhalten wir durch partielles Differenzieren und Nullsetzten ein Gleichungssystem, um die gesuchten Schätzwerte zu ermitteln.

$$0 = \sum_{j=1}^{3}\left[\sum_{i=1}^{n} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta}m_j(t_i) - \frac{\mathrm{d}}{\mathrm{d}\Theta}m_j(t_{i-1})}{m_j(t_i) - m_j(t_{i-1})}(y_{i,j} - y_{i-1,j}) - \frac{\mathrm{d}}{\mathrm{d}\Theta}m_j(t_n)\right]$$

Dabei ist die Mittelwertfunktion $m_j(t_i)$ gegeben durch

$$m_j(t_i) = \frac{ap_j}{(1-\beta_j)}[1 - e^{-(1-\beta_j)b_j t_i}]$$

Die partielle Ableitung nach a ist sodann

$$\frac{\mathrm{d}}{\mathrm{d}a}m_j(t_i) = \frac{p_j}{(1-\beta_j)}[1 - e^{-(1-\beta_j)b_j t_i}]$$

Setzen wir diese beiden Ausrücke in das Gleichungssystem ein, so erhalten wir

$$\begin{aligned}
0 &= \sum_{j=1}^{3}\sum_{i=1}^{n} \frac{\frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_i}] - \frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_{i-1}}]}{\frac{ap_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_i}] - \frac{ap_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_{i-1}}]}(y_{i,j}-y_{i-1,j}) \\
&\quad - \frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_n}] \\
&= \sum_{j=1}^{3}\left[\sum_{i=1}^{n}\frac{1}{a}(y_{i,j}-y_{i-1,j}) - \frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_n}]\right] \\
&= \sum_{j=1}^{3}\frac{y_{n,j}}{a} - \sum_{j=1}^{3}\frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_n}]
\end{aligned}$$

Durch Umformung erhalten wir

$$\sum_{j=1}^{3}\frac{y_{n,j}}{a} = \sum_{j=1}^{3}\frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_n}]$$

Der Schätzer für a ist somit

$$a = \frac{\sum_{j=1}^{3} y_{n,j}}{\sum_{j=1}^{3}\frac{p_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_n}]}$$

In ähnlicher Weise sollen Schätzwerte für b_j, mit $j = 1,2,3$, gefunden werden. Die partielle Ableitung von $m_j(t_i)$ nach b_j ist wie folgt.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}b_j}m_j(t_i) &= \frac{ap_j(1-\beta_j)t_i}{(1-\beta_j)}e^{-(1-\beta_j)b_j t_i} \\
&= ap_j t_i e^{-(1-\beta_j)b_j t_i}
\end{aligned}$$

Erneut können wir in das Gleichungssystem einsetzen. Wir erhalten für $j = 1,2,3$ jeweils eine Gleichung der folgenden Form.

$$\begin{aligned}
0 = \sum_{i=1}^{n} &\frac{ap_j t_i e^{-(1-\beta_j)b_j t_i} - ap_j t_{i-1} e^{-(1-\beta_j)b_j t_{i-1}}}{\frac{ap_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_i}] - \frac{ap_j}{(1-\beta_j)}[1-e^{-(1-\beta_j)b_j t_{i-1}}]}(y_{i,j} \\
&- y_{i-1,j}) - ap_j t_i e^{-(1-\beta_j)b_j t_n}
\end{aligned}$$

Durch Kürzen und Umformen erhalten wir als Ergebnis das Gleichungssystem mit $j = 1,2,3$.

$$\begin{aligned}
\sum_{i=1}^{n}(y_{i,j}-y_{i-1,j})&\frac{(1-\beta_j)[t_i e^{-(1-\beta_j)b_j t_i} - t_{i-1}e^{-(1-\beta_j)b_j t_{i-1}}]}{[e^{-(1-\beta_j)b_j t_{i-1}} - e^{-(1-\beta_j)b_j t_i}]} \\
&= ap_j t_n e^{-(1-\beta_j)b_j t_n}
\end{aligned}$$

Wurden die Ausfalldaten als Intervallbereichsdaten erfasst, so können die gesuchten Schätzwerte für a, b_1, b_2 und b_3 durch das Lösen des eben ermittelten Gleichungssystems berechnet werden.

9.4.2 Maximum Likelihood Schätzung im Fall von Zeitbereichsdaten

Bei diesem Modell werden im Fall von Zeitbereichsdaten die Zeitpunkte der Ausfälle je Fehlerkategorie beobachtet. Es soll angenommen werden, dass die erwartete Anzahl an Fehlern je Fehlerkategorie $j = 1, 2, 3$ mit n_1, n_2 und n_3 gegeben ist. Außerdem sind $S_{j,i}$ mit $S_{1,1} \leq S_{1,2} \leq \cdots \leq S_{1,n_1}, S_{2,1} \leq S_{2,2} \leq \cdots \leq S_{2,n_2}, S_{3,1} \leq S_{3,2} \leq \cdots \leq S_{3,n_3}$ die beobachteten Zeiten, zu denen der i-te Fehler der Kategorie j eintritt.

