GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

INSTITUT FÜR WIRTSCHAFTSINFORMATIK

PROFESSUR FÜR ANWENDUNGSSYSTEME UND E-BUSINESS

Master Thesis

# Titel: „Software Reliability Engineering im Infotainment“

Dozenten:

Prof. Dr. Jens Grabowski (erster Prüfer)

Prof. Dr. Matthias Schumann (zweiter Prüfer)

Betreuer bei BMW:

Dr. Jürgen Gehring

Dr. Klaus Ries

Beginn der Arbeit: 01.02.2008

Ende der Arbeit: 31.07.2008

Eduard Enríquez Alayo
Agnes-Bernauerstr. 74-A
80687 München
Matrikel-Nr.: 20552025

München, 28.07.2008

## Zusammenfassung

Software Reliability ist ein Kriterium von Softwarequalität. Um Software Reliability messen zu können, setzt man Software Reliability Modelle ein. Hierbei handelt es sich um stochastische Zuverlässigkeitsanalysen, die bisher kaum in der Praxis eingesetzt werden. Zuverlässigkeit ist wichtig, da unzuverlässige Software Kundenunzufriedenheit auslösen und dadurch Kosten verursachen kann. Im Fokus der vorliegenden Arbeit steht die Anwendung von Software Reliability Modellen auf Fehlerdaten eingebetteter Software von unterschiedlich komplexen Infotainment-Geräten von BMW. Hierzu wurden zuerst die Eigenschaften der Fehlerdaten analysiert, daraufhin geeignete Software Reliability Modelle – nämlich Nonhomogeneous Poisson Prozessmodelle – ausgewählt, auf die Daten angewendet und die Ergebnisse interpretiert und verglichen. Ziel der Anwendung der Software Reliability Modelle war es, die aktuelle Zuverlässigkeit zu schätzen, zukünftige Restfehler vorherzusagen, Testendekriterien zu bestimmen und damit wichtige Informationen für das Testmanagement zur Verfügung zu stellen.



## Abstract

Software reliability is a criterion for software quality. In order to measure software reliability, one must utilize software reliability models. This requires the use of stochastic reliability analyses which have hardly been utilized in practice. Reliability is important, as unreliable software can lead to customer dissatisfaction and lead to increased costs. The focus of the following paper will be placed on the use of software reliability models in regard to error-data in embedded software for BMW infotainment-instruments of varying complexity. First, the characteristics of error-data were analysed and appropriate software reliability models – specifically non-homogeneous poisson process models – chosen. These models were then utilized and the results interpreted and compared. The goals of the use of software reliability models were to estimate the current reliability, to predict future remaining errors, to determine test-end criteria and thereby provide important information for test management.

# Inhaltsverzeichnis

**Seite**

# Abbildungsverzeichnis

Seite

# Tabellenverzeichnis

Seite

# Abkürzungsverzeichnis

| | |
|---|---|
| AIC | Akaike's Information Criterion |
| GHU | Große Head Unit |
| HU | Head Unit |
| IEEE | Institute of Electrical and Electronics Engineers |
| IS | Integrationsstufe |
| KHU | Kleine Head Unit |
| LIC | Lower Confidential Interval |
| LLF | Log Likelihood Funktion |
| MHU | Mittlere Head Unit |
| MLE | Maximum Likelihood Estimator |
| MSE | Mean Squared Errors |
| MTR | Modul Test Review |
| MVF | Mean Value Function |
| OEM | Original Equipment Manufacturer |
| PRR | Predictive Ratio Risk |
| SAAR | Software Anforderung Review |
| SAER | Software Architekturentwurf Review |
| SATR | Software-Abnahme Review |
| SER | Steuergerät Entwurf Review |
| SGTR | Steuergeräte-Abnahmetest Review |
| SMERFS^3 | Statistical Modeling and Estimation of Reliability Functions for Software: Software, Hardward und Systems |
| SMR | Software Modulentwurf Review |
| SR | Software Reliability |
| SRE | Software Reliability Engineering |
| SRGM | Software Reliability Growth Model |
| SRM | Software Reliability Model |
| SSE | Sum of Squared Errors |
| TEL | Steuergerät für Telefonie |
| UCI | Upper Confidential Interval |
| ZE | Zeiteinheit |

# 1 Einleitung

## *1.1 Software Reliability bei eingebetteter Software am Beispiel des Infotainment*

Alltag und Wirtschaft sind durch Software geprägt. Viele Funktionen in Haushaltsgeräten, Flugzeugen und Automobilen – um nur einige zu nennen – werden heutzutage durch Software realisiert. Damit hat Software eine hohe wirtschaftliche und technische Bedeutung. Dabei spielt vor allem eingebettete Software eine wichtige Rolle. Eingebettete Software ist in einen Hardwaresystemkontext eingebunden und Teil eines eingebetteten Systems. Eingebettete Systeme sind informationsverarbeitende Systeme, die aus Hardware- und Softwarekomponenten bestehen. Sie sind integraler Bestandteil komplexer mikroelektronischer bzw. mechatronischer Systeme. Sie sind häufig in größere, teilweise heterogene Umgebungen eingefügt. Eingebettete Systeme führen spezielle Funktionen innerhalb eines Gesamtsystems aus (vgl. Salzmann; Stauner 2005, Kap. 4 „Domäne Infotainment“). Auch Infotainment-Geräte in Automobilen sind eingebettete Systeme.

Viele technische Systeme, die Software beinhalten, erfordern eine hohe Zuverlässigkeit (vgl. Liggesmeyer 2005b, S. 205). Zuverlässigkeit ist ein Kriterium von Softwarequalität. Die Sicherung der Softwarequalität ist eine wichtige Herausforderung bei der Softwareproduktion. Für den Zuverlässigkeitsbegriff wird im Englischen der Begriff „Reliability“ verwendet (vgl. Liggesmeyer 2002, S. 428): „Software reliability is the most important aspect of quality to the user because it quantifies how well the software product will function with respect to his or her needs (Musa 1999, S. 11). Software Reliability ist definiert als die Wahrscheinlichkeit, dass eine Software fehlerwirkungsfrei für eine bestimmte Zeitperiode in einer spezifischen Umgebung arbeitet (vgl. Lyu 1996, S. 5). Um Software Reliability messen zu können, setzt man Software Reliability Modelle (SRM) ein. Bei der Identifizierung der Produktrisiken können Zuverlässigkeitsanalysen und -prognosen helfen (vgl. Rinke et al. 2006). Software Reliability Engineering (SRE) ist die Anwendung von statistischen bzw. stochastischen Methoden auf Daten, die während der Softwareentwicklung erhoben wurden.

Die Messung der Software Reliability dient dazu, einen Aspekt der Qualität von Software zu messen. Zuverlässigkeit ist wichtig, da unzuverlässige eingebettete

Software Kundenunzufriedenheit auslösen und dadurch Kosten verursachen kann. Messungen zur Software Reliability können helfen, Test- und Fehlerkosten in einer Balance zu halten, da mit ihnen beispielsweise die aktuelle Zuverlässigkeit geschätzt und zukünftige Restfehler vorhergesagt werden können (vgl. Spillner et al. 2006, S. 71f.).

## *1.2 Fragestellungen und Ziele der Arbeit*

In der vorliegenden Arbeit steht die Anwendung von Software Reliability Modellen auf Fehlerdaten eingebetteter Software von Infotainment-Geräten von BMW im Fokus. Gerade bei eingebetteter Software sollten dynamisches Testen und quantifizierende Analysen als Maßnahmen der Qualitätssicherung kombiniert werden (vgl. Liggesmeyer 2005b, S. 213). Konkret geht es darum, geeignete SRE-Modelle auszuwählen, um sie auf diese Daten anzuwenden. Die Modelle werden anhand ihrer Vorhersagegüte bewertet und verglichen. Die Modelle liefern Einschätzungen und Vorhersagen über die Zuverlässigkeit der getesteten Komponenten. Die gewonnenen Ergebnisse werden zu managementtauglichen Metriken verdichtet um Testkampagnen in Zukunft besser planen zu können.

Dabei kann man sagen, dass es sich bei der Aufgabenstellung um einen innovativen und schwer zu bearbeitenden Bereich handelt, da stochastische Zuverlässigkeitsanalysen bisher „keine weite Verbreitung in der Praxis der Software-Entwicklung“ besitzen (Liggesmeyer 2002, S. 474).

## *1.3 Überblick über die Arbeit*

Kapitel 2 „Softwarequalität und Qualitätssicherungsmaßnahmen“ definiert in einem ersten Schritt Softwarequalität, zeigt Kriterien von Softwarequalität auf und erläutert Qualitätssicherung von Software (Kap. 2.1). In einem zweiten Schritt wird ein Überblick über Maßnahmen der Qualitätssicherung von eingebetteter Software gegeben. Hier wird auch auf Testendekriterien eingegangen (Kap. 2.2). Dieses Kapitel ist notwendig, da es aufzeigt, dass Software Reliability ein Qualitätskriterium von Software ist und da es die stochastische Zuverlässigkeitsanalyse – wie sie in der vorliegenden Arbeit eingesetzt wird – in den verschiedenen Prüftechniken verortet.

Das Kapitel 3 „Messung von Software Reliability mit Software Reliability Modellen“ widmet sich dem Thema Software Reliability und Software Reliability Modelle. Zuerst

werden zentrale Begriffe, Ziele und Indikatoren geklärt (Kap. 3.1). Danach werden verschiedene Software Reliability Modelle vorgestellt (Kap. 3.2). Ausführlich wird dabei auf die Nonhomogenous Poisson Prozessmodelle eingegangen, da diese bei der empirischen Analyse in der vorliegenden Arbeit eingesetzt werden. In Kapitel 3.3 werden die theoretischen Kriterien für die Modellauswahl beschrieben, die konkrete Auswahl für die empirische Analyse erfolgt nach der Datenvoranalyse in Kapitel 4.5. Dieses Kapitel bildet die theoretische Basis um eine empirische Analyse durchführen zu können.

In Kapitel 4 „Anwendungsfeld, Datenbasis und Design der empirischen Analyse“ geht es um das Design der empirischen Analyse. Zuerst wird das Anwendungsfeld beschrieben, nämlich Infotainment in der Automobilbranche (Kap. 4.1). Darauf folgend werden Ziele und Fragestellungen der empirischen Analyse vorgestellt (Kap. 4.2). Kapitel 4.3 gibt die Analyse der Datenausgangsbasis bei BMW wider. In Kapitel 4.4 werden die Testinstanzen modelliert und erste Testmetriken über alle vorhandenen Daten hinweg im Durchschnitt erstellt. Kapitel 4.5 erläutert die konkrete Modellauswahl für die empirische Analyse aufgrund theoriebasierter Kriterien, die in Kapitel 3.3 beschrieben wurden. Danach wird in Kapitel 4.6 das in der vorliegenden Arbeit eingesetzte Werkzeug des Software Reliability Engineerings erläutert.

In Kapitel 5 „Ergebnisse der empirischen Analyse“ werden die ausgewählten Software Reliability Modelle für vier Beispiele eingesetzt, Vorhersagen und Schätzungen der Software Reliability berechnet sowie spezifische Testmetriken erstellt (Kap. 5.1). Für jedes Beispiel werden die Gesamtfehler und Restfehler mit den Software Reliability Modellen geschätzt und mit den realen Daten verglichen. Mit diesem Vergleich ist es möglich, die Güte der Passung der Modelle auf die spezifischen Daten zu bestimmen. Kapitel 5.2 widmet sich dem Vergleich und der Interpretation der Ergebnisse unter anderem in Bezug auf die Passung der Modelle sowie in Bezug auf die Schätzung von Gesamtfehlern und Restfehlern.

Die Arbeit schließt mit Kapitel 6 mit einer Zusammenfassung. Hier werden Ergebnisse zum Vorgehen bei der Anwendung von Software Reliability Modellen zusammengefasst und ein Ausblick auf den Einsatz von Software Reliability Modellen zur Unterstützung des Testmanagements gegeben.

# 2 Softwarequalität und Qualitätssicherungsmaßnahmen

In diesem Kapitel geht es darum, in einem ersten Schritt Softwarequalität zu definieren, Kriterien von Softwarequalität aufzuzeigen sowie Qualitätssicherung von Software zu erläutern (Kap. 2.1). In einem zweiten Schritt wird ein Überblick über Maßnahmen der Qualitätssicherung von Software gegeben (Kap. 2.2). Hier wird auch auf Testendekriterien und Metriken eingegangen. Außerdem wird die stochastische Zuverlässigkeitsanalyse – wie sie in der vorliegenden Arbeit eingesetzt wird – innerhalb der verschiedenen Prüftechniken verortet.

## *2.1 Qualitätssicherung von Software*

### 2.1.1 Definition von Qualität

Qualität ist die Übereinstimmung mit bestimmten vorher festgelegten Anforderungen. Dabei können diese Qualitätsanforderungen aus Hersteller- und Nutzersicht unterschiedlich aussehen. Nach der Norm DIN 55350-11 ist Qualität die „Gesamtheit von Merkmalen (und Merkmalswerten) einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen“ (vgl. Stahlknecht; Hasenkamp 2005, S. 309). Um Qualität beurteilen zu können, gibt es drei Schritte. In einem ersten Schritt stellt man qualitative Beurteilungskriterien auf. Danach entwickelt man Messgrößen zur quantitativen Bewertung der Qualitätsmerkmale. Im dritten Schritt legt man Maßstäbe fest, um die Erfüllung der Qualitätsmerkmale anhand der Messgrößenwerte zu beurteilen (vgl. ebd.). Auch Software Reliability Engineering hilft dabei, zu überprüfen, inwiefern vorher festgelegte Kriterien erfüllt werden (vgl. Kap. 3).

### 2.1.2 Qualitätssicherung und Qualitätsmanagement

Die DIN-ISO 9126 definiert: „Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen“. Software Qualität ist multikausal, d.h. dass es nicht das eine Kriterium gibt, mit dem man sie in direkter Weise und vor allem quantitativ verbindet, sondern dass es eine ganze Reihe vielschichtiger Kriterien gibt (vgl. Hoffmann 2008, S. 6). Qualität von Software ist sehr wichtig, da mangelnde Qualität viele Kosten verursachen kann (z.B. in der Wartung, s.u., aber auch durch folgenschwere Fehler, vgl. hierzu die Beispiele in Thaller 1990, S. 19f.). Darüber hinaus kann mangelnde Qualität zur Verringerung der Anwenderakzeptanz und zu

Kundenverlust führen. Auch können Kosten durch Produkthaftung und Sicherheitsprobleme entstehen. Qualität ist damit ein entscheidender Wettbewerbsfaktor (vgl. Thaller 1990, S. 9). Softwarequalität kann nur dann gesichert werden, wenn während des gesamten Entwicklungsprozesses Qualitätssicherungsmaßnahmen eingesetzt werden (vgl. Stahlknecht; Hasenkamp 2005, S. 313). Dabei unterscheidet sich die traditionelle Qualitätssicherung von der Qualitätssicherung von Software aufgrund der Eigenschaften von Software (wie: Software ist immateriell; Software altert nicht; es gibt keine Ersatzteile für Software; Kopie ist identisch mit Original; keine industrielle Produktion; Softwarequalität ist schwer quantifizierbar) (vgl. Thaller 1990, S. 13ff.; Spillner; Linz 2005, S. 6). Eine Ausweitung der Qualitätssicherung ist das Qualitätsmanagement, mit dem sich z.B. die Normenreihe DIN EN ISO 9000ff. beschäftigt, die weniger Produkte als Prozesse zertifiziert (vgl. Stahlknecht; Hasenkamp 2005). Zum Qualitätsmanagement gehören die Qualitätsplanung, -lenkung, -sicherung und -verbesserung (vgl. ebd., S. 314). Ein Qualitätssicherungsprozess, der ausführliches Softwaretesten mit einschließt, kann zu einer qualitativ hochwertigen Software führen. Das Testen von Software wird zunehmend auch in den größeren Rahmen von Verifikation und Validation gestellt (s.u.).