Die Likelihood Funktion ist dann, wie in Kapitel 8.3 erläutert, gegeben durch

$$L(a, b_1, b_2, b_3 | s_i) = \prod_{j=1}^{3} \prod_{i=1}^{n_j} e^{-m_j(S_r)} \lambda_j(S_{j,i})$$

Dabei bezeichnet S_r den Zeitpunkt des letzten beobachteten Fehlers über alle Kategorien. Also

$$S_r = \max\{S_{1,n_1}, S_{2,n_2}, S_{3,n_3}\}$$

Gemäß Gleichung (8.3.1) aus Kapitel 8.3 muss nun folgendes Gleichungssystem gelöst werden, um die verschiedenen Parameter schätzen zu können.

$$0 = \sum_{j=1}^{3} \left[\sum_{i=1}^{n_j} \frac{\frac{\mathrm{d}}{\mathrm{d}\Theta} \lambda_j(S_i)}{\lambda_j(S_i)} - \frac{\mathrm{d}}{\mathrm{d}\Theta} m_j(S_r) \right]$$

Wir kennen bereits die Formen von m_j und λ_j.

$$\begin{aligned} m_j(s_i) &= \frac{a p_j}{1 - \beta_j} [1 - e^{-(1-\beta_j) b_j s_i}] \\ \lambda_j(s_i) &= a p_j b_j e^{-(1-\beta_j) b_j s_i} \end{aligned}$$

Ebenso haben wir bereits die partiellen Ableitungen von m_j ermittelt. Diese sind

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}a} m_j(s_i) &= \frac{p_j}{(1 - \beta_j)} [1 - e^{-(1-\beta_j) b_j s_i}] \\ \frac{\mathrm{d}}{\mathrm{d}b_j} m_j(s_i) &= a p_j s_i e^{-(1-\beta_j) b_j s_i} \end{aligned}$$

Weiters werden die partiellen Ableitungen von λ_j benötigt. Also

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}a} \lambda_j(s_i) &= p_j b_j e^{-(1-\beta_j) b_j s_i} \\ \frac{\mathrm{d}}{\mathrm{d}b_j} \lambda_j(s_i) &= a p_j e^{-(1-\beta_j) b_j s_i} - a p_j b_j s_i (1 - \beta_j) e^{-(1-\beta_j) b_j s_i} \\ &= a p_j e^{-(1-\beta_j) b_j s_i} [1 - b_j s_i (1 - \beta_j)] \end{aligned}$$

Setzen wir nun λ_j und die Ableitungen nach a in das Gleichungssystem ein, so erhalten wir

$$0 = \sum_{j=1}^{3} \left[\sum_{i=1}^{n_j} \frac{p_j b_j e^{-(1-\beta_j)b_j s_i}}{a p_j b_j e^{-(1-\beta_j)b_j s_i}} - \frac{p_j}{(1-\beta_j)} [1 - e^{-(1-\beta_j)b_j s_r}] \right]$$

Nun soll die Gleichung umgeformt werden.

$$\begin{aligned}
0 &= \sum_{j=1}^{3} \left[\sum_{i=1}^{n_j} \frac{1}{a} - \frac{p_j}{(1-\beta_j)} [1 - e^{-(1-\beta_j)b_j s_r}] \right] \\
0 &= \sum_{j=1}^{3} \frac{n_j}{a} - \sum_{j=1}^{3} \frac{p_j}{(1-\beta_j)} [1 - e^{-(1-\beta_j)b_j s_r}] \\
\sum_{j=1}^{3} \frac{n_j}{a} &= \sum_{j=1}^{3} \frac{p_j}{(1-\beta_j)} [1 - e^{-(1-\beta_j)b_j s_r}]
\end{aligned}$$

Wir erhalten somit für a folgenden Ausdruck als Ergebnis.

$$a = \frac{\sum_{j=1}^{3} n_j}{\sum_{j=1}^{3} \frac{p_j}{(1-\beta_j)} [1 - e^{-(1-\beta_j)b_j s_r}]}$$

In gleicher Weise werden λ sowie die partiellen Ableitungen nach b_j in das Gleichungssystem eingesetzt. Wir erhalten für $j = 1, 2, 3$ jeweils eine Gleichung der Form

$$0 = \sum_{i=1}^{n_j} \frac{a p_j e^{-(1-\beta_j)b_j s_i} [1 - b_j s_{j,i} (1-\beta_j)]}{a p_j b_j e^{-(1-\beta_j)b_j s_{j,i}}} - a p_j s_r e^{-(1-\beta_j)b_j s_r}$$

Auch dieses Gleichungssystem soll vereinfacht werden.