### 2.1.3 Softwarequalität und ihre Kriterien

Ziel des Testens von Software ist die Steigerung der Softwarequalität. Dies geschieht durch die Auffindung von Fehlern und deren Beseitigung durch das Debugging (vgl. Spillner; Linz 2005, S. 11). Die Testfälle sollten dabei so ausgewählt werden, dass sie der späteren Benutzung der Software nahe kommen. Wenn dies der Fall ist, entspricht die nachgewiesene Qualität der Software während des Testens der zu erwartenden Qualität während der späteren Benutzung (vgl. ebd.).

Softwarequalität beschreibt die Erfüllung von Anforderungen an Software. Anforderungen an Software sind 1) die Erfüllung der Anforderungen des Fachentwurfs, 2) leichte Ergänzbarkeit oder Änderbarkeit und 3) Benutzerfreundlichkeit (Softwareergonomie) (vgl. Stahlknecht; Hasenkamp 2005, S. 213). Liggesmeyer (2002, S. 425) unterscheidet die Qualitätseigenschaften Sicherheit, Zuverlässigkeit und Verfügbarkeit. Diese Anforderungen sind Qualitätskriterien. Verschiedene nationale und internationale Normen befassen sich mit Softwarequalität (DIN/ISO) (vgl. Stahlknecht; Hasenkamp 2005, S. 309). So legt die DIN 66272 bzw. ISO-Norm 9126 als Qualitätsmerkmale von Software folgende Punkte fest: Funktionalität,

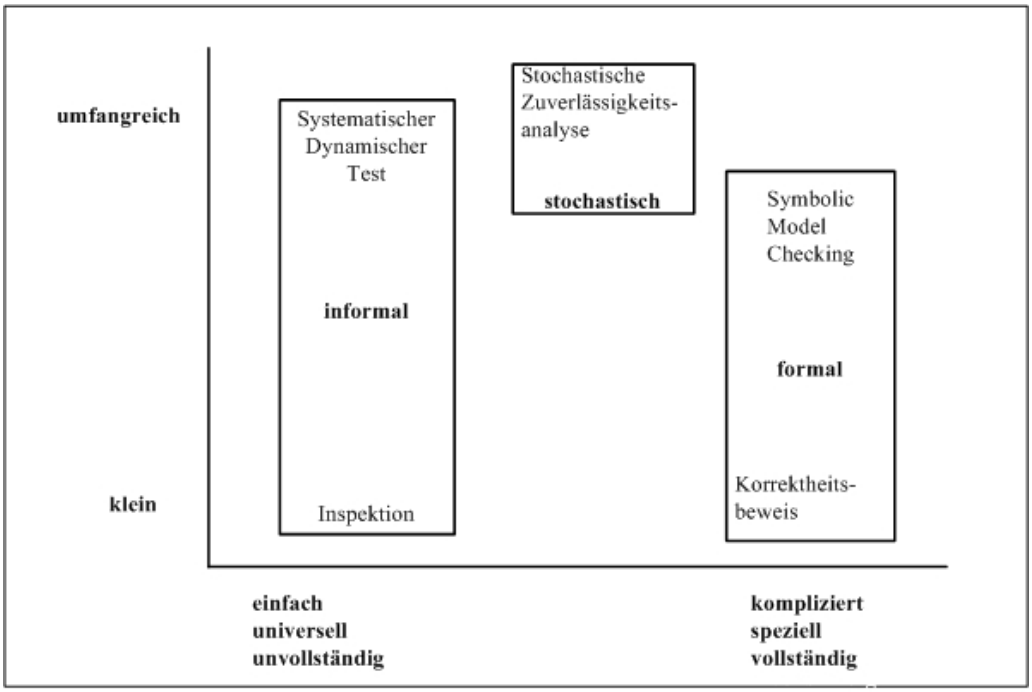
Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit (vgl. ebd., S. 310; ausführlich siehe Spillner; Linz 2005, S. 11). Zuverlässigkeit ist dabei ein zentrales Qualitätskriterium (vgl. Elbel; Mäckel o.Jg.), es wird ausführlich als Software Reliability in Kapitel 3 besprochen. Alle diese Qualitätsmerkmale muss man beim Testen berücksichtigen, um die Gesamtqualität einer Software bewerten zu können. Vorab muss festgelegt werden, welches Qualitätsniveau in den einzelnen Qualitätsmerkmalen erreicht werden soll – dies wird dann beim Testen überprüft. Thaller weist darauf hin, dass sich die Qualitätsattribute jedoch auch widersprechen können, dies muss man bei ihrer Festsetzung bedenken. So können sich beispielsweise Effizienz und Übertragbarkeit widersprechen (vgl. Thaller 1990, S. 88). Es muss also im Vorhinein festgelegt werden, welche Qualitätseigenschaften welche Priorität haben (vgl. Spillner; Linz 2005, S. 13).

## *2.2 Maßnahmen der Qualitätssicherung von eingebetteter Software*

### 2.2.1 Überblick über verschiedene Prüftechniken

Es gibt verschiedene Prüftechniken oder Methoden bei der Qualitätssicherung von eingebetteter Software, wie die folgende Abbildung 2.2-1 zeigt.

**Abbildung 2.2-1 Einordnung von Prüftechniken**




Quelle: Liggesmeyer 2002, S. 473; Liggesmeyer 2005a, S. 7

Zu den informalen Techniken gehören neben dem dynamischen Test auch die Inspektions- und Review-Techniken (statischer Test). Das (dynamische) Testen ist dabei „die wichtigste und häufigste Maßnahme der Qualitätssicherung.“ (Mellis 2001, S. 471). Unter Testen versteht man die Prüfung von codierten Programmen auf korrekte Formulierung und Ausführung. Testen ist ein analytisches Verfahren und gehört dort zu den so genannten dynamischen Prüfungen, um Fehler aufzufinden (vgl. Stahlknecht; Hasenkamp 2005, S. 288; vgl. Rätzmann 2004, S. 31). Dabei bedeutet ein Fehler die Nichterfüllung einer Anforderung (s.o.), eine Abweichung zwischen dem Istverhalten und dem Sollverhalten (vgl. Spillner; Linz 2005, S. 7). Das Testen kann prinzipiell nur die Anwesenheit von Programmierfehlern zeigen, jedoch nicht die völlige Abwesenheit von diesen (vgl. Stahlknecht; Hasenkamp 2005, S. 289). Man kann u.a. im Test nicht alle Anwendungssituationen nachbilden sowie nicht die Fehlerursache zeigen (vgl. Frühauf; Ludewig; Sandmayr 2007, S. 23). Der statische Test wird auch als statische Analyse bezeichnet (vgl. Spillner; Linz 2005, S. 77). Bei Inspektions- und Review-Techniken werden keine Werkzeuge im Vergleich mit den anderen statischen Techniken – wie Stilanalyse, Grafiken und Tabellen, Slicing und Datenflussanomalieanalyse – verwendet (vgl. Liggesmeyer 2005b, S. 211). Der Nachteil der informalen Techniken ist, dass die erzeugten Ergebnisse unvollständig sind im Gegensatz zu den formalen, dafür sind sie jedoch einfach und universell anwendbar. Diese informalen Techniken schließen Restfehler in der Software nicht aus, daher sind sie oftmals für sicherheitskritische Softwareanwendungsbereiche nicht ausreichend.

Formale und stochastische Prüftechniken sind in der Software-Entwicklung nicht so verbreitet (vgl. Liggesmeyer 2002, S. 473; Liggesmeyer 2005a, S. 6f.). Das Testen stellt die Grundlage auch für die Anwendung stochastischer Analysen dar, da in den Testprozessen die erforderlichen Daten gewonnen werden. Der Nachteil der formalen Techniken ist, dass formale Techniken kompliziert sind und nur anwendbar auf bestimmte Bereiche. Stochastische Techniken wiederum liefern Ergebnisse, die quantifiziert werden, damit können sie zur Bestimmung der Restfehler dienen. Sie sind sozusagen ein Kompromiss zwischen formalen und informalen Techniken.

Die stochastische Zuverlässigkeitsanalyse als eine Prüftechnik steht im Fokus der vorliegenden Arbeit, daher wird sie ausführlich in Kapitel 3 beschrieben.

### 2.2.2 Anforderungen an die Prüfung eingebetteter Software

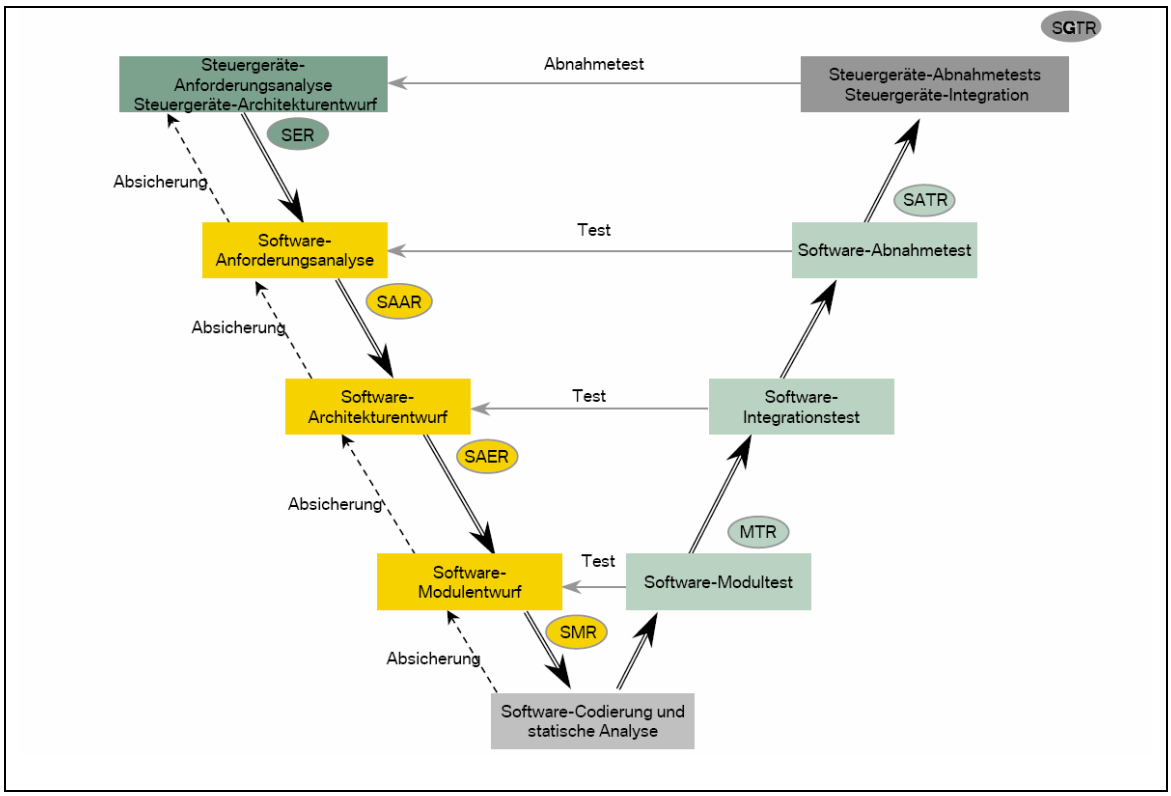
„Software Engineering für eingebettete Systeme ist eine wichtige Disziplin, die zwischen dem „klassischen“ Software Engineering und dem System Engineering angesiedelt ist.“ (Liggesmeyer 2005a, S. 1). Es werden Techniken, Methoden und Prozessen aus beiden genannten Disziplinen angepasst und ergänzt. Das heißt, die oben genannten Prüftechniken sind auch für die Prüfung eingebetteter Software relevant. Bei eingebetteter Software kommen jedoch noch einige Anforderungen hinzu, wie z.B. nicht-funktionale Qualitätsanforderungen (z.B. safety) sowie bei sicherheitskritischen Systemen Zulassungsprozesse (vgl. Liggesmeyer 2005a, S. 8). Der dynamische Test – wie Pfadüberdeckungs- und Ausweisungsüberdeckungstest – und stochastische Verfahren ermöglichen eine wichtige quantitative Aussage zur Zuverlässigkeit für eingebettete Software. Es ist auch möglich, die kombinierte Verwendung von statischer Analyse und formale Techniken zur Prüfung eingebetteter Software zu wählen, um so deren jeweilige Schwächen zu vermeiden (vgl. Liggesmeyer 2005b, S. 213). Die Quantifizierung der stochastischen Zuverlässigkeitsanalyse ermöglicht die Aufdeckung von Restrisiken, die durch Software verursacht werden (vgl. Liggesmeyer 2005b, S. 221) (vgl. ausführlich Kap. 3). Zusammenfassend lässt sich sagen, dass gerade bei der Prüfung eingebetteter Software Risiken, die von Restfehlern ausgehen, quantifiziert werden sollten, das heißt, Software Reliability Modelle zusätzlich zu dynamischen Tests eingesetzt werden sollten: „Die Quantifizierung der Zuverlässigkeit eingebetteter Software ist wichtig im Rahmen der Ermittlung der durch die Software verursachten Restrisiken.“ (Liggesmeyer 2005b, S. 221).

### 2.2.3 Testen innerhalb des Softwareentwicklungsprozesses

Je nach Phase des Lebenszyklus von Software kann man verschiedene Arten von Tests unterscheiden. Spillner und Linz (2005, S. 40) beschreiben das Testen am V-Modell nach Boehm (zur Weiterentwicklung und grafischen Darstellung hin zum W-Modell vgl. Spillner et al. 2006). In diesem Modell wird das Testen als gleichwertig zu Entwicklung und Programmierung dargestellt, daher hat das Testen einen eigenen gleichberechtigten „Ast“. Im Gegensatz zum Wasserfall-Modell, bei dem Entwicklungsschritten Tests nur zugeordnet werden, scheint das V-Modell für das Testen noch vielversprechender. Das „V“ weist darüber hinaus auf Verifikation und Validation hin (vgl. Spillner; Linz 2005, S. 41). Verifikation überprüft ob die

Umsetzung der Anforderung einer Phase in das Ergebnis der Phase fehlerfrei ist und Validation überprüft, ob die entsprechenden Anforderungen an das Produkt erfüllt wurden (vgl. Liggesmeyer 2005b, S. 222). Im Bezug auf das „V“ Modell wird bei BMW mit diesem Prinzip entwickelt (siehe folgende Abb. 2.2-2).

**Abbildung 2.2-2 Verfahrensmodell der Software Entwicklung bei BMW**




Quelle: Salzmann; Stauner 2005, Kap. 8 „Qualitätssicherung und Test“

Man beginnt mit der Anforderungsanalyse und dem Architekturentwurf der Steuergeräte, worauf ein Steuergerät Entwurf Review (SER) folgt. Der nächste Schritt ist die Software-Anforderungsanalyse gefolgt von einem Software Anforderung Review (SAAR), dann kommt der Software-Architekturentwurf mit einem Software Architekturentwurf Review (SAER). Daraufhin folgt der Software-Modulentwurf mit einem Software Modulentwurf Review (SMR). Abschließend wird die Software codiert und eine statische Analyse durchgeführt. Die Absicherung in dem linken Ast ist der Nachweis, der am Ende einer Entwicklungsphase eines Systems oder eine Komponente sicherstellt, dass die an diese Entwicklungsphase gestellten Anforderungen erfüllt werden. Der rechte Ast ordnet jedem Spezifikationsentwurf eine korrespondierende Teststufe zu. Die Teststufen sind der Software-Modultest, darauf folgend ein Modul Test Review (MTR), der Software-Integrationstest, der Software-Abnahmetest und ein Software-Abnahme Review (SATR). Es folgt der Abnahmetest und die Integration der

Steuergeräte, gefolgt vom Steuergeräte-Abnahmetest Review (SGTR). Die Abnahme ist der Nachweis, der während oder am Ende einer Entwicklungsphase eines Systems oder eine Komponente sicherstellt, dass die an diese Entwicklungsphase gestellten Anforderungen erfüllt werden (vgl. Salzmann; Stauner 2005, Kap. 8 „Qualitätssicherung und Test“). Die Daten, die den Analysen in der vorliegenden Arbeit zugrunde liegen, entstammen Abnahmetests.