$$\begin{aligned}
0 &= \sum_{i=1}^{n_j} \frac{1 - b_j s_{j,i} (1-\beta_j)}{b_j} - a p_j s_r e^{-(1-\beta_j)b_j s_r} \\
&= \sum_{i=1}^{n_j} \frac{1}{b_j} - \sum_{i=1}^{n_j} \frac{b_j s_{j,i} (1-\beta_j)}{b_j} - a p_j s_r e^{-(1-\beta_j)b_j s_r} \\
&= \frac{n_j}{b_j} - \sum_{i=1}^{n_j} s_{j,i} (1-\beta_j) - a p_j s_r e^{-(1-\beta_j)b_j s_r} \\
&= \frac{n_j - a p_j b_j s_r e^{-(1-\beta_j)b_j s_r}}{b_j} - \sum_{i=1}^{n_j} s_{j,i} (1-\beta_j)
\end{aligned}$$

Wir erhalten für $j = 1, 2, 3$

$$\sum_{i=1}^{n_j} S_{j,i} = \frac{n_j - a p_j b_j S_r e^{-(1-\beta_j)b_j S_r}}{b_j (1-\beta_j)}$$

Für Zeitbereichsdaten können die gesuchten Parameter durch das Lösen des nun gefundenen

Gleichungssystems ermittelt werden.

9.5 Vergleich NHPP perfect und imperfect debugging Modelle

In den vorherigen beiden Kapiteln konnten zwei NHPP Modelle kennengelernt werden, deren Mittelwertfunktionen $m(t)$ einerseits von der Anzahl an Fehlern im System a, andererseits von der Fehlererkennungsrate b abhängig sind. Beide Parameter können jeweils von der Zeit abhängig oder konstant sein.

Im einfachsten Modell sind beide Parameter konstant. Das ist im Goel-Okumoto NHPP Modell der Fall. In den meisten Modellen wird allerdings davon ausgegangen, dass die Fehlererkennungsrate $b(t)$ von der Anzahl an Fehlern im System abhängt. Gleichzeitig wird die Anzahl an Fehlern dann als konstant angenommen, also $a(t) = a$. In diesen Modellen wird somit davon ausgegangen, dass während der Fehlerkorrektur keine neuen Fehler eingeführt werden (perfect debugging). [23]

Ist umgekehrt die Anzahl an im System verbleibenden Fehlern zeitabhängig, geht man davon aus, dass nicht jede Fehlerkorrektur erfolgreich ist, also dabei neue Fehler eingeführt werden können. Wird nun weiters angenommen, dass die Fehlererkennungsrate proportional zur Anzahl an Fehlern im System und somit konstant ist, werden diese Modelle als imperfect debugging Modelle bezeichnet. [23] Ein solches Modell haben wir in Kapitel 9.4 kennengelernt.

Ein Modell, in dem beide Paramter als von der Zeit abhängig angenommen werden, ist etwa das generalized NHPP Modell von [23].

Auch wenn in der Realität davon ausgegangen werden sollte, dass durch Fehlerkorrekturen neue Fehler in die Software eingeführt werden können, hat sich in einer Studie von [22] gezeigt, dass bei den untersuchten Modellen durch die Erweiterung um die imperfect debugging Annahme keine signifikante Verbesserung festzustellen war. In der Studie wurden perfect debugging Modelle um die Annahme erweitert, dass im Zuge von Fehlerkorrekturen neue Fehler eingeführt werden können, wobei die Fehlereinführungsrate als konstant angenommen wurde. Die ermittelten Mittelwertfunktionen waren dann isomorph zu den Mittelwertfunktionen im perfect debugging Modell. Gemäß den Autorinnen sollte im Weiteren untersucht werden, ob das auch für Modelle anderer Form gilt, sowie ob die Annahme bezüglich der konstanten Fehlereinführungsrate mit der Realität übereinstimmt. [22]

9.6 Weitere Modellparameter

Abgesehen von der Berücksichtigung einer zeitabhängigen Fehlererkennungsrate beziehungsweise Anzahl an Fehlern im System werden in der Literatur auch weitere Aspekte, Überlegungen und Annahmen zur Modellierung herangezogen.

Nach [6] bietet sich etwa eine Möglichkeit, Systemeigenschaften wie die Anzahl an Code-Zeilen oder die Architektur der Software zu berücksichtigen.

Ebenso hat sich laut [6] gezeigt, dass Modelle, die Informationen aus vergleichbaren Projekten oder früheren Releasezyklen berücksichtigen, eine hohe Verlässlichkeit und gute Vorhersagekraft

bieten. Das ist etwa dann der Fall, wenn die Ausfalldichte von Produkten der gleichen Entwicklerfirma in ähnlichem Zeitraum oder die Ausfalldichten der letzten ein bis zwei Releases des zu untersuchenden Produkts einbezogen werden.

Auch die Anzahl an gefundenen Fehlern in frühen Phasen der Software-Entwicklung kann zur Modellierung verwendet werden. [6]

Ein Bereich, der nach [6] noch wenig Beachtung gefunden hat, ergibt sich aus den Informationen über die zugrunde liegenden Testfälle, die bei der Datengewinnung verwendet werden. So werden Testfälle im Allgemeinen als einzigartig angenommen. Betrachtet man allerdings modellbasierte Testfallermittlung, kann festgestellt werden, dass diese auf einem Meta-Modell an Anwendungsfällen und Einschränkungen basiert. [6]

In einigen Modellen finden auch Informationen bezüglich der Testabdeckung Berücksichtigung, etwa in [24].