### 2.2.4 Testendekriterien und Testmetriken

#### *2.2.4.1 Wirtschaftlichkeit beim Testen*

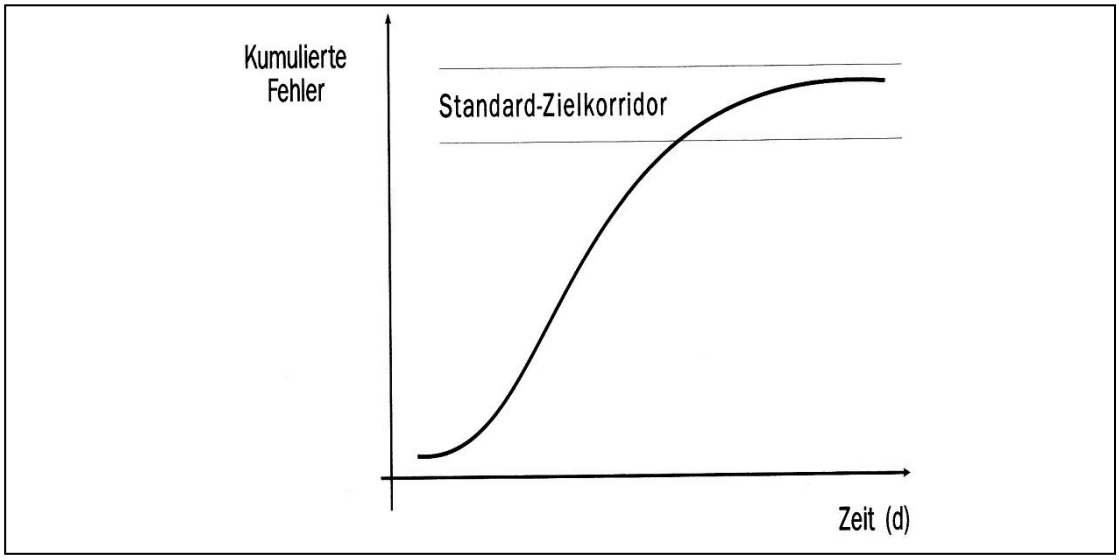
Die Testintensität und der Testumfang sind in Abhängigkeit vom Risiko festzulegen (vgl. Spillner; Linz 2005, S. 16). Testmanager müssen auch mit dem Problem umgehen, dass oft eine Testfallexplosion auftritt bei gleichzeitig beschränkten Ressourcen (vgl. ebd., S. 17). Das Testteam muss also vorneweg auch die Testaufwände schätzen und die Wirtschaftlichkeit bedenken. Der Aufwand für Testen ist hoch, er beträgt 30-50% der gesamten Entwicklungskosten (vgl. Thaller 1990, S. 223). Allerdings sind Fehler, die erst nach der Auslieferung gefunden werden, um ein Vielfaches teurer. Schätzt man die Kosten für die Beseitigung eines Fehlers in der Spezifikationsphase mit „1“, steigt dieser Wert auf „100“ nach der Auslieferung der Software an den Kunden. Außerdem verursacht ein Fehler desto mehr Kosten je mehr Zeit zwischen Entstehung des Fehlers und Entdeckung verstreicht (vgl. Mellis 2001, S. 472). Es gibt verschiedene Arten von Kosten, die durch nicht entdeckte Fehler entstehen können. Direkte Fehlerkosten sind Kosten, die beim Kunden durch Fehlerwirkungen entstehen (z.B. Datenverlust). Indirekte Kosten können dadurch aufkommen, weil der Kunde unzufrieden mit dem Produkt ist (z.B. Imageverlust, erhöhter Aufwand für Support). Als dritte Form der Fehlerkosten sind die Fehlerkorrekturkosten zu nennen, die anfallen, um den Fehler zu beheben und z.B. eine erneute Auslieferung zur Folge haben (vgl. Spillner; Linz 2005, S. 178).

Allerdings ist es nicht sinnvoll, so extensiv vor Auslieferung zu testen, dass (fast) alle Fehler gefunden werden, dafür wäre der zeitliche Aufwand zu hoch. Die Nullfehlerqualität ist somit meist kaum wirtschaftlich vertretbar. Es gilt daher: „Das Ziel des Testens kann also nur lauten, mit einem vertretbaren und dem Zweck der Software angepaßten Test ein Maximum der in der Software vorhandenen Fehler zu finden.“ (vgl. Thaller 1990, S. 223).

### *2.2.4.2 Testendekriterien*

Besonders schwierig ist somit die Frage zu beantworten, wann das Testen zu beenden ist (vgl. Thaller 1990, S. 260). Mittels statistischer Methoden kann man beispielsweise die Fehlermeldungen kumuliert gegen die verstrichene Zeit in ein Diagramm eintragen (siehe folgende Abb. 2.2-3, ausführlich Kap. 3). Hat man die Grenze der erwarteten Fehler erreicht bzw. erfolgt eine Sättigung (vgl. Kap. 3), kann man das Testen beenden (vgl. Thaller 1990, S. 262f.).

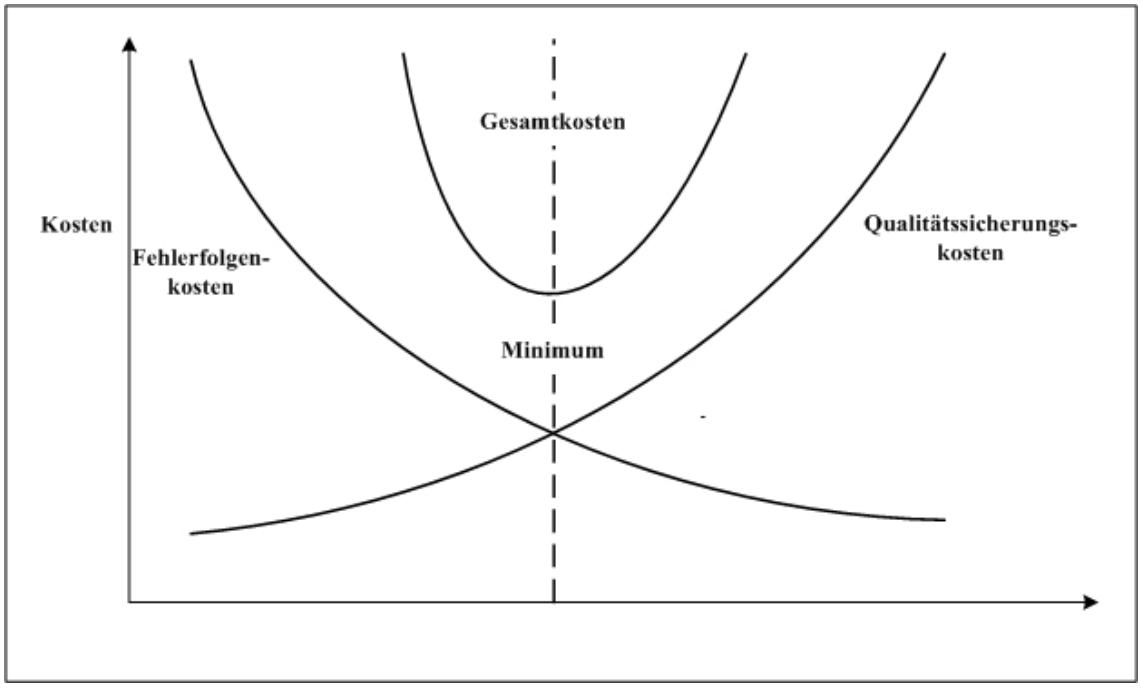
**Abbildung 2.2-3 Fehlerverfolgung bei Software**




Quelle: Thaller 1990, S. 262

Testendekriterien dienen dazu, eine Balance zwischen Test- und Fehlerkosten zu finden (Spillner et al. 2006, S. 71f.). Die folgende Abbildung 2.2-4 zeigt dies auf. Betrachtet man die Kurve der Fehlerfolgekosten zeigt sich, dass je länger man testet, die Fehlerfolgekosten sinken, da weniger Fehler in der Software verbleiben und Kosten produzieren. Die Kurve der Qualitätssicherungskosten zeigt die Kosten des Testens an. Sie steigt im Verlauf des Testens. Folglich geht es darum, das optimale und wirtschaftliche Ziel der beiden zu erreichen, so dass die Gesamtkosten, die sich aus beiden Kurven bilden, so minimal wie möglich sind, also weder zu viel noch zu wenig Aufwand in die Softwarequalität zu stecken (vgl. Schneider 2007, S. 16).

**Abbildung 2.2-4 Qualitätskosten und Risiko (nach Juran 1988)**




Quelle: Spillner et al. 2006, S. 72

Die Faktoren wie Zeit und Kosten setzen oft in der Praxis fest, das Testen zu beenden und haben den Abbruch der Testaktivitäten zur Folge, obgleich das Testen im Ganzen Einsparungen durch Aufdeckung von Fehlerwirkungen und Entfernung der Fehlerzustände in der Software verursacht. Höhere Kosten für die Unternehmen bewirken die nicht entdeckten Fehlerzustände (vgl. Spillner et al. 2006, S. 16). Um Testendekriterien darzustellen setzt man Testmetriken ein.

### *2.2.4.3 Auswahl und Art von Testmetriken*

Liggesmeyer (2002, S. 212) sagt, dass der Begriff „Metrik“ falsch in der Software-Technik verwendet wird, da dieser in der Mathematik besser geeignet ist. „Das Messen […] ist in diesem Sinne als eine Maßbestimmung zu verstehen, da Objekten […] (z.B. einer Systemkomponente) Messwerte zugeordnet werden“. In der Arbeit wird der „Metrik“ Begriff verwendet aber gemeint sind „Maße“.

Einfach, reproduzierbar und nachvollziehbar sind Metriken zu definieren, die erst einmal Hypothesen sind und validiert werden müssen. Zudem sollen die Metriken eine quantifizierte und objektive Grundlage für Entscheidungen zur Verfügung stellen. Es ist zu definieren, welche Ziele durch Messungen verfolgt werden sollen (vgl. Spillner et al. 2006, S. 100f. und 220). Der IEEE Standard 1061 definiert: „Eine Softwaremetrik ist allgemein eine Funktion, die eine Softwareeinheit in einen Zahlenwert abbildet. Dieser

Wert ist interpretierbar als der Erfüllungsgrad eines Qualitätszieles für die Softwareeinheit“ (Schneider 2007, S. 54). Methoden wie Faktor-Kriterien-Metriken (FCM) und Goal-Question-Metric (GQM) helfen einer klaren Darstellung der Ziele, die erreicht werden sollen. Zudem sind wichtige Skalen für einsetzbare Metriken in Schneider (2007, S. 57) und in Spillner et al. (2006, S. 223) gut erläutert. Testmetriken sind unterschiedlich je nach den betrachteten Messobjekten. Sie können wie folgt unterteilt werden: testfallbasierte-, testbasis- und testobjektbasierte-, fehlerbasierte– und kosten- bzw. aufwandsbasierte Metriken. Zu den fehlerbasierten Metriken gehören die Anzahl der Fehler nach Kritikalität (Fehlerschwere), Anzahl der Fehler im Zeitverlauf oder nach Testintensität (Fehlertrend) sowie nach Status (Fehlerbehebungsfortschritt) usw. (vgl. Spillner et al. 2006, S. 100f. und 220ff.). Wurde eine Metrik mehrmals mit Erfolg eingesetzt, so ist sie ein Beleg dafür, das sie brauchbar ist. Hierfür sind die Metriken zu dokumentieren (vgl. Spillner et al. 2006, S. 223f.).

In der vorliegenden Arbeit liegt der Fokus auf fehlerbasierten Metriken, da die nach dem Test verbliebenen Fehler geschätzt werden müssen, um die erwartete Zuverlässigkeit (Reliability) eines Systems zu bestimmen.

## *2.3 Zusammenfassung*

Nach der Definition von Softwarequalität, ihren Kriterien und Qualitätssicherungsmaßnahmen (siehe Kap. 2.1) wurden die verschienen Prüftechniken (siehe Kap. 2.2.1) sowie die Anforderung an die Prüfung eingebetteter Software dargestellt (siehe Kap. 2.2.2.). Wie das Testen innerhalb des Softwareentwicklungsprozesses bei BMW durchgeführt wird, wurde in Kapitel 2.2.3 kurz beantwortet. Außerdem wurden die Testendekriterien in Bezug auf Kosten und Risiken in Kapitel 2.2.4.1 kurz erläutert sowie die verschiedenen Metriken in Kapitel 2.2.4.2 beschrieben.

# 3 Messung von Software Reliability mit Software Reliability Modellen

Dieses Kapitel widmet sich dem Thema Software Reliability und Software Reliability Modelle. Zuerst werden zentrale Begriffe, Ziele und Indikatoren, die wichtig für das weitere Verständnis sind, geklärt (Kap. 3.1). Kapitel 3.2 widmet sich Software Reliability Modellen. Diese werden zuerst grundlegend geklärt um dann eine Auswahl der für diese Masterarbeit relevanten Modelle treffen zu können. Aufgrund der bei BMW vorliegenden Daten und ihrer Merkmale sowie der Ziele der Analyse (vgl. Kap. 4) kommen hierfür Nonhomogenous Poission Prozessmodelle (NHPP) in Betracht. Aus den NHPP-Modellen wurden zwei Modelle (Goel-Okumoto und Yamada-Delayed-S-shaped Modell) ausgewählt, da sie zum Einen erprobte Modelle darstellen und zum Anderen das Yamada-Delayed-S-shaped Modell auch die Einschwingphase beim Testen berücksichtigt. In Kapitel 3.3 wird beschrieben, wie eine Auswahl der Software Reliability Modelle nach bestimmten Kriterien erfolgt. Abschließend folgt eine Zusammenfassung des Kapitels.

## *3.1 Software Reliability und Software Reliability Engineering*

### 3.1.1 Definition von Software Reliability und Software Reliability Engineering

Für den Zuverlässigkeitsbegriff nach DIN 40041/DIN 4004190 wird im Englischen der Begriff „Reliability“ verwendet (vgl. Liggesmeyer 2002, S. 428) (vgl. auch Kap. 2.1.3). Software Reliability (SR) wird von den verschiedenen Autoren ähnlich definiert, wie im Folgenden aufgezeigt wird.

SR ist definiert als die Wahrscheinlichkeit, dass eine Software fehlerwirkungsfrei für eine bestimmte Zeitperiode in einer spezifischen Umgebung arbeitet (vgl. Lyu 1996, S. 5). Eine Fehlerwirkung bezeichnet hier „the inability of performing an intended task specified by the requirement.“ (Xie 1991, S. 5). Die SR wird also erhöht, wenn Fehlerzustände, die Fehlerwirkungen produziert haben, entfernt werden (vgl. Musa 1999, S. 265). In Spillner; Linz (2005, S. 260) wird Zuverlässigkeit definiert als „eine Menge von Merkmalen, die sich auf die Fähigkeit der Software beziehen, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren“. Leitch betont, dass Reliablity als eine Wahrscheinlichkeit gemessen wird: „The reliability of a product is the measure of its ability to perform its function, when

required, for a specified time, in a particular environment. It is measured as a probability" (Leitch 1995, S. 2). Nach Ansicht von Huang, Lyu und Kuo könnte SR die wichtigste Qualitätseigenschaft von Software-Anwendungen sein, da diese Fehlerwirkungen während des Software-Entwicklungs-Prozesses misst. (vgl. Huang; Lyu; Kuo 2003, S. 261). Das zeitlich verteilte Eintreten von Fehlerwirkungen, die oft durch Fehlerzustände der eingebetteten Software verursacht werden, bestimmt die Zuverlässigkeit (vgl. Liggesmeyer 2005b, S. 221).

Software Reliability Engineering (SRE) wiederum ist die Anwendung von statistischen bzw. stochastischen Methoden auf Daten, die während der Softwareentwicklung erhoben wurden (vgl. Farr 2002a).

Eine sehr wichtige Phase der Softwareentwicklung ist das Testen, da die höchsten Kosten im Entwicklungsprozess damit assoziiert sind (vgl. Kap. 2.2.4.1). Darüber hinaus sind die Kosten der Software-Wartung, welche ebenfalls recht hoch sind, normalerweise eine Konsequenz der Unzuverlässigkeit der Software (vgl. Xie 1991, S. 2). Die wichtigsten Fragen, die somit vom Software Reliability Engineering beantwortet werden können, sind die Folgenden: Wie gut ist die Software (wie fehlerfrei)? Wie oft muss man testen vor der Auslieferung/Abnahme? Wann kann man mit dem Testen aufhören? (vgl. Farr 2002a).

### 3.1.2 Ziele des Messens von Software Reliability

Die Messung von SR kann für die Planung und das Controlling der Ressourcen des Testens während der Software-Entwicklung genutzt werden (vgl. Huang; Lyu; Kuo 2003, S. 261). Um die Fehlerwirkungsdaten für Entscheidungen einsetzen zu können, werden Zuverlässigkeitsmodelle benutzt.

Zweck des SRE ist es, die SR zu spezifizieren, vorherzusagen, zu schätzen und zu bewerten (vgl. Farr 2002a). SR ist also eine „mit Mitteln der Stochastik beschreibbare Eigenschaft" (Liggesmeyer 2002, S. 428). Stochastik umfasst hier Wahrscheinlichkeitstheorie und Statistik, der Begriff wird in Abgrenzung zu funktionalen, deterministischen Verteilungen, Gleichungen und Zusammenhängen verwendet (vgl. Bortz 1993, S. 166). Zuverlässigkeitsprognosen können dabei mit statistischen Verfahren erzeugt werden (ebd.). Mit der stochastischen Software-Zuverlässigkeitsanalyse kann Zuverlässigkeit „gemessen, statistisch ausgewertet und für die Zukunft prognostiziert werden" (Liggesmeyer 2002, S. 440f.). Die hierzu

eingesetzten Modelle werden als Zuverlässigkeitsmodelle bezeichnet (ebd.) (siehe ausführlich Kap. 3.2).

Warum setzt man SRE ein? Es geht darum, Einschätzungen und Vorhersagen zu treffen. Ergebnisse des SRE können dazu genutzt werden, den optimalen Release-Zeitpunkt festzulegen sowie für bereits bestehende Programme zu entscheiden, wann diese neu geschrieben werden sollten (vgl. Farr 2002a).

Software Reliability Messungen beinhalten zwei Aktivitäten: reliability *estimation* and reliability *prediction* (vgl. Lyu 1996, S. 17). Es ist dabei wichtig, zwischen Vorhersagen („predictions“) und Schätzungen („estimations“) zu unterscheiden. Elbel und Mäckel bezeichnen die Schätzung als Analyse („Zuverlässigkeitsanalyse und -prognose“) (vgl. Elbel; Mäckel o.Jg.).

Schätzungen legen die aktuelle SR fest, indem inferenzstatistische Methoden auf Fehlerdaten angewandt werden (vgl. Lyu 1996, S. 17; Musa 1999, S. 260). Schätzungen werden mit Wahrscheinlichkeitsmodellen berechnet. Schätzungsmodelle („estimation models“) nutzen beobachtete Testdaten als Inputdaten.

Vorhersagen wiederum legen die zukünftige SR anhand von Softwaremetriken fest. Dabei kann man noch einmal, abhängig von dem Stadium der Softwareentwicklung, zwischen *reliability prediction* und *early prediction* unterscheiden (vgl. Lyu 1996, S. 17). Bei Vorhersagen werden die Werte von den Eigenschaften des Softwareprodukts und vom Entwicklungsprozess abgeleitet. Dies kann auch vor der Ausführung des Programms geschehen (vgl. Musa 1999, S. 260).

Üblicherweise sind die SR Schätzungen nicht das letzte Ziel, sondern es ist für einen Softwaremanager wichtig in der Lage zu sein, das zukünftige Verhalten von Softwarefehlerwirkungen vorherzusagen (vgl. Xie 1991, S. 35). Vorhersagen sind damit zentral, um den Einsatz zukünftiger Testressourcen planen und um Softwarereleaseprobleme analysieren zu können (vgl. ebd.).

### 3.1.3 Indikatoren für Software Reliability

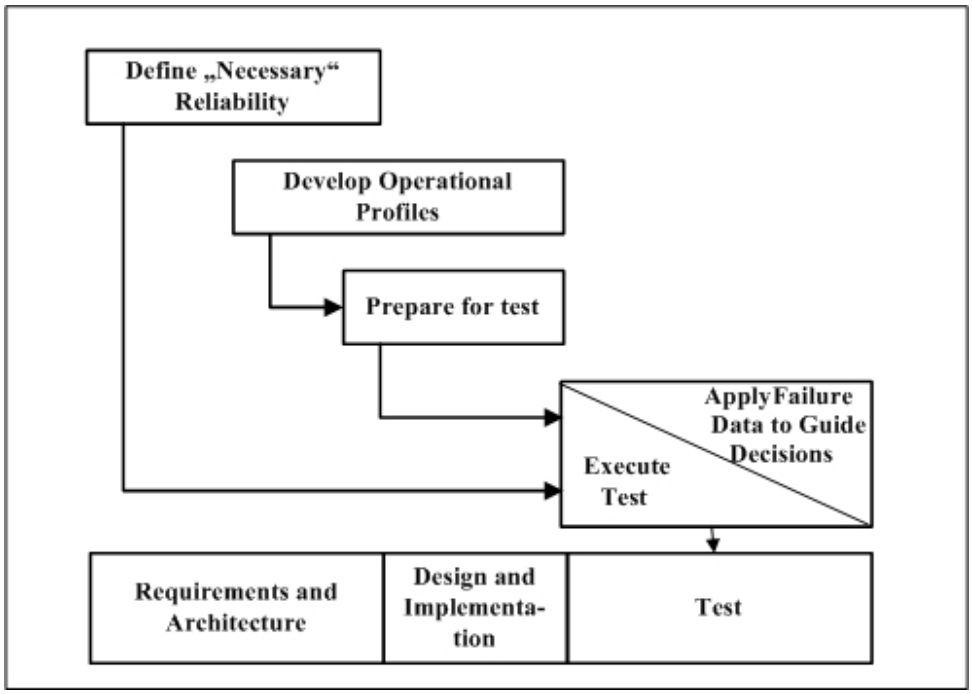
Als Zuverlässigkeitsmaße sind MTBF und MTTF zu nennen (vgl. Liggesmeyer 2002, S. 428). MTBF (mean time between failures) ist die erwartete Zeit zwischen zwei aufeinander folgenden Fehlerwirkungen/Ausfällen eines Systems. Daher ist MTBF eine zentrale Reliabilitätsmetrik für ein System, das verbessert und repariert werden kann.

MTTF (mean time to failure) ist die erwartete Zeit, bis ein System eine Fehlerwirkung zeigt. Nicht-reparierbare Systeme können dies nur einmal tun. Daher ist MTTF für ein nicht-reparierbares System ein Äquivalent zum Durchschnitt seiner Fehlerwirkungszeit-Verteilung. Reparierbare Systeme können öfter fehlschlagen. Im Allgemeinen dauert es länger, bis die erste Fehlerwirkung auftritt als es Zeit braucht bis die folgenden Fehler auftreten. Daher kann MTTF für ein reparierbares System eine von zwei Tatsachen repräsentieren: (1) die durchschnittliche Zeit bis zur ersten Fehlerwirkung (mean time to first failure, MTTFF) oder (2) die durchschnittliche Betriebszeit (mean uptime, MUT) innerhalb eines langfristigen Fehlerwirkungsreparierungs-Zyklus. Indikator für Vorhersagen ist die „Gesamtzahl übrig gebliebener Fehlerzustände“ (vgl. Farr 2002b), d.h. die Restfehler, die in dieser Arbeit Zentral sind (vgl. Kap. 5).

### 3.1.4 Der Software Reliability Prozess

Musa beschreibt den SRE-Prozess (vgl. folgende Abb. 3.1-1). Im SRE-Prozess ist es zunächst notwendig, festzulegen, welche Systeme mit dem Produkt, das man testen möchte, assoziiert sind. Hierzu muss man die verschiedenen Typen von SRE-Tests verstehen. Der SRE-Prozess besteht dann aus fünf Aktivitäten: „define necessary reliability, develop operational profiles, prepare for test, execute test, and apply failure data to guide decisions“ (Musa 1999, S. 5).

**Abbildung 3.1-1 Diagramm des Prozesses des Software Reliability Engineering**




Quelle: Musa 1999, S. 6

Die Aufgaben der vorliegenden Masterarbeit lassen sich in der obigen Abbildung 3.1-1 zwischen „Execute Test“ und „Apply Failure Data to Guide Decisions“ verorten, da es darum geht Software Reliability Modelle auf Fehlerdaten anzuwenden und daraus Informationen für das Testmanagement abzuleiten.

## *3.2 Software Reliability Modelle*

### 3.2.1 Grundlagen der Software Reliability Modelle

#### *3.2.1.1 Definition*

Die Bewertung der SR ist wie oben erläutert wichtig, um die Zuverlässigkeit und die Arbeitsleistung eines Softwaresystems beurteilen und vorhersagen zu können. Eine Möglichkeit, die SR zu messen, sind – neben Inspektionen und Testen – Software Reliability Modelle (vgl. Wallace 2001). Die Modelle, die für diese Bewertung eingesetzt werden, nennt man „software reliability growth models“ (SRGM[1]). Diese Modelle bieten außerdem eine plausible Beschreibung des Auftretens von Softwarefehlerwirkungen (vgl. Kapur; Garg 1999, S. 85). SRGMs können die Anzahl der Initial-Fehlerzustände, die Fehlerwirkungsintensität, die MTBF, die Restfehler sowie die Gesamtfehleranzahl einschätzen (vgl. Huang; Lyu; Kuo 2003).

Faktoren, die einen Einfluss auf den Fehlerprozess haben, sind: Einfügen von Fehlerzuständen (abhängig von den Produktmerkmalen und dem Entwicklungsprozess), Fehlerzustandbehebung (abhängig von der Zeit, der operationalen im Test genutzten Profile, Qualität der Behebungshandlungen) und die Verwendung (charakterisiert durch das operationale Profil) bzw. die Intensität und Art der Benutzung (vgl. Liggesmeyer 2002, S. 441). Musa definiert SRM folgendermaßen: ein SRM „specifies the general form of the dependence of the failure process on the factors mentioned“ (1999, S. 259). Die verschiedenen Modelle werden daher auch unterschieden durch die Wahrscheinlichkeitsverteilung der Fehlerwirkungszeiten oder der Anzahl der gefundenen Fehlerwirkungen und durch die Variation der Zufallsprozesse mit der Zeit (vgl. Musa 1999, S. 259). Denn Zuverlässigkeit wird durch die „zeitliche Verteilung des Eintretens von Fehlerwirkungen (…) bestimmt“ (Liggesmeyer 2002, S. 441).

[1] SRGM werden auch nur Software Reliability Modelle (SRM) genannt.

*3.2.1.2 Merkmale*

Software Reliability Modelle haben normalerweise die Form von Zufallsprozessen (random process), die das Verhalten der Fehlerwirkung in Bezug auf die Zeit beschreiben. Die Spezifikation eines Modells beinhaltet normalerweise eine Spezifikation einer Funktion der Zeit wie beispielsweise die Mittelwertfunktion der erwarteten Fehlerwirkungszahl oder die Fehlerwirkungsintensität. Die Funktionsparameter sind zuerst abhängig von der Fehlerzustandsbeseitigung (fault removal) und den Eigenschaften des Software-Produkts und des Entwicklungsprozesses. Die Produkt-Eigenschaften beinhalten die Größe, Komplexität und Struktur. Die wichtigste Produkteigenschaft ist die Größe von dem Entwicklungscode. Die in die Charakterisierung der Modelle eingebundene „Zeit“ ist eine kumulierte Zeit (vgl. Musa 1999, S. 261f.).

Die Mittelwertfunktion $\mu(t)$ ist definiert als $\mu(t) = E(M(t))$, welche die erwartete Fehlerwirkungsanzahl an einem Zeitpunkt *t* repräsentiert. Man nimmt an, dass die Funktion $\mu(t)$ eine nicht verringerbare, kontinuierliche und differenzierbare Funktion der Zeit *t* ist. Die Fehlerwirkungsintensitätsfunktion von dem *M(t)* Prozess ist die unmittelbare Rate der Änderungen der erwarteten Fehlerwirkungsanzahl mit der Zeit. Man definiert sie

$$\lambda(t) = \frac{d\mu(t)}{dt}$$

Der Einsatz von Zufallsprozessmodellen ist angemessen, weil die Prozesse von vielen Zeitvariablen abhängen. So lässt sich der Fehlerwirkungszufallprozess auf zwei äquivalente Weisen beschreiben: die Fehlerwirkungszeit oder die Anzahl von Fehlerwirkungen in einer gegebenen Zeit (vgl. Musa 1999, S. 263).

SRM nehmen fast immer an, dass die Fehlerwirkungen unabhängig voneinander sind. Fehlerwirkungen sind das Resultat von zwei Prozessen: die Einführung von Fehlerzuständen und ihr Auslösen durch Selektion der Input-Zustände. Beide Prozesse sind zufällig, daher ist die Wahrscheinlichkeit, dass ein Fehler einen anderen beeinflusst, sehr gering (vgl. ausführlich zur Unabhängigkeitsannahme Musa 1999, S. 262).

Ein gutes Software Reliability Modell hat verschiedene wichtige Eigenschaften. Es gibt (1) eine gute Prognose des zukünftigen Fehlerwirkungsverhaltens, berechnet (2)

nützliche Quantitäten, ist (3) einfach, (4) breit anwendbar und basiert (5) auf soliden Vermutungen (vgl. Musa 1999, S. 260).

SRM müssen zwei Situationen abdecken, nämlich Programme, in welchen die Fehlerzustände behoben werden, wenn Fehlerwirkungen erscheinen („fault removal“) und Programme, in welchen die Fehlerzustände nicht bzw. später behoben werden („no fault removal“). Bei letzterem identifizieren die Tester die Fehlerzustände, aber die Entwickler beheben die Fehlerzustände nicht bis zum nächsten Release – d.h. die Fehlerwirkungsintensität ist konstant für die Dauer des Release. Man kann diese Fehlerwirkungsprozesse mit Homogeneous Poisson Prozessen modellieren (vgl. Musa 1999, S. 265). Bei „fault removal“ ist die Situation schwieriger: „The situation of „with fault removal“ occurs in reliability growth test“ (Musa 1999, S. 265).

Für viele Modelle gibt es analytische Ausdrucke (1) für die durchschnittliche Anzahl von Fehlerwirkungen an jedem Zeitpunkt, (2) die durchschnittliche Anzahl von Fehlerwirkungen in einem Zeitintervall, (3) die Intensität von Fehlerwirkungen an jedem Zeitpunkt und (4) die Wahrscheinlichkeit der Verteilung von Fehlerwirkungsintervallen (vgl. Musa 1999, S. 260).

#### *3.2.1.3 Annahmen*

Die Prognose des zukünftigen Verhaltens der Fehlerwirkungen nimmt an, dass sich die Werte der Modellparameter während der Prognoseperiode nicht ändern. Wenn es jedoch der Fall ist, dass die „fault introduction“ und „fault removal“ wesentlich geändert werden, muss man entweder eine Kompensation für die Änderung vornehmen oder man muss warten, bis genügend neue Fehlerwirkungen erscheinen, so dass man neue Schätzungen der Modellparameter machen könnte. Das Einfügen solcher Änderungen ist theoretisch möglich, aber in der Praxis wegen der zusätzlichen Komplexität unmöglich (vgl. Musa 1999, S. 260).

Die meisten SRM basieren auf der Verwendung von stabilen Programmen in einer stabilen Art. Diese Methode bedeutet, dass weder der Code noch die Verwendung (operational set of possible operations and their probabilities of ocurrence) gewechselt worden sind. Wenn die Programme und Umgebung sich verändern, muss man die entsprechenden Modelle wählen. Daher gibt es Modelle, die die Fehlerzustandsbehebung fokussieren (vgl. Musa 1999, S. 260f.).