Abgesehen von ergänzenden Parametern, die dazu dienen sollen, die Kurvenanpassung oder Vorhersagekraft der Wahrscheinlichkeitsverteilungen in Bezug auf den Grad der Software-Zuverlässigkeit zu verbessern, besteht wie eingangs besprochen eine Intention in der Software-Zuverlässigkeits-Analyse in der Fragestellung nach dem optimalen Testende hinsichtlich Kosten und Risiken. Hier interessiert, wie sich die Kosten während der Software-Entwicklung zu den potentiellen Kosten – etwa für Fehlerkorrekturen nach Markteinführung – verhalten. Kosten, die hierbei typischerweise berücksichtigt werden, sind nach [16] die Kosten für die Fehlerkorrektur während der Testphase, Kosten für die Fehlerkorrektur während der Betriebsphase, sowie die Kosten für das Testen an sich.

Auch wenn weitere Kosten-Komponenten in verschiedenen Modellen berücksichtigt werden, so geht es doch immer um eine Abwägung zwischen den Kosten der Softwareentwickung beziehungsweise dem Softwaretest im Vergleich zu den Vorteilen und Risiken einer frühen Markteinführung.

9.7 Schwierigkeiten in Bezug auf gängige Modellannahmen

Wie bereits festgestellt, werden im Bereich der Software-Zuverlässigkeit Modelle anhand von Annahmen gewählt. Dabei kommt es allerdings immer wieder zu Differenzen zwischen diesen und der Realität.

So ist eine typische Annahme, dass die Software abgesehen von der Entfernung der gefundenen Fehler während der Testphase nicht verändert wird. Allerdings sind insbesondere bei großen Systemen die Software-Entwicklungs-Phasen oft nicht strikt getrennt und können parallel laufen. Das kann sich auf die Ausfallrate auswirken, so dass klassische Modelle, die sich nur auf Fehler pro Zeiteinheit beziehen, inadäquat sein können. [6, 15] Nach [6] ist ein möglicher Lösungsansatz, das Modell um eine weitere unabhängige Variable zu ergänzen, die sich auf die Anzahl an neu hinzukommenden Codezeilen bezieht, sowie einer Schätzung, wie viele Fehler pro ergänzten zigtausend Zeilen eingeführt werden.

Eine weitere typische Annahme bezieht sich darauf, dass Fehler umgehend nach ihrer Entdeckung behoben werden. Oft ist dies allerdings nicht der Fall und der gleiche Fehler führt

möglicherweise mehrfach zu einem Ausfall. Nach dieser Modellannahme dürfte derselbe Fehler jedoch nicht mehrfach zu einem Ausfall führen, da er bereits behoben sein müsste. Insofern schlagen die Autoren von [15] als Behelf vor, die entsprechenden Ausfälle nicht doppelt zu berücksichtigen und die Testzeiten zu summieren, wenn die nicht korrigierten Fehler dazu führen, dass andere Fehler nicht entdeckt werden können.

Schließlich können durch die Korrekturen neue Fehler eingeführt werden. Da jedoch bereits intensiv getestet wurde, werden diese neuen Fehler möglicherweise mit einer geringeren Wahrscheinlichkeit entdeckt, da ein Retest oft nicht so gründlich erfolgt wie die erste Prüfung. [15]

Auch [14] weist darauf hin, dass die Zeit, die zur Fehlerbehebung benötigt wird, nicht vernachlässigbar ist.

Selbst die Annahme, dass Fehler unabhängig voneinander sind, ist nach [15] mit Vorsicht zu betrachten. Demnach kann es vorkommen, dass bestimmte Code-Abschnitte beziehungsweise Software-Bereiche weniger intensiv getestet werden als andere Abschnitte. In weiterer Folge können dann in den wenig getesteten Bereichen während der Betriebsphase überdurchschnittlich viele Ausfälle beobachtet werden. [15]

Eine weitere Grundannahme ist, dass der Softwaretest annähernd gleichförmig voranschreitet, also jede Zeiteinheit gleichwertig ist. [6, 15] Für Software-Zuverlässigkeits-Modelle verwendete typische Einheiten sind Kalenderzeit, Computer-Ausführungszeit (CPU) oder etwa die Anzahl an Testfällen. Ein geeignetes Zeitmaß muss dabei in Zusammenhang mit dem Testaufwand stehen. Dies ist allerdings für jede dieser Einheiten problematisch. So verläuft im Allgemeinen der Testaufwand asynchron in Bezug auf die Kalenderzeit, manche Tests produzieren mehr Last in Bezug auf die Ausführungszeit und nicht jeder Testfall hat die gleiche Wahrscheinlichkeit einen Fehler zu finden. [15] In [6] wird darauf hingewiesen, dass nach [19] das Messen der Ausführungszeit als am erfolgreichsten betrachtet wird.