#### *3.2.1.4 Ziele und Einsatz von Software Reliability Modellen in der Softwareentwicklung*

Ein gutes Modell verbessert die Kommunikation in einem Projekt und liefert einen gemeinsamen Rahmen des Verständnisses für die Software-Entwicklungs-Prozesse. Außerdem fördert es die Transparenz für das Management. Die Entwicklung von SRM, die in der Praxis nützlich sind, umfasst theoretische Arbeit, Werkzeugerstellung, und die Akkumulation von einer Menge von Wissen aus der praktischen Erfahrung. Der Versuch braucht normalerweise einige Personenjahre (vgl. Musa 1999, S. 261).

Während stochastiche Zuverlässigkeitsanalysen bei Hardware standardmäßig eingesetzt werden, werden für Software-Systeme in der Praxis aufgrund des hohen Aufwands bislang noch eher selten Zuverlässigkeitsanalysetechniken eingesetzt (vgl. Liggesmeyer 2002, S. 441).

Für Forschungsprojekte wird der Einsatz von mehreren Modellen vorgeschlagen. Allerdings ist die Anwendung von mehr als einem oder zwei Modellen konzeptuell und wirtschaftlich nicht praktikabel für reale Projekte. Denn es ist notwendig, dass die Projektpersonen verstehen, was die Parameter von den Modellen bedeuten, damit sie diese bewerten können. Dies ist nicht gleichzeitig für mehrere Modelle möglich. Außerdem eskalieren Zeit und involvierte Kosten sehr schnell, je mehr Modelle man einsetzt (vgl. Musa 1999, S. 261).

### 3.2.2 Klassifizierung der Modelle

Etwa 75 Software Reliability Modelle stehen zur Verfügung. Es gibt verschiedene Möglichkeiten, die SRM zu klassifizieren. Musa (1999, S. 267) erstellte beispielsweise ein Klassifikationsschema, um die Modelle nach verschiedenen Eigenschaften zu unterteilen. Xie unterscheidet u.a. Markov Modelle, Nonhomogeneous Poisson Prozessmodelle, Bayesian Modelle und statistische Datenanalysemethoden (vgl. Xie 1991, S. 24). Die Klassifizierung nach Xie ist besonders übersichtlich und wird daher im Folgenden kurz erläutert.

#### *3.2.2.1 Markov Modelle*

Ein Modell gehört zu den Markov Modellen, wenn seine Wahrscheinlichkeitsannahme des Fehlerwirkungszählprozesses ein Markov-Prozess ist, üblicherweise ein Geburt-Tod-Prozess. Die Hauptcharakteristik dieser Modelle ist, dass die Software an einem gegebenen Zeitpunkt zählbare Zustände hat und dass diese Zustände die Anzahl der

übriggebliebenen Fehlerzustände repräsentieren. Daraus ist abgeleitet, dass die zukünftige Entwicklung nicht von der vergangenen abhängt, es wird daher eine diskontinuierliche Funktion angenommen (vgl. Xie 1991, S. 24). Markov Modelle arbeiten mit der Stochastik. Zu dieser Klassifikation gibt es mehrere Modelle (vgl. Blischke; Murthy 2000, S. 303).

#### *3.2.2.2 Nonhomogeneous Poisson Prozessmodelle (NHPP)*

Bei diesen Modellen ist die Annahme, dass der Fehlerwirkungsprozess als ein Nonhomogeneous Poisson Prozess zu beschreiben ist. NHPP werden oftmals bei der Analyse von Hardware Reliability eingesetzt. Sie können jedoch aufgrund der Ähnlichkeit des Zuverlässigkeitswachstums von Hard- und Software auch für die Analyse von SR adaptiert werden. NHPP zeichnet Folgendes aus: „there is a mean value function which is defined as the expected number of failures up to a given time." (Xie 1991, S. 25). NHPP arbeiten mit der Stochastik. In dieser Arbeit werden NHPP-Modelle zur Analyse verwendet, weil die Merkmale der Fehlerdaten ihren Einsatz sinnvoll erscheinen lassen. Daher werden sie ausführlich in Kapitel 3.2.3 beschrieben.

#### *3.2.2.3 Bayesian Modelle*

Diese Modelle nutzen inferenzstatistische Methoden. Sie schließen auch vorheriges Wissen mit ein. D.h. diese Modelle nutzen Informationen über die Software, die vor dem Testen erhoben werden, in Kombination mit Testdaten (vgl. Xie 1991, S. 25).

#### *3.2.2.4 Statistische Datenanalysemethoden*

Unter diesen Methoden fasst Xie time series Modelle, proportional hazards Modelle und Regressionsmodelle zusammen (1991, S. 25f.).

### 3.2.3 Darstellung der Nonhomogeneous Poisson Prozessmodelle

In diesem Kapitel wird zuerst allgemein auf die NHPP-Modelle eingegangen, um danach zwei NHPP-Modelle näher zu beschreiben, die besonders geeignet sind und mit denen die Analyse in dieser Arbeit durchgeführt wird. Warum diese Modelle zu den Daten bei BMW passen, wird in Kapitel 4.5 beschrieben.

#### *3.2.3.1 Allgemeine Theorie*

Das NHPP-Modell ist ein Modell des Poisson-Typs, das die Fehlerzustandsanzahl pro Zeiteinheit als unabhängige Poisson-Zufallsvariable annimmt. Es handelt sich um ein

„finite failure model“ (Farr 1996, S. 80). Das Modell wurde erstmals 1979 von Goel und Okumoto vorgestellt und stellt die Basis für Modelle dar, die die beobachtete Fehlerzustandsanzahl pro Zeiteinheit nutzen (vgl. Farr 1996, S. 80). Obwohl verschiedene NHPP-Modelle existieren, können die meisten NHPP-basierten SRMs als spezielle Fälle dieses allgemeinen Modells gesehen werden (vgl. Huang; Lyu; Kuo 2003, S. 262). Diese Modelle beachten den „debugging“ Prozess als einen Zählprozess (counting process), der sich durch die Mittelwertfunktion auszeichnet (Pham 2006, S. 179). Die Modelle gehen von folgenden Annahmen aus:

- Die kumulierte Anzahl der Fehlerwirkungen zu einem Zeitpunkt *t*, *M(t)*, folgt einem Poissonprozess mit einer Mittelwertfunktion $\mu(t)$. Die Mittelwertfunktion ist so gestaltet, dass die erwartete Anzahl von Fehlerzustandereignissen für eine beliebige Zeit *t* zu $t + \Delta t$ proportional zu der erwarteten Anzahl von unentdeckten Fehlerzuständen in einer Zeit *t* ist.

- Die Anzahl der entdeckten Fehlerzustände $(f_1, f_2,..., f_n)$ in jeder der entsprechenden Intervalle $[(t_0 = 0, t_1), (t_1, t_2), (t_{i-1}, t_1), ...(t_{n-1}, t_n)]$ ist unabhängig für eine beliebige Sammlung für die Zeit, $t_1 < t_2 < ... < t_n$.

Das Hauptmerkmal dieses Modelltyps ist, dass es eine Mittelwertfunktion gibt, welche definiert ist, als die erwartete Fehlerwirkungsanzahl bis zu einem gegebenen Zeitpunkt (vgl. Xie 1991, S. 25). Die NHPP-Modelle unterscheiden sich daher in ihren Mittelwertfunktionen (vgl. Huang; Lyu; Kuo 2003, S. 262). Die Parameter des Modells werden normalerweise mit der „maximum likelihood“ Methode bestimmt (vgl. Pham 2006, S. 179).

$\{N(t), t \geq 0\}$ stellt einen Zählprozess, der die kumulierte Anzahl der entdeckten Fehlerzustände beim Zeitpunkt *t* abbildet, dar. Ein SRM, das auf einem NHPP mit der Mittelwertfunktion (MVF) *m(t)* basiert, kann wie folgt formuliert werden:

$$P\{N(t) = n\} = \frac{m(t)^n}{n!} e^{-m(t)}, \; n = 0, 1, 2,...$$

wobei *m(t)* die erwartete kumulierte Anzahl von Fehlerzuständen, die in der Zeit *t* entdeckt wurden, repräsentiert. Die MVF *m(t)* nimmt nicht ab in Bezug auf die Testzeit *t* unter der begrenzten Voraussetzung $m(\infty) = a$, wobei *a* die erwartete totale Fehlerzustandsanzahl ist, die eventuell entdeckt wird. Die MVF *m(t)* zu kennen, kann

dabei helfen, festzulegen, ob die Software an den Kunden geliefert werden kann und wie viele Testressourcen man noch benötigt. Man kann außerdem die Anzahl der Fehlerwirkungen einschätzen, die eventuell noch vom Kunden entdeckt werden. Die Fehlerwirkungsintensitätsfunktion für eine gegebene Zeit *t* ist wie folgt definiert

$$\lambda(t)=\frac{dm(t)}{dt} = m'(t)$$

Die Wahrscheinlichkeit, dass keine Fehlerwirkungen in $(s, s+t)$ auftreten, angenommen, dass die letzte Fehlerwirkung in der Zeit $s$ $(s \geq 0, t > 0)$ auftritt, lautet dann:

$$R(t/s) = e^{[-(m(t+s)-m(t))]}$$

Die Fehlerzustandsentdeckungsrate pro Fehlerzustand bei Zeit des Testens *t* ist gegeben als

$$d(t) = \frac{m'(t)}{a - m(t)} = \frac{\lambda(t)}{a - m(t)}$$

Die vorige Gleichung meint die Entdeckbarkeit eines Fehlerzustands im laufenden Fehlerzustandsinhalt (vgl. Huang; Lyu; Kuo 2003, S. 262).

Im Folgenden werden die beiden in der Masterarbeit eingesetzten Modelle Goel-Okumoto und Yamada-Delayed-S-shaped erläutert.

#### *3.2.3.2 Goel-Okumoto Modell*

Dies Modell wurde von Goel und Okumoto vorgeschlagen und ist das wichtigste NHPP-Modell, das einen „konkaven“ Modelltyp hat (Zhang; Pham 2001, S. 22). Es heißt auch exponentielles NHPP Modell. Man kann die Fehlerwirkungsentdeckung als einen NHPP mit einer exponentiell abfallenden Funktionsrate sehen (vgl. Huang; Lyu; Kuo 2003, S. 262). Dieses Modell hat folgende Voraussetzungen:

1. Alle Fehlerzustände im Programm sind voneinander unabhängig von der Fehlerwirkungsentdeckung aus betrachtet.
2. Die Anzahl der entdeckten Fehlerwirkungen an einem Zeitpunkt ist proportional zu der aktuellen Anzahl der Fehlerzustände. Das bedeutet, dass die Wahrscheinlichkeit der Fehlerwirkungen für wirklich auftretende Fehlerzustände konstant ist.

3. Die isolierten Fehlerzustände sind behoben, bevor der zukünftige Test passiert.
4. Jedes Mal, wenn eine Software-Fehlerwirkung auftaucht, wird der Software-Fehler, welcher die Fehlerwirkung verursachte, sofort behoben, und keine neuen Fehler werden eingefügt (vgl. Pham 2006, S.183).

Die Mittelwertsfunktion lautet

$$m(t) = a\left(1 - e^{(-bt)}\right),\ a > 0,\ b > 0,$$

wobei *a* die erwartete totale Fehlerzustandsanzahl ist, die eventuell entdeckt wird und *b* entspricht der Fehlerzustandsentdeckungsrate (vgl. Huang; Lyu; Kuo 2003, S. 262; vgl. Yin; Trivedi 1999, S. 6f.).

Für die Intervall-Domain-Data Methode (vgl. Kap. 3.3.1.1) kann man die Parameter *a* und *b* mittels der Maximum Likelihood Estimator (MLE) festlegen. Wir können auch die Reliability-Funktion wie folgt darstellen (vgl. Pham 2006, S. 183f.):

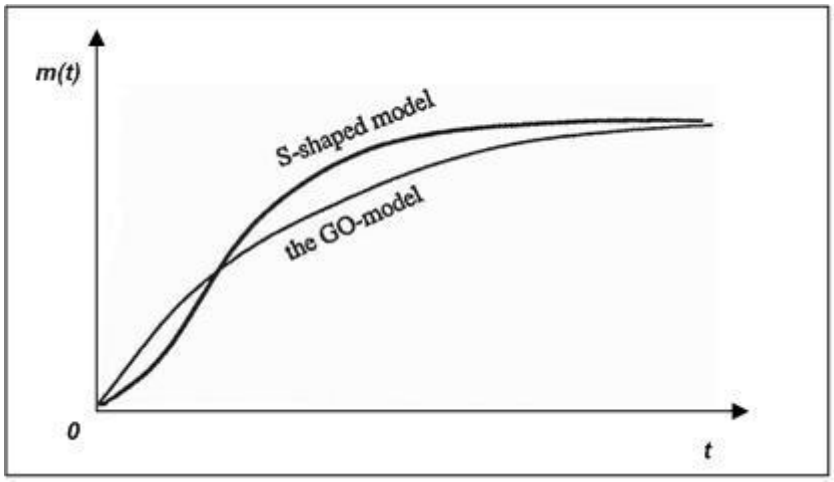
$$R(x/t) = e^{-a\left[e^{-bt} - e^{-b(t+x)}\right]}$$

#### *3.2.3.3 S-shaped NHHP Modelle*

Es gibt mehrere S-shaped NHPP-Modelle. Im Folgenden wird das besonders geeignete „Yamada-Delayed S-shaped“ vorgestellt (vgl. Xie 1999, S. 95; vgl. zu den anderen S-shaped NHPP-Modellen Pham 2006, S. 188ff.).

Die Mittelwertfunktion des Goel-Modells hat eine exponentielle Form (shape). Auf der Basis von dieser Erfahrung, beobachtet man in der folgenden Abbildung 3.2-1, dass die Kurve von der kumulierten Fehlerzustandsanzahl des S-shaped-Modells oft eine S-Form (S-shaped) relativ zu einer exponentiellen Mittelwertsfunktion hat, was bedeutet, dass die Kurve die exponentielle Kurve ganz unten kreuzt und die Kreuzung nur einmal auftritt. Diese Anzahl von Modifikationen oder Generalisationen des Modells wurden von Ohba und Yamada vorgeschlagen (vgl. Xie 1999, S. 94).

**Abbildung 3.2-1 Die S-Form der Fehlerwirkungsintensitätsfunktion**




Quelle: Xie 1999, S. 95

Ein Grund für das S-shaped-Kurven-Verhalten wird in Xie (1991, S. 95) genannt: Das Software-Reliability-Testen beinhaltet normalerweise einen Lernprozess, in dem die Tester vertraut mit der Software und den Testtools werden. Dadurch nehmen ihre Fähigkeiten und damit auch die Testeffektivität langsam zu.

So gibt es eine Tendenz, dass die kumulierte Anzahl der Fehlerwirkungen eine S-shaped-Form hat, was impliziert, dass der Anstieg der Reliabilität am Anfang aufgrund der geringen Testeffektivität nur sehr moderat ist. Danach steigt die Reliabilität schnell an und später wird die Verbesserung wieder langsamer, weil die meisten der Fehlerzustände bereits entfernt wurden (vgl. Xie 1991, S. 95). Die übrig gebliebenen Fehlerzustände sind darüber hinaus schwieriger zu finden (vgl. Farr 2002b).

Das „Yamada-Delayed-S-shaped" Modell ist eine Art der NHPP, um eine „S-shaped" Kurve für die kumulierte Fehlerwirkungsanzahl zu erhalten, so dass die Fehlerwirkungsrate anfangs zunimmt und sie später exponentiell abfällt. Man kann das Modell als ein generalisiertes exponentielles Modell mit einer zuerst erhöhenden und dann verringernden Fehlerwirkungsrate sehen. Der Software-Fehlerentdeckungsprozess, der mit solch einer S-shaped Kurve beschrieben wird, kann als ein Lernprozess betrachtet werden, da die Fähigkeit der Tester langsam mit dem Lauf der Zeit verbessert werden kann. Dieses Modell hat u.a. folgende Voraussetzungen:

1. Alle Fehlerzustände im Programm sind voneinander unabhängig von der Fehlerwirkungsentdeckung aus betrachtet.

2. Die Fehlerwirkungsentdeckungswahrscheinlichkeit jederzeit ist proportional zu der aktuellen Anzahl von entdeckten Fehlerzuständen in der Software.
3. Die Proportionalität der Fehlerwirkungsentdeckung ist konstant.
4. Der Anfang des Fehlerinhaltes von der Software ist eine Zufallsvariable.
5. Jedes Mal, wenn eine Software-Fehlerwirkung auftaucht, wird der Software Fehler, welcher die Fehlerwirkung verursachte, sofort behoben, und keine neuen Fehler werden eingefügt (vgl. Pham 2006, S. 190f.)