Ein weiteres Problemfeld ergibt sich bei der Erfassung von Fehlern. Oft besteht die Annahme, dass lediglich das Testteam Fehler einmeldet. In der Praxis können allerdings Fehler von verschiedenen Personengruppen – beispielsweise Fachbereichen oder Entwickler/innen – gefunden und dokumentiert werden. Werden die so gefundenen Fehler berücksichtigt, müssen die Testzeiten dieser Personengruppen in die erhobenen Daten integriert werden. Dabei kann sich die Effizienz des Testens (etwa aufgrund unterschiedlicher Testtypen) unterscheiden und somit sind die Zeiteinheiten nicht mehr vergleichbar. Werden umgekehrt die so gefundenen Fehler nicht berücksichtigt, so werden wichtige Daten vernachlässigt. [15]

Eine ähnliche Problematik entsteht, wenn sich die Organisation verändert. Werden mehrere Versionen einer Software entwickelt oder ist der Projektzeitraum länger, kann es zu signifikanten Änderungen des Umfelds, beispielsweise des Entwicklungs-Vorgehensmodells oder der Teamzusammensetzung, kommen, und somit auch einer Änderung der Teststrategie. [15]

Schließlich liegt eine Schwierigkeit in der Definition der operativen Profile. Oft kann – beispielsweise im Fall von neuer Software – die tatsächliche Verwendung während der Betriebsphase schwer abgeschätzt werden. Für die Verwendung von Userprofilen müssen allerdings Klassen an Eingabebereichen mit ihren jeweiligen Eintrittswahrscheinlichkeiten spezifiziert werden. Dementgegen werden während der Testphase Testfälle oft kontinuierlich ergänzt. Kommt es also zu

einer Fehleinschätzung bei der Erstellung der Userprofile oder einer Abweichung von diesen im Testvorgehen, wirkt sich das auf die geschätzte Zuverlässigkeit aus. [15] Ähnlich verhält es sich, wenn das System anders oder in einer anderen Umgebung als geplant eingesetzt wird. Auch in diesen Fällen kann es zu Abweichungen zwischen Realität und Schätzung kommen. [23]

9.8 Wahl eines geeigneten Modells

Laut [6] werden Modelle oft dahingehend beurteilt, wie gut die Kurve den beobachteten Daten entspricht. Allerdings kann eine gute Kurvenanpassung auch aufgrund einer Überanpassung entstehen. Dementsprechend kann aus dieser Eigenschaft alleine noch keine Aussage darüber getroffen werden, wie geeignet das Modell ist, um zukünftige Fehler im gegenwärtigen System vorherzusagen. [6] Auch [27] stellt fest, dass die meisten Studien auf eine gute Kurvenanpassung fokussiert sind, nicht aber auf die Vorhersagekraft der Modelle.

Gleichzeitig kritisiert [23], dass sich gängige Modelle nur auf Fehlerdaten beziehen, weitere Informationen jedoch außer Acht gelassen werden. Dadurch werden große Datenmengen benötigt, um die Zuverlässigkeit treffend vorhersagen zu können.

Insgesamt wird bemängelt, dass zur Abschätzung der Software-Zuverlässigkeit beziehungsweise zur Wahl eines geeigneten Modells zunächst ausreichend Daten gesammelt werden müssen, und somit bei herkömmlichem Vorgehen keine schnellen Abschätzungen oder Entscheidungen möglich sind. [27, 6, 23],

Das von [27] vorgeschlagene Vorgehen beinhaltete daher folgende Schritte:

- Es ist sicherzustellen, dass die richtigen Daten (aus dem Test) gesammelt werden. Dazu soll der Software-Entwicklungs-und Testprozess in der entsprechenden Herstellerfirma verstanden werden, da die Fehlerdokumentation und der zugehörige Managementprozess beeinflussen, welche Daten (beispielsweise welche Datenbank oder welche Phase im Software Lebenszyklus) am besten für die Analyse geeignet sind.

- Vor der Wahl eines Modells sollten die Daten für eine erste Analyse visualisiert werden. Die explorative Analyse von aktuellen und vergangenen Projekten hilft, die Form des Fehlerzuflusses zu verstehen und entsprechende Profile zu erstellen.

- Die Ziele für die Verwendung eines Wahrscheinlichkeitsmodells sollen klar definiert werden. Insbesondere sollte entschieden werden, ob das Modell zur Beurteilung eingesetzt werden soll, ob das Produkt die geforderten Qualitätsansprüche zur Freigabe für die Betriebsphase erfüllt, oder ob das Modell bei der effektiven Verteilung der Test-Ressourcen und der Erreichung der Zuverlässigkeits-Forderungen in einer gewünschten Zeitspanne unterstützen soll. Je nach Zielsetzung können unterschiedliche Modelle besser geeignet sein.