Die Mittelwertsfunktion lautet

$$m(t) = a\left(1 - (1 + bt)e^{(-bt)}\right),\ a > 0,\ b > 0,$$

wobei *a* die total erwartete Fehlerzustandsanzahl ist, die eventuell entdeckt wird, und *b* die Fehlerzustandsentdeckungsrate ist (vgl. Huang; Lyu; Kuo 2003, S. 262f.; vgl. Yin; Trivedi 1999, S.7).

Die Reliability vom Softwaresystem ist

$$R(x/t) = e^{-a\left[(1+bt)e^{-bt} - (1+b(t+s))e^{b(t+s)}\right]}$$

Für die Intervall-Domain-Data Methode kann man die Parameter *a* und *b* mittels der MLE festlegen und eine Gleichungssystem berechnen (vgl. Pham 2006, S. 191).

## *3.3 Die Auswahl von Software Reliability Modellen*

Die Modellauswahl ist eine komplexe Herausforderung: „Die Einschätzung der Eignung der Modelle ist aufgrund der Modellvielfalt und ihrer spezifischen Voraussetzungen für den Anwender kompliziert“ (Liggesmeyer 2005b, S. 221).

Die Modellauswahl beinhaltet zwei Schritte: der erste Schritt (siehe Kap. 3.3.1) besteht in der generellen Auswahl von Modellen, die für die Fragestellung und die Daten passend wären. Hierzu ist die Datenanalyse wichtig, um die Charakteristika der Daten mit den Anforderungen der Modelle zu vergleichen (wie z.B. „mit/ohne fault removal“) (siehe Kap. 3.3.1). Im zweiten Schritt (siehe Kap. 3.3.2) wird – nachdem die ausgewählten Modelle auf die Daten angewandt und berechnet wurden – das von diesen Modellen am besten passende ausgewählt. In diesem Kapitel wird die Modellauswahl theoretisch erläutert, nach den gleichen Kriterien erfolgt dann die Auswahl in Bezug auf die empirischen Daten in Kapitel 4.5.

### 3.3.1 Analyse der Daten nach Kriterien

Die Datenanalyse – und darauf aufbauend die spätere Modellauswahl – erfolgt nach drei Kriterien: Fehlerdatentypen, Art des Testprozesses und Art der Testobjekte in Bezug auf die Anzahl der Fehlerwirkungen. Darüber hinaus spielt die absolute Anzahl der vorhandenen Daten für die Modellauswahl eine Rolle. Es sollten mindestens fünfmal so viele Daten zur Verfügung stehen wie Parameter in einem speziellen Modell (vgl. Wallace 2001).

#### *3.3.1.1 Fehlerdatentypen*

Abhängig vom Format, in welchem die Testdaten vorliegen, gibt es zwei verschiedene Methoden, die normalerweise benutzt werden.

Intervall-Domain-Data Methode

Der Intervall-Domain Ansatz ist durch das Zählen der Anzahl von Fehlerwirkungen charakterisiert, die in einem festen Intervall (z.B. Testsession, Stunde, Woche, Tag, etc.) erscheinen. Unter Verwendung von dieser Methode sind die gesammelten Daten die Fehlerwirkungsanzahl in einem Intervall (vgl. Pham 2006, S. 136).

Angenommen, dass die Daten für die kumulierte Anzahl von entdeckten Fehlern $y_i$ in einem gegebenen Zeitintervall $(0, t_i)$ gegeben sind, wobei $i = 1, 2, ..., n$ und $0 < t_1 < t_2 < .... < t_n$, dann nimmt die Log Likelihood Funktion (LLF) die folgende Form an:

$$LLF = \sum_{i=1}^{n} (y_i - y_{i-1}). \log[m(t) - m(t_{i-1})] - m(t_n)$$

In Pham (2006, S. 180) wird das Maximum von der LLF festgelegt, um die erwartete Anzahl von Fehlern zu entdecken.

Time-Domain-Data Methode

Um diese Methode anzuwenden, müssen die Fehlerwirkungen minutengenau erfasst werden. Man nimmt an, dass die Daten für die Erscheinungszeit der Fehlerwirkung oder die Zeit von den nachfolgenden Fehlerwirkungen gegeben sind, z.B. die Ausführung des Freiheitsgrads $S_j$ für $j = 1, 2, .., n$. Gegeben, dass die Daten *n* aufeinanderfolgende Zeiten von betrachtenden Fehlerwirkungen $s_j$ für $0 < s_1 \le s_2 \le ... s_n$ liefern, können wir diese Daten in der Zeit bis zur nächsten Fehlerwirkung (MTBF) $x_i$ wobei $x_i = s_i - s_{i-1}$

für $i = 1, 2, ..., n$ konvertieren, dann nimmt die Log Likelihood Funktion (LLF) die folgende Form an:

$$LLF = \sum \log[\lambda(s_i)] - m(s_n)$$

Je nachdem, in welcher Form die Daten erhoben werden (intervall- oder minutenbasiert), kommen unterschiedliche SRM für die Analyse in Betracht.

#### *3.3.1.2 Art des Testprozesses*

Neben der Art der vorliegenden Daten, ist auch der Testprozess, in dem die Daten erhoben wurden, für die Modellauswahl relevant. Hier kann man Prozesse unterscheiden, in welchen die Fehlerzustände behoben werden, wenn Fehlerwirkungen erscheinen („fault removal“) und solche, in welchen die Fehlerzustände nicht bzw. später behoben werden („no fault removal“). „No fault removal“-Prozesse werden mit Homogeneous Poisson Prozessen modelliert, solche mit „fault removal“ mit NHPP-Modellen (vgl. Musa 1999, S. 265). Prozess mit „fault removal“ führen zu einer Sättigung der kumulierten Fehlerkurven (da davon ausgegangen wird, dass keine neuen Fehler bei der Beseitigung eingefügt werden) (vgl. auch Abb. 2.2-3 „Fehlerverfolgung bei Software“).

#### *3.3.1.3 Art der Testobjekte in Bezug auf die Anzahl der Fehlerwirkungen*

Nicht nur die Art des Testprozesses ist relevant für die Modellauswahl, sondern auch die Art der Testobjekte in Bezug auf die Anzahl der Fehlerwirkungen. Man kann zwischen „finite“ und „infinite“ failure Modellen unterscheiden (vgl. Farr 1996, S. 80; vgl. Musa 1999, S. 267). Finite Failure Modelle gehen davon aus, dass die Fehlerwirkungen endlich sind. Hierzu zählen die NHPP-Modelle. Infinite failure Modelle gehen davon aus, dass die Fehlerwirkungen unendlich sind. Dabei implizieren unendliche Fehlerwirkungen nicht notwendig unendliche Fehlerzustände. Eine Situation von unendlichen Fehlerzuständen erscheint unwahrscheinlich, da die Anzahl der Codes endlich ist. Der unendliche Fehlerwirkungsfall ist nur glaubhaft für ein Programm, das für eine unendliche Zeit durchgeführt wird (vgl. Musa 1999, S. 266f.). Bei der Modellauswahl ist es daher wichtig, zu überlegen, ob von unendlichen oder endlichen Fehlerwirkungen ausgegangen wird.

### 3.3.2 Passung der Modelle nach Anwendung der Modelle und Parameterschätzung

Der zweite Schritt betrachtet die deskriptive und voraussagende Stärke eines Modells in Bezug auf die speziellen Daten, d.h. die Passung auf die spezifischen Daten. Es gibt mehrere Kriterien, um die Modelle auf ihre deskriptive („goodness of fit“) und voraussagende Fähigkeit und Stärke zu vergleichen (vgl. ausführlich Pham 2006, S. 181).

„Die Einschätzung der Eignung der Modelle ist aufgrund der Modellvielfalt und ihrer spezifischen Voraussetzungen für den Anwender kompliziert. Darüber hinaus ist neben der Bewertung der grundsätzlichen Eignung eines Modells eine Festlegung der Modellparameter erforderlich” (Liggesmeyer 2002, S. 451):

Die Herausforderung besteht darin, die Modelle an die Daten anzupassen – also die empirische Verteilung der Fehlerhäufigkeit mit einer theoretischen Verteilung zu vergleichen – d.h. die zwei bis drei Parameter (siehe Kap. 3.2.3.2 und 3.2.3.3) zu bestimmen, was häufig über Maximum Likelihood Schätzungen gemacht wird (vgl. Pham 2006, S. 180; vgl. Liggesmeyer 2002, S. 452ff.; vgl. Huang et al. 2000, S. 76). Diese Likelihood-Funktion der stochastischen Prozesse ist jedoch meist analytisch nicht lösbar, weshalb numerische Optimierungsalgorithmen (z.B. Nelder-Mead-Simplex-Methode) eingesetzt werden. Dies erfordert ausführliche mathematische Prozesse, die den Rahmen dieser Masterarbeit jedoch überschreiten (vgl. M-SRAT o.Jg.).

Um diese Parameter einer bekannten Funktion zu bestimmen, benötigt man das „kleinste Fehlerquadrate (Least Squares)“ Verfahren und das „Maximum-Likelihood“ Verfahren sowie Chi-Quadrat, um ein ausgewähltes Zuverlässigkeitsmodell möglichst gut an die vorliegenden Ausfalldaten anzupassen. Einerseits ist das kleinste Fehlerquadrat definiert als das Quadrat der Abweichung zwischen jeder Beobachtung und dem Wert, den das Zuverlässigkeitsmodell an dieser Stelle liefert. Die Parameter ergeben als die Summe dieser Quadrate einen möglichst kleinen Wert. Wenn es so der Fall ist, so bewertet dieses Verfahren die entsprechenden Parameter als die beste Lösung. Andererseits ist das Ziel des Maximum-Likelihood-Verfahrens die freien Parameter so zu wählen, dass die Wahrscheinlichkeit maximiert wird, eine zur vorliegenden Beobachtung „ähnliche“ Beobachtung zu erzeugen (vgl. Liggesmeyer 2002, S. 451; vgl. Tosteson; Demidenko 2002, S. 1057f.).

Der Chi-Quadrat-Test wird als „goodness of fit test“ bezeichnet, da er dazu dient, zu überprüfen, ob die geschätzte Kurve der SR-Modelle zu den realen Daten passt, er ermittelt die Güte der Anpassung. Es geht darum, eine beobachtete Verteilung mit einer theoretischen Verteilung zu vergleichen. Eine einfache Methode ist der Chi-Quadrat-Test (vgl. Pham 2006, S. 96).

Die Schritte des Chi-Quadrat-Test sind wie folgt:

1. Die Daten werden in gegenseitig exklusive Zellen (normalerweise 8-12) geteilt bis der Rang der Zufallsvariable gedeckt ist.
2. Festlegen der Frequenz $f_i$ der beobachteten Daten für jede Zelle.
3. Festlegen der theoretischen Frequenz $F_i$ für jede Zelle. Die theoretische Frequenz soll nicht kleiner als 1 sein. Um diesen Schritt durchzuführen, benötigt man die Einschätzung der Populationsparameter, welche man aus den Daten erhalten kann.
4. Festlegung der Statistik

$$S = \sum_{i=1}^{k} \frac{\left(f_i - F_i\right)^2}{F_i}$$

5. Aus der $X^2$ Tabelle wählt man einen Wert von $X^2$ mit dem gewünschten Signifikanzniveau und mit dem Freiheitsgrad *(= k-1-r)*, wobei *r* die Anzahl der eingeschätzten Populationsparameter darstellt.
6. Ablehnen der Hypothese, dass die beobachtete Verteilung die gleiche wie die theoretische Verteilung ist, wenn

   $$S > X^2_{1-\alpha,k-1-r}$$

   wobei $\alpha$ das Signifikanzniveau ist (vgl. Pham 2006, S. 96f.; vgl. Lyu 1996, S. 774f.; vgl. Birolini 2007, S. 316ff.).

In der vorliegenden Arbeit werden nach Wallace (2001) die deskriptive und voraussagende Stärke eines Modells durch den Chi-Quadrat-Test sowie die Konfidenzintervalle berechnet sowie dadurch, dass die geschätzte Gesamtfehleranzahl und die vorhergesagte Restfehleranzahl mit den realen Anzahlen verglichen werden.

## *3.4 Zusammenfassung*

Nach der Definition von Software Reliability und Software Reliability Engineering (siehe Kap. 3.1) wurden die Software Reliability Modelle nach Xie klassifiziert. Aufgrund der großen Anzahl von Software Reliability Modellen (siehe Kap. 3.2) ist es für die Anwendung zentral, einige Modelle auszuwählen. Wichtig für die Auswahl von Software Reliability Modellen ist die Analyse der Daten nach drei verschiedenen Kriterien (Fehlerdatentypen, Art des Testprozesses und Art der Testobjekte) und danach die Einschätzung der Parameter und die Passung der Modelle auf die spezifischen Daten (siehe Kap. 3.3). Nach diesen Kriterien wird auch die Modellauswahl mit den empirischen Daten in Kapitel 4.5 erfolgen. Da sich bei der Analyse der empirischen Daten herausstellte, dass insbesondere die Modelle der Nonhomogeneous Poisson Prozesse geeignet sind – und zwar die Goel-Okumoto und Yamada-Delayed-S-shaped Modelle – wurden diese im vorliegenden theoretischen Kapitel detailliert beschrieben (siehe Kap. 3.2.3).

# 4 Anwendungsfeld, Datenbasis und Design der empirischen Analyse

In diesem Kapitel geht es um das Design der empirischen Analyse. Zuerst wird das Anwendungsfeld beschrieben, nämlich Infotainment in der Automobilbranche und speziell bei BMW (Kap. 4.1). Darauf folgend werden Ziele und Fragestellungen der empirischen Analyse vorgestellt (Kap. 4.2).

Nach der Erstellung eines Überblicks über SRE-Modelle (vgl. Kap. 3) waren vor allem vier vorbereitende Schritte vor der empirischen Anwendung der SRE-Modelle (vgl. Kap. 5) notwendig:

- Einarbeitung in vorliegende statistische Daten (vgl. Kap. 4.3)
- Modellierung der Testinstanzen zur Auswahl der Daten (vgl. Kap. 4.4)
- Abgleich der Daten mit passenden SRE-Modellen (vgl. Kap. 4.5)
- Erstellung eines Überblicks über Softwarewerkzeuge zu SRE-Modellen und Auswahl eines Werkzeugs (vgl. Kap. 4.6)

Nach der Anwendung der SRE-Modelle auf die vorliegenden Daten erfolgte die Bewertung und Analyse der Ergebnisse (vgl. Kap. 5 und Kap. 6).

## *4.1 Anwendungsfeld: Infotainment in der Automobilbranche*

### 4.1.1 Infotainment und Software in der Automobilbranche

Neue Innovationen im Fahrzeug, die auf Software basieren, sind in der Automobilindustrie sehr wichtig. Sie sind komplex und entstehen unter Zeit- und Kostendruck. An sie werden hohe Qualitätsanforderungen gestellt mit dem Ziel, die Sicherheit der Fahrzeuge zu erhöhen, den Komfort der Fahrer zu optimieren und Verbrauch und Emissionen zu vermindern (vgl. Grimm 2005, S. 407f.). Wichtige Innovationen z.B. im Bereich des Infotainments im Fahrzeug sind ohne Software-Lösungen nicht zu denken und zukünftig wird sich der Software-Anteil sogar noch verstärken (vgl. Grimm 2005, S. 409). Bei Infotainment-Geräten handelt es sich um eingebettete Systeme (Hardware und Software) (vgl. Salzmann; Stauner 2005, Kap. 2 „Eingebettete Systeme: Domäne Karosserie / Komfort“).