- Für die Wahl eines geeigneten Modells sollte zunächst eine Auswahl an Modellen aufgrund von Fehlerzufluss-Profilen aus vergangener und dem aktuellen Projekt getroffen werden. Danach sollten die Modelle anhand der Daten evaluiert und so das beste daraus ermittelt werden.

- Schon während eines laufenden Projekts kann die Form des Fehlerzuflusses vorhergesagt werden und so die Wahl des Modells erleichtert (und beschleunigt) werden.
- Durch die Verwendung von Informationen aus früheren Projekten kann die Vorhersagegenauigkeit verbessert werden. So können etwa durch den Vergleich mit ähnlichen Projekten Modellparameter reduziert werden.

Auch nach [6, 23] sollten Informationen aus frühen Phasen der Entwicklung beziehungsweise weitere Informationen mit einbezogen werden, um schneller sowie aussagekräftigere Prognosen anstellen zu können.

Demgegenüber ist der Vorschlag von [13], eine Kombination aus verschiedenen Modellen zeitgleich zu verwenden und diese mithilfe genetischer Algorithmen gewichtet zu kombinieren. So sollen Modelle, die tendenziell zu hoch oder zu niedrig schätzen, ausgeglichen werden.

10 Zusammenfassung, Reflexion und Ausblick

10.1 Zusammenfassung

In dieser Arbeit wurde versucht, die wichtigsten Grundlagen und Konzepte der Software-Zuverlässigkeit vorzustellen.

Dazu wurden zunächst grundlegende Begriffe wie „Qualität“, „Fehler“ oder „Ausfall“ erklärt. Weiters wurden Hintergründe aus der Software-Entwicklung und dem Softwaretest beziehungsweise der Qualitätssicherung in diesem Bereich vorgestellt, um so einen besseren Einblick in den Kontext zu erhalten. Ebenso wurde ein kurzer Vergleich zu Hardware-Zuverlässigkeit beziehungsweise zwischen reparierbaren und nicht-reparierbaren Objekten angestellt.

Gleichzeitig wurde versucht, Software-Zuverlässigkeit aus der Perspektive der Wahrscheinlichkeitstheorie darzulegen. Dazu erfolgte zunächst die mathematische Definition von Software-Zuverlässigkeit, sowie der einschlägigen Begriffe in diesem Zusammenhang. Zur Veranschaulichung wurde eines der ersten und einfachsten Modelle – das Jelinski-Moranda Modell – eingeführt.

Weiters konnten die wichtigsten mathematischen Hintergründe betrachtet werden. Es wurden einerseits vier Wahrscheinlichkeitsverteilungen, die in Zusammenhang mit (Software-) Zuverlässigkeitskonzepten von Bedeutung sind, erläutert. Anderseits konnte ein kurzer Einblick zu Punktschätzungen – mit besonderem Augenmerk auf dem Maximum Likelihood Prinzip – gewährt werden. Die Maximum Likelihood Schätzung wurde sodann auf die eben genannten Verteilungen angewendet.

Da sich im Verlauf der Arbeit die Bedeutung von NHPP Modellen aus den verschiedenen Software-Zuverlässigkeitsmodellen und Konzepten gezeigt hat, war es ein Anliegen, näher auf diese einzugehen. Daher wurden zwei NHPP Modelle näher vorgestellt und einem kurzen Vergleich unterzogen. Das sollte einerseits für einen besseren Einblick in das Vorgehen sorgen, andererseits als Basis dazu dienen, anschließend über einige Schwierigkeiten in der Anwendung von Modellen zu diskutieren – insbesondere über Annahmen, die über den Fehlerfindungsprozess

getroffen werden müssen und ihrer Diskrepanz zur Realität, sowie der Selektion von Modellen im Allgemeinen.

Wenngleich also keine umfassende und detaillierte Darstellung von Software-Zuverlässigkeit vorgenommen werden konnte, so konnte doch ein erster Einblick in den Anwendungskontext einerseits, sowie die wichtigsten mathematischen Grundlagen und Konzept andererseits gegeben werden.

10.2 Kontroverse in Bezug auf die Verwendung von Software-Zuverlässigkeits-Modellen

Während des Verfassens dieser Arbeit tauchten immer wieder Überlegungen bezüglich der praktischen Anwendung von Software-Zuverlässigkeitsmodellen auf. Insbesondere konnte einerseits der wahrgenommene Widerspruch zwischen Teststrategien und -methodik, die im Testumfeld zur Steigerung der Testabdeckung postuliert werden, und dem Testen nach Userprofilen zum Erhalten von Daten für sinnvolle Zuverlässigkeitsauswertungen nicht vollständig aufgelöst werden. Andererseits stellte sich bald die Frage, wann Software-Zuverlässigkeitsmodelle in der Praxis tatsächlich zum Einsatz kommen.

So finden laut [23] Software-Zuverlässigkeitsmodelle derzeit keine große Anwendung, da oft das Wissen bezüglich der Wahl und Verwendung eines geeigneten Modells fehlt. Ähnliches stellt [29] fest. Der Autor konstatiert, dass Systeme während des Software-Entwicklungs-Prozesses derzeit lediglich auf ihre Anforderungen geprüft werden, nicht jedoch auf ihre Zuverlässigkeit.

Auch [10] stellt fest, dass in der Industrie (standardisierte) Software-Zuverlässigkeits-Modelle kaum Beachtung und Anwendung finden. Die Autor/innen sehen die größte Problematik in der hohen Komplexität des Themas, sowie der Notwendigkeit, Software-Zuverlässigkeit aus verschiedenen Blickwinkeln zu betrachten, um die Erwartungen der unterschiedlichen Interessensgruppen zu erfüllen. [10]

Ein ähnliches Bild ergibt sich aus meinem beruflichen Kontext. Durch die vorangegangene Tätigkeit als Beraterin im Softwaretest einerseits und die laufende Teilnahme an Konferenzen im Bereich der Qualitätssicherung andererseits konnte ich Einblicke in die Software-Entwicklung unterschiedlicher Projekte erlangen beziehungsweise mich mit verschiedenen Personen aus diesem Umfeld austauschen. Betrachte ich diese beiden Kontexte, so scheint Software-Zuverlässigkeit wenig Beachtung zu finden. Demgegenüber stehen eine Unzahl an Modellen und eine große Menge an einschlägiger Literatur, sowie die Notwendigkeit, stabile und zuverlässige Software zu produzieren.

10.3 Ausblick

In dieser Arbeit wurde versucht, das Konzept der Software-Zuverlässigkeit aus mathematischer Perspektive aufzubereiten. Wenngleich eine möglichst umfassende Betrachtung angestrebt wurde, so ist doch klar, dass der Themenkomplex nur angerissen werden konnte. An vielen Stellen musste auf weitere Detaillierung verzichtet werden. Insbesondere konnte nicht auf die Vielzahl verschiedener Modelltypen und Modelle eingegangen werden.

Es wäre gewiss lohnenswert, Modelle zu betrachten oder zu entwickeln, die wie in Kapitel 9.6 kurz umrissen weitere Aspekte des Software-Entwicklungsprozesses berücksichtigen. Auch eine Evaluierung dieser bezüglich ihrer Vorhersagekraft und Anwendbarkeit wäre von großem Interesse. Insbesondere scheint es von Bedeutung zu sein, den Zusammenhang zwischen Software-Zuverlässigkeit und Testkonzepten genauer zu untersuchen, da dieser Aspekt einerseits noch in wenigen Modellen Beachtung gefunden hat, andererseits die Anwendung im laufenden Betrieb erleichtern könnte, wenn beides stärker ineinander greift.

Gleichzeitig scheint es von Belang, den Trend zur agilen Software-Entwicklung mit kurzen Software-Entwicklungs-Zyklen zu berücksichtigen. Da hier die in der Testphase zur Verfügung stehende Zeit verhältnismäßig kurz ist, muss gewiss überlegt werden, inwieweit ausreichend Daten gesammelt werden können, um valide Prognosen und Schätzungen erstellen zu können.

Auch andere aktuelle Entwicklungen im Software-Engineering stellen lohnenswerte Untersuchungsgegenstände dar. Inwieweit kann das Konzept der Software-Zuverlässigkeit etwa auf das Maschinen-Lernen – beispielsweise bei der Entwicklung selbstfahrender Autos – angewendet werden?

Schließlich stellt sich die Frage, wie Software-Zuverlässigkeit im Allgemeinen in der Praxis stärker etabliert werden kann. Dazu wären sicherlich mehrere Maßnahmen denkbar und notwendig. Ein erster Schritt dahingehend könnte es wohl sein, die Konzepte für Personen, die keinen technischen oder mathematischen Hintergrund vorweisen, zugänglicher zu machen. In diesem Zusammenhang könnte eine Übersicht über etablierte Modelle und ihre Anwendungsbedingungen hilfreich sein. Gleichzeitig wäre es wohl notwendig zu überlegen, wie die Datenerfassung in der Praxis ohne gravierenden Mehraufwand vonstatten gehen könnte.

Literatur

[1] IEEE Std. 610.12-1990(R2002). *IEEE Standard Glossary of Software Engineering Terminology*. The Institute of Electrical and Electronics Engineers, Inc., New York, September 1990/2002.

[2] ISO 9000:2015. *Qualitätsmanagmentsysteme – Grundlagen und Begriffe*. Austrian Standards Institute, Wien, November 2015.

[3] May Barghout. On parameter estimation of software reliability models. *Communications in Statistics - Simulation and Computation*, 46(2):910–932, 2017.

[4] Fevzi Belli, Matthias Grochtmann, and Oliver Jack. Erprobte Modelle zur Quantifizierung der Software-Zuverlässigkeit. *Informatik Spektrum*, 21:131–140, 1998.