Infotainment entsteht aus Information und Entertainment und bezeichnet eine multimediale Kommunikationsform, bei der die Information mit Unterhaltung

kombiniert wird (vgl. ITWissen 2007). Infotainment-Produkte gehen über die Darstellung von Textinformationen hinaus. Zentrale Bestandteile sind daher neben Textinformationen auch Videos, Animationen und Sounds (vgl. Irlbeck 1998, S. 315). Infotainmentgeräte werden sowohl im privaten Bereich und im Unternehmen als auch in der Automotive-Technik, bei Rundfunk und Fernsehen, Mobiltelefonie und zur Navigation eingesetzt (vgl. ITWissen 2007).

### 4.1.2 Infotainmentgeräte bei BMW

BMW ist ein deutscher Automobilhersteller (Original Equipment Manufacturer (OEM[2])) (Generell) und bietet seit Jahren Infotainmentprodukte, die von Zulieferern (Tier-1, Tier-2 usw.) hergestellt werden, für Automobile an (spezifisches Produkt). BMW ist damit zur Software-Sekundärbranche zu zählen, da Software dort nicht Produkt, sondern unverzichtbarer Produktbestandteil ist (vgl. Liggesmeyer 2005a, S. 2).

Bei der Entwicklung von Infotainment-Geräten werden große Mengen von Software durch Tier-1s entwickelt, die vom OEM qualifiziert werden müssen. Dazu werden große Testkampagnen geleistet, Fehlerzustände gesammelt, Probleme analysiert und die Problembehebung ausgesteuert. Statistische Daten aus den Problemmelde- und Problemlösungsprozessen werden unter anderem zur Steuerung der Testkampagnen benutzt.

Zu den Infotainment-Produkten bei BMW gehören Internet, Email, Telefone, Online-Dienste, Freisprechanlagen, Schnittstellen, DVD-Player, Navigationssysteme und Kamerasysteme (vgl. Salzmann; Stauner 2005, Kap. 4 „Domäne Infotainment“). Die Bedienung erfolgt über eine Mensch-Maschine-Schnittstelle (human machine interface). Die Schnittstelle soll alle Funktionen erfüllen und gleichzeitig die Konzentration nicht zu sehr beeinträchtigen, hierzu ist auch die Sprachsteuerung wichtig (vgl. Salzmann; Stauner 2005, Kap. 4 „Domäne Infotainment“). Die technische Umsetzung erfolgt durch eine Head Unit (HU), eine Plattform für die verschiedenen Features. Sie ist das zentrale Steuergerät für die Mensch-Maschine-Schnittstelle, für Entertainment und Fahrerinformation sowie für die Kommunikation (vgl. folgende Abb. 4.1.-1).

---

[2] OEM sind Hersteller, die für ihre Produkte Komponenten anderer Hersteller einkaufen und diese in unveränderter Form in ihre eigenen Produkte integriert bzw. unter eigenem Namen auf den Markt weiterverkauft werden (vgl. Irlbeck 1998, S. 430).

**Abbildung 4.1-1 Head Unit im BMW-Automobil**


Quelle: Salzmann; Stauner 2005, Kapitel 4 „Domäne Infotainment“

## *4.2 Ziele und Fragestellungen der empirischen Analyse*

### 4.2.1 Ziele

Infotainment-Geräte sind zwar nicht so sicherheitskritisch wie z.B. Antiblockiersysteme in Automobilen, dennoch können Fehler in Infotainment-Geräten die Aufmerksamkeit des Fahrers im Straßenverkehr beeinträchtigen (vgl. auch Salzmann; Stauner 2005, Kap. 4 „Domäne Infotainment“). Die Messung der Software Reliability von eingebetteter Software von Infotainment-Geräten dient dazu, ein Qualitätskriterium von Software – nämlich die Zuverlässigkeit – zu messen. Zuverlässigkeit ist wichtig, da unzuverlässige Software Kundenunzufriedenheit auslösen und dadurch Kosten verursachen kann. Messungen zur Software Reliability können helfen, Test- und Fehlerfolgenkosten in einer Balance zu halten, da mit ihnen beispielsweise die aktuelle Zuverlässigkeit geschätzt und zukünftige Restfehler vorhergesagt werden können (vgl. Spillner et al. 2006, S. 71f). Gerade bei der Prüfung eingebetteter Software sollten Risiken, die von Restfehlern ausgehen, quantifiziert werden und damit dynamisches Testen und SRE kombiniert werden (vgl. Liggesmeyer 2005b, S. 221).

Im Rahmen der vorliegenden Masterarbeit werden die wichtigsten der oben genannten statistischen Modelle des Software Reliability Engineerings auf bei BMW vorliegende Daten angewendet und die Methodik auf ihre Aussagefähigkeit überprüft. Die gewonnenen Einsichten werden zu managementtauglichen Metriken verdichtet um Testkampagnen in Zukunft besser planen zu können.

Konkret geht es darum, geeignete SRE-Modelle auszuwählen, um sie auf die Daten anzuwenden, die zuerst aus dem Mercury Quality Center[3] extrahiert werden müssen. Welche Modelle im Einzelnen ausgewählt werden, hängt u.a. von der Beschaffenheit der vorliegenden Daten ab. Die Modelle werden anhand ihrer Vorhersagegüte bewertet. Neben den beschriebenen Modellen muss bei realen Fehlerkurven der Einfluss der Art und des Umfangs des Tests in verschiedenen Testinstanzen einbezogen werden.

### 4.2.2 Fragestellungen

Als Qualitätsmerkmal wurde die Zuverlässigkeit der getesteten Komponenten festgelegt (vgl. auch Kap. 2). Die wichtigsten Fragen, die somit in der vorliegenden Masterarbeit mithilfe des Einsatzes von Software Reliability Engineering beantwortet werden sollen, sind die Folgenden:

1. Welche SR-Modelle liefern für die jeweiligen Komponenten die besten Einschätzungen und Vorhersagen? Hierzu ist es notwendig, verschiedene Modelle zu vergleichen (zur Auswahl der Modelle siehe Kap. 3.3 und Kap. 4.5).
2. Liefern für die Komponenten auch Regressionen gute Ergebnisse? Hierzu wurden Regressionen mit SR-Modellen vergleichen.
3. Wie gut ist die Software (wie fehlerfrei)? Hierzu ist die Einschätzung der Restfehler notwendig.
4. Wann kann man mit dem Testen aufhören? (vgl. Farr 2002a) Dies kann man mit der Reliability Funktion beantworten.

Diese Fragestellungen werden in Kapitel 5 beantwortet. Insgesamt lässt sich sagen, dass es sich bei der Aufgabenstellung um einen innovativen und schwer zu bearbeitenden Bereich handelt, da stochastische Zuverlässigkeitsanalysen bisher „keine weite Verbreitung in der Praxis der Software-Entwicklung“ besitzen (Liggesmeyer 2002, S. 474).

[3] „Das Mercury Quality Center ist ein umfangreiches System für die projektbegleitende Qualitätssicherung eines Softwareproduktes.“ (Sneed; Baumgartner; Seidl 2007, S. 183).

## *4.3 Analyse der Datenausgangsbasis bei BMW*

Die Daten werden aus dem BMW Fehlermanagementsystem „Mercury „Quality Center“ extrahiert, das eine ActiveX Schnittstelle hat und auch Excel Exporte erlaubt. Sie stammen aus Abnahmetests (vgl. Kap. 2.2.3). Da es sich um sensible Daten handelt, müssen die Daten in Absprache mit BMW anonymisiert dargestellt werden, d.h. es können in der Arbeit keine absoluten Fehleranzahlen, sondern nur Prozentzahlen angegeben werden. Es kann jedoch festgehalten werden, dass pro analysierte Integrationsstufe (IS) weit mehr als die erforderliche Menge (nämlich fünfmal mehr als vorhandene Parameter im Modell, vgl. Kap. 3.3.1) an Daten in die Analyse mit einbezogen wurde.

### 4.3.1 Getestete Komponenten

Nach Absprache mit BMW werden Fehlerdaten aus vier Infotainmentgeräten analysiert. Es sind eine große Head Unit, eine mittlere Head Unit, eine kleine Head Unit sowie ein Telefoniesteuergerät zu bearbeiten. Eine große Head Unit umfasst mehrere komplexe Teilsysteme (z.B. Navigationssystem, Video, Radio). Eine mittlere Head Unit umfasst mehr als ein Radio aber weniger Teilsysteme als eine große Head Unit, eine kleine Head Unit umfasst noch weniger Teilsysteme. Das Telefoniesteuergerät ist ähnlich komplex wie die große Head Unit, es umfasst Audioplayer, Telematik, Sprachverarbeitung und Telefonie. Die vier getesteten Komponenten unterscheiden sich also in ihrer Komplexität und in der Anzahl ihrer Teilsysteme. Die Fehleranzahl lässt eine gute Aussagekraft erwarten.

### 4.3.2 Fehlermerkmale

Die vorliegenden Daten haben unter anderem folgenden Fehlermerkmale:

1. Indikation
   - TicketNo: Problem-Nr. bei BMW.
   - assigned ECU: Name des Problem verursachenden Steuergerätes (z.B. Head Unit).
   - Recorded date: Datum an dem das Problem im Mercury Quality Center erfasst wurde.
   - Problem finder: Name dessen, der das Problem entdeckt hat.
   - Analysis Team: Team welches das Problem entdeckt hat.

2. Klassifikation
   - Status BMW: Status der Problembearbeitung bei BMW, Workflow abhängig.
   - Involved I-Step: Integrationsstufe, in der das Problem aufgetreten ist.

   Der Entwicklungsprozess ist in Integrationsstufen organisiert, das bedeutet, dass zu festgelegten Meilensteinen der Umsetzungs-/Entwicklungsstand festgestellt wird.
   - Problem Severity: BMW Bewertungsindex (BI) (zur Einstufung des Problems).

   Es gibt bei BMW ein kundenorientiertes Bewertungssystem. Dieses Bewertungssystem wurde für die Masterarbeit derart umgesetzt, dass verschiedenen Beurteilungen je eine unterschiedliche Art der Fehler, die die Problemschwere signalisiert, zugeordnet wird. Es handelt sich um leichte, mittelschwere und schwere Fehler. In die Analyse gingen alle Fehler ein.
3. Problembeschreibung
   - Error Description: Beschreibung des Problems, das aufgetreten ist.

Für die Masterarbeit waren aus den oben genannten Fehlermerkmalen u.a. folgende relevant: assigned ECU, Recorded date, Involved I-Step sowie Problem Severity.

## *4.4 Modellierung der Testinstanzen*

In diesem Kapitel werden die Testinstanzen modelliert und erste Testmetriken über alle vorhandenen Daten hinweg im Durchschnitt erstellt. Neben den Ergebnissen der SRM (vgl. Kap. 5) können auch diese Testmetriken für die Testplanung zusätzliche Informationen bereitstellen. Hierzu gehört unter anderem die Analyse der Verteilung der Fehler nach Problemschwere im Zeitverlauf.

### 4.4.1 Auswahl der Datenbasis für die weitere Analyse

Die Modellierung der Testinstanzen ist notwendig, um einen Überblick über die vorliegenden Daten zu erhalten und die Daten für die weiteren Analysen auszuwählen, da für den Einsatz von SRM umfangreiche Voranalysen nötig sind. Insgesamt wurden mehr als 50 Integrationsstufen voranalysiert und dann 11 Integrationsstufen aus den vier verschiedenen Komponenten (vgl. Kap. 4.3.1) exemplarisch für die detaillierte Analyse in dieser Arbeit ausgewählt. Da die Art der Fehlerdaten (wie z.B. Intervallgruppierung) den Einsatz von NHPP-Modellen erforderte (vgl. Kap. 3.3.1.1) ging es bei der Detailauswahl der Datenbasis darum, Kurven zu analysieren, die sich tatsächlich für den Einsatz von NHPP-Modellen eignen.

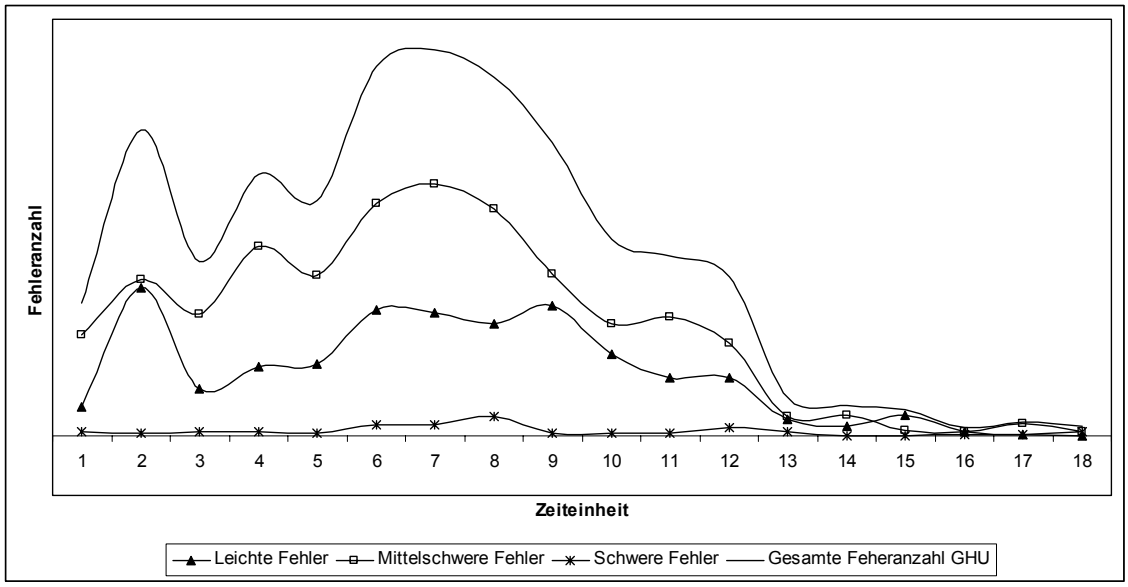
Bei allen ausgewählten Komponenten bestehen die Integrationsstufen aus 26 Zeiteinheiten, aus diesem Grund werden immer jeweils diese 26 Zeiteinheiten analysiert.

### 4.4.2 Problemschwere, Anzahl der Tester und Testintensität

Von allen untersuchten Fehlern liegt die Anzahl der leichten Fehler im Durchschnitt bei 48.80%, bei 50.29% für die mittelschweren Fehler und bei 0,91% für die schweren Fehler (vgl. zum Vergleich der Komponenten Kap. 5.2.1).

In der folgenden Abbildung 4.4-1 sieht man exemplarisch für die große Head Unit leichte, mittelschwere und schwere Fehler sowie die Gesamtzahl der Fehler. Hier zeigt sich, dass 34.8% der gesamten Fehleranzahl leicht war und 62.6% der gesamten Fehleranzahl mittelschwer. Der Anteil der schweren Fehler war 2.6%.

**Abbildung 4.4-1 Darstellung der Fehler nach Problemschwere GHU**




Je nach Zielen z.B. für das Testende, kann man die Kurven der einzelnen Fehlerarten genauer analysieren. D.h. man könnte auch z.B. für die mittelschweren und schweren Fehler ohne die leichten Fehler SRM-Berechnungen durchführen. In der vorliegenden Arbeit werden jedoch alle Fehlerarten zusammen betrachtet, weil die Verteilung der Anteile der Problemschwere bei den vier Komponenten unterschiedlich ist (siehe Tab. 5.2-1 „Anteil der Fehlerarten nach Problemschwere für die Komponenten“). Durch einen Einbezug aller Fehlerarten kann hier eine bessere Vergleichbarkeit gewährleistet werden.

Des Weiteren wurde auch die Anzahl der Tester modelliert, hier hat sich gezeigt, dass in 80% der Fälle je mehr Tester eingesetzt werden desto mehr Fehler werden gefunden aber nicht in einer mäßigen Proportion, da man zwischen erfahrenen und nicht erfahrenen Testern unterscheiden muss.