[5] Antonia Bertolino, Breno Miranda, Roberto Pietrantuono, and Stefano Russo. Adaptive Coverage and Operational Profile-based Testing for Reliability Improvement. *Proceedings of the 39th International Conference on software engineering*, pages 541–551, 2017.

[6] Siddartha R Dalal. Software reliability models: A selective survey and new directions. In Hoang Pham, editor, *Handbook of Reliability Engineering*, pages 201–211. Springer Verlag, London, 2003.

[7] Aleksandar Dimov, Senthil Kumar Chandran, and Sasikumar Punnekkat. How do we collect data for software reliability estimation? *International Conference on Computer Systems and Technologies - CompSysTech'10*, pages 155–160, 2010.

[8] Wolfgang Ehrenberger. *Software-Verifikation. Verfahren für den Zuverlässigkeitsnachweis von Software*. Carl Hanser Verlag, München, Wien, 2002.

[9] Felipe Febrero, Coral Calero, and Maria Angeles Moraga. Systematic Mapping Study of Software Reliability Modeling. *Information and Software Technology*, 56:839–849, 2014.

[10] Felipe Febrero, Coral Calero, and Maria Angeles Moraga. Software reliability modeling based on ISO/IEC SQuaRE. *Information and Software Technology*, 70:18–29, 2016.

[11] Phyllis Frankl, Dick Hamlet, Bev Littlewood, and Lorenzo Strigini. Choosing a Testing Method to Deliver Reliability. *Proceedings of the (19th) International Conference on Software Engineering*, pages 68–78, 1997.

[12] Gisela Härtler. *Statistik für Ausfalldaten. Modelle und Methoden für Zuverlässigkeitsuntersuchungen*. Springer Verlag, Berlin Heidelberg, 2016.

[13] Chao-Jung Hsu and Chin-Yu Huang. Optimal Weighted Combinational Models for Software Reliability Estimation and Analysis. *IEEE TRANSACTIONS ON RELIABILITY*, 63(3):731–749, September 2014.

[14] P. K. Kapur, Hoang Pham, Sameer Anand, and Kalpana Yadav. A Unified Approach for Developing Software Reliability Growth Models in the Presence of Imperfect Debugging and Error Generation. *IEEE TRANSACTIONS ON RELIABILITY*, 60(1):331–340, 2011.

[15] Richard Lai and Mohit Garg. A Detailed Study of NHPP Software Reliability Models. *JOURNAL OF SOFTWARE*, 7(6):1296–1306, 2012.

[16] Richard Lai, Mohit Garg, P. K. Kapur, and Shaoying Liu. A Study of When to Release a Software Product from the Perspective of Software Reliability Models. *Journal of Software*, 6(4):651–661, 2011.

[17] Peter Liggesmeyer, Martin Rothfelder, Michael Rettelbach, and Thomas Ackermann. Qualitätssicherung Software-basierter technischer Systeme – Problembereiche und Lösungsansätze. *Informatik-Spektrum*, 21:249–258, 1998.

[18] Nancy R. Mann, Ray E. Schafer, and Nozer D. Singpurwalla. *Methods for Statistical Analysis of Reliability and Life Data.* John Wiley & Sons, New York e.a., 1974.

[19] John D. Musa. A theory of software reliability and its application. *IEEE Transactions on Software Engineering*, 1(3):312–327, 1975.

[20] John D. Musa. *Software Reliability Engineering. More Reliable Software. Faster Development and Testing.* McGraw-Hill, New York e.a., 1999.

[21] John D. Musa, Anthony Iannino, and Kazuhira Okumoto. *Software reliability : measurement, prediction, application.* McGraw-Hill, New York e.a., 1987.

[22] Mitsuru Ohba and Mitsuru Xiao-Mei Chou Ohba. Does imperfect debugging affect software reliability growth? *Proceedings – International Conference on Software Engineering*, 11:237–244, 1989.

[23] Hoang Pham. *Software Reliability.* Springer Verlag, Singapur, 2000.

[24] Hoang Pham and Xuemei Zhang. NHPP software reliability and cost models with testing coverage. *European Journal of Operational Research*, 145:443–454, 2003.

[25] Andrei D. Polyanin. Eqworld: Inverse laplace transforms: Expressions with rational functions.

[26] Andrei D. Polyanin. Eqworld: Laplace transforms: General formulas.

[27] Rakesh Ranaa, Miroslaw Starona, Christian Bergera, Jörgen Hanssona, Martin Nilssonb, Fredrik Törner, Wilhelm Medingc, and Christoffer Höglundd. Selecting software reliability growth models and improving their predictive accuracy using historical projects data. *The Journal of Systems and Software*, 98:59–78, 2014.

[28] Peter Roitzsch. *Analytische Softwarequalitätssicherung in Theorie und Praxis. Der Weg zur Software mit hoher Qualität durch statisches Prüfen dynamisches Testen formales Beweisen.* Monsenstein und Vannerdat, Münster, 2005.

[29] Kevin Taylor-Sakyi. Reliability Testing Strategy. Reliability in Software Engineering, May 2016.