### 4.4.3 Erfassen der Fehler in Abhängigkeit von der Zeit

Da die Daten zur Fehlererfassung nur den Tag, an dem die Fehler erfasst wurden, vermerken und nicht die genaue Uhrzeit, können keine Modelle eingesetzt werden, die auf der Time-Domain-Data Methode beruhen (siehe Kap. 3.3.1.1). Deswegen wurden in Absprache mit BMW die erfassten Fehler pro ZE gruppiert. D.h. es wird mit der Intervall-Domain-Data Methode gerechnet. Zudem habe ich die Kalenderwochen 1 und 52 nicht einzeln aufgeführt (Weihnachtsferien/Produktionsunterbrechung), sondern mit der entsprechenden Zeiteinheit zusammengefasst.

### 4.4.4 Fehleranzahl

Am Anfang der Bewertung der Daten wurden alle Fehler ausgewählt, unabhängig von ihrer Integrationsstufe. Um die SRM einzusetzen, war es danach notwendig, die Fehleranzahl in den entsprechenden Integrationsstufen zu berücksichtigen.

Im Rahmen meiner Masterarbeit wende ich die SRE-Modelle auf die Daten an. Um dieses Verfahren zu ermöglichen, musste ich die Fehleranzahl pro Integrationsstufe auswählen. In Kapitel 5.1 werde ich die verschiedenen Fehleranzahlentwicklungen als Beispiele betrachten.

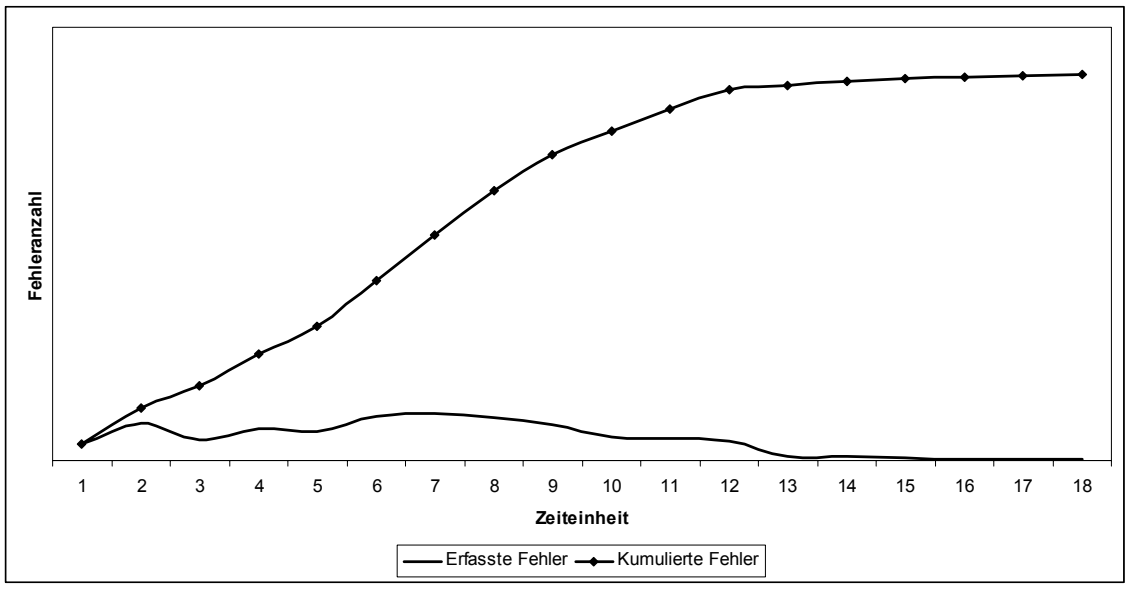
## *4.5 Prüfung der Daten auf Einsetzbarkeit in Modellen des Software Reliability Engineering und Modellauswahl*

Wie in Kapitel 3.3 beschrieben, umfasst die Modellauswahl zwei Schritte. In diesem Kapitel soll der erste Schritt beschrieben werden, der die Auswahl der Modelle vor Anwendung der Daten beschreibt. Durch die Analyse der Datenausgangsbasis bei BMW stellte sich heraus, dass es sich um intervallgruppierte Daten handelt, d.h. es können keine Modelle eingesetzt werden, die auf der Time-Domain-Data Methode beruhen (siehe Kap. 3.3.1.1). Es muss die Intervall-Domain-Data Methode angewandt werden, bei der die gefundenen Fehler pro Intervall gruppiert werden. Diese Anforderung erfüllen die NHPP-Modelle, da sie eine Mittelwertfunktion haben, welche definiert ist, als die erwartete Fehlerwirkungsanzahl bis zu einem gegebenen Zeitpunkt

(vgl. Xie 1991, S. 25). Da es sich bei den vorliegenden Daten um Daten aus einem Testprozess „with fault removal“ handelt, können keine Modelle eingesetzt werden, die auf Homogeneous Poisson Prozessen beruhen, sondern es können NHPP-Modelle eingesetzt werden (siehe Kap. 3.1 und Voraussetzungen in Kap. 3.2.3). Zudem kann man bei den getesteten Komponenten annehmen, dass es endliche und nicht unendliche Fehlerwirkungen gibt, da die Komponenten nicht eine unendliche Zeit benutzt werden. Dies spricht auch für den Einsatz von NHPP-Modellen, die finite failure Modelle sind (siehe Kap. 3.3.1.3).

Um aus den verschiedenen NHPP-Modellen Modelle für die Anwendung auf die Daten auszuwählen, wurden die Daten zuerst mit Excel grafisch analysiert (vgl. folgende Abb. 4.5-1). Hierzu wurden die Fehler und die kumulierten Fehler dargestellt. Aus der Abbildung 3.2-1 „Die S-Form der Fehlerwirkungsintensitätsfunktion“ geht hervor, dass es eine Einschwingphase beim Testen gab.

**Abbildung 4.5-1 Grafische Darstellung zwischen erfassten und kumulierten Fehlern**




Quelle: Aus Datenanalyse bei BMW

Die obige Abbildung 4.5-1 wurde danach mit den beiden Modellen Goel-Okumoto und Yamada-Delayed-S-shaped (vgl. Abb. 3.2.-1 „Die S-Form der Fehlerwirkungsintensitätsfunktion“) verglichen. Da der Verlauf der Kurven ähnlich erscheint, werden diese beiden Modelle ausgewählt, um sie auf die Daten anzuwenden. Welches der beiden Modelle besser geeignet ist, wird sich nach Anwendung der Modelle in einem zweiten Schritt (vgl. Kap. 5) zeigen. Da es für reale Projekte und später für die Praxis konzeptuell und wirtschaftlich nicht praktikabel ist, mit mehr als

ein bis zwei Modellen zu arbeiten (vgl. Kap. 3.1), werden die oben genannten beiden Modelle ausgewählt.

## *4.6 Software Reliability Engineering Werkzeuge (Tools)*

Liggesmeyer (2005b, S. 221) betont, dass eine erfolgreiche Anwendung von Zuverlässigkeitsmodellen die Unterstützung durch ein leistungsfähiges Werkzeug voraussetzt. Zur Anwendung der SRE-Modelle gibt es verschiedene Software zur Unterstützung, beispielsweise die Software SMERFS^3[4]. Diese integriert mehrere Modelle, die auf unterschiedlichen Daten basieren (vgl. Farr 2002a). Sechs Modelle nutzen als Input-Daten die Zeit zwischen dem Auftreten von Fehlern (Time-Domain): Littlewood und Verrall's Bayesian Modell, Moranda's Geometric Modell, John Musa's Basic Execution Time Modell, John Musa's the Logarithmic execution time Poisson Modell, the Jelinski-Moranda Modell, eine Adaption des Goel's Non-Homogeneous Poisson Prozessmodells für „time between error" Daten. Fünf nutzen die Zahl der entdeckten Fehler pro Testperiode (Interval-Domain): das Generalized Poisson Modell, Goel's NHPP Modell, Brooks and Motley's Modell, Yamada's S-shaped Growth Modell, Norman Schneidewind's Modell.

Die Tabelle A-1 „Übersicht über weitere SRE-Software" im Anhang A gibt eine Übersicht über zwei weitere SRE-Tools. Lyu (1996, S. 729ff.) beschreibt ausführlich verschiedene SRE-Tools. Die Software SMERFS^3 wird in der Masterarbeit eingesetzt, da die beiden ausgewählten Modelle der Nonhomogeneous Poisson Prozessmodelle dort zur Verfügung stehen. Die Software wurde und wird u.a. von der NASA bei sensiblen Projekten eingesetzt. Dies bedeutet, dass die Software anerkannt ist, sich für sensible Daten eignet und auch im wissenschaftlichen Bereich geschätzt wird (vgl. Lyu 1996).

Es werden SMERFS^3 und Excel kombiniert eingesetzt, was auch als besonders effektiv in der Literatur beschrieben wird (vgl. Wallace; Liang 2002, S. 10). SMERFS^3 bietet die SRM mit Schätzungen und Vorhersagen, Excel wiederum ermöglicht statistische Analysen (z.B. Regressionen) und benutzerdefinierte Grafiken. Das Werkzeug SMERFS^3 braucht eine Input-Datei in der Form <Dateiname>.s3d und sie sieht wie folgt aus:

---

[4] SMERFS^3 steht für Statistical Modeling and Estimation of Reliability Functions for Software: Software, Hardware und Systems

**Abbildung 4.6-1 Beispiel Intervall-Domain-Data für SMERFS^3**

```
1 1.
6 1.
7 1.
3 1.
7 1.
5 1.
0 1.
3 1.
1 1.
```

Die Abbildung 4.6-1 zeigt ein Beispiel von einer Input-Datei für neun Intervalle von je einer Zeiteinheit (ZE)[5]. Die siebte ZE hat keine Fehlerwirkung und sie muss daher so dargestellt werden. Die Fehlerwirkungsanzahl ist in der ersten Spalte und die Intervallzahl ist in der zweiten Spalte dargestellt. In diesem Fall bedeutet die Nummer „1." eine ZE (vgl. Wallace 2001). Mehr über SMERFS^3 findet man in NAVSEA (2002).

Gemäß Farr (1996, S. 95) sind in SMERFS^3 das Yamada-Delayed-S-shaped Modell (dort benannt als Yamadas-Modell) und das Goel-Okumoto Modell (dort benannt als NHPP-Modell) integriert. Bei beiden handelt es sich um NHPP-Modelle (vgl. Kap. 3.2.3).

## *4.7 Zusammenfassung*

In diesem Kapitel wurde das Anwendungsfeld – Infotainment in der Automobilbranche, speziell bei BMW – beschrieben. Danach wurden die Ziele und Fragestellungen der vorliegenden Masterarbeit erläutert. Darauf folgten die Analyse der Datenausgangsbasis bei BMW sowie die Modellierung der Testinstanzen. Aufbauend auf diese Voranalysen wurden die Daten ausgewählt sowie die passenden Modelle. Die Daten werden mit dem Yamada-Delayed-S-shaped Modell und dem Goel-Okumoto Modell analysiert. Abschließend wurden Werkzeuge (Tools) gesichtet und SMERFS^3 ausgewählt, um die SRM-Berechnungen durchzuführen.

[5] Eine Zeiteinheit entspricht einer bestimmten Anzahl von Kalenderwochen. Die genaue Zuordnung von Zeiteinheiten zu Kalenderwochen kann hier aufgrund der Sensibilität der Daten nicht angegeben werden.

# 5 Ergebnisse der empirischen Analyse

In Kapitel 5.1 werden die ausgewählten SRM (Goel-Okumoto und Yamada-Delayed-S-shaped) für die vier Geräte/Komponenten (vgl. Kap. 4.3.1) eingesetzt und spezifische Testmetriken erstellt. Kapitel 5.2 widmet sich dem Vergleich und der Interpretation der Ergebnisse. In diesem Kapitel geht es also darum, sowohl Fallanalysen einzeln vorzustellen, als auch die verschiedenen exemplarischen Beispiele miteinander zu vergleichen, um damit Ähnlichkeiten und Unterschiede in den Schätzungen und Vorhersagen herauszuarbeiten. Hieraus kann abgeleitet werden, welches SRM am besten passt und in welchen Fällen der Einsatz von SRM im Vergleich mit einfachen Regressionen besser geeignet ist. Das Kapitel 5.3 beantwortet zusammenfassend die Fragestellungen aus Kapitel 4.2.2.

## *5.1 Ergebnisse der Software Reliability Analysen*

Im Folgenden wird die Anwendung der Software Reliability Modelle auf die vier getesteten Komponenten dargestellt. Bei der Analyse der Komponenten wurden unterschiedliche Integrationsstufen miteinbezogen, insgesamt 11 Integrationsstufen. Es gibt Integrationsstufen, die bereits abgeschlossen waren und solche, die noch nicht abgeschlossen sind. In diesem Kapitel werde ich in 5.1.1 eine nicht abgeschlossene Integrationsstufe exemplarisch untersuchen und ab 5.1.2 abgeschlossene Integrationsstufen untersuchen. Im folgenden Kapitel 5.1.1 werde ich exemplarisch die große Head Unit als Beispiel genau untersuchen. Pro Komponente wird jeweils nur eine Integrationsstufe ausführlicher behandelt, in Kapitel 5.2 – dem Vergleich – werden jedoch auch die anderen Integrationsstufen der Komponenten mittlere und kleine Head Unit sowie Telefoniesteuergerät ergänzend miteinbezogen[6]. Die Grafiken zu diesen anderen Integrationsstufen sind im Anhang C dargestellt.

### 5.1.1 Analyse der Komponente „große Head Unit“

Wie oben erklärt, wird eine nicht abgeschlossene Integrationsstufe (reale Daten von der 1 bis 18 ZE) untersucht. Hier kann kein Vergleich mit den realen Daten stattfinden, da Daten von „19 bis 26“ ZE noch nicht zur Verfügung stehen. Dagegen kann man eine

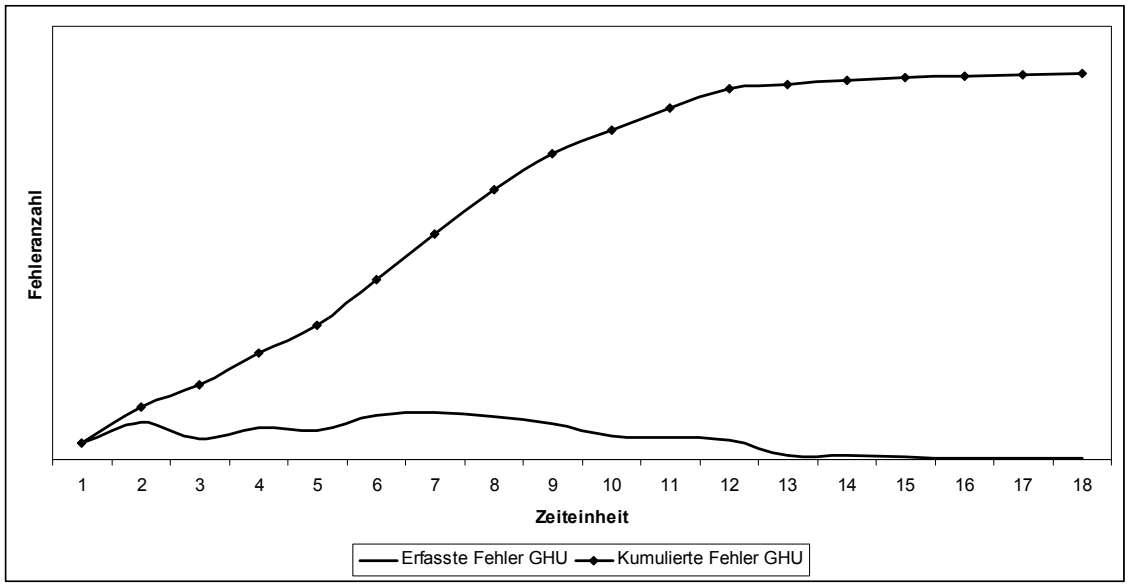
[6] Die einzelnen Integrationsstufen wurden mit A, B, C etc. benannt.

Prognose der Restfehler sowohl mit SRE-Modellen als auch mit einfachen Regressionen berechnen.

#### *5.1.1.1 Analyse der Fehlerdaten mit Software Reliability Modellen*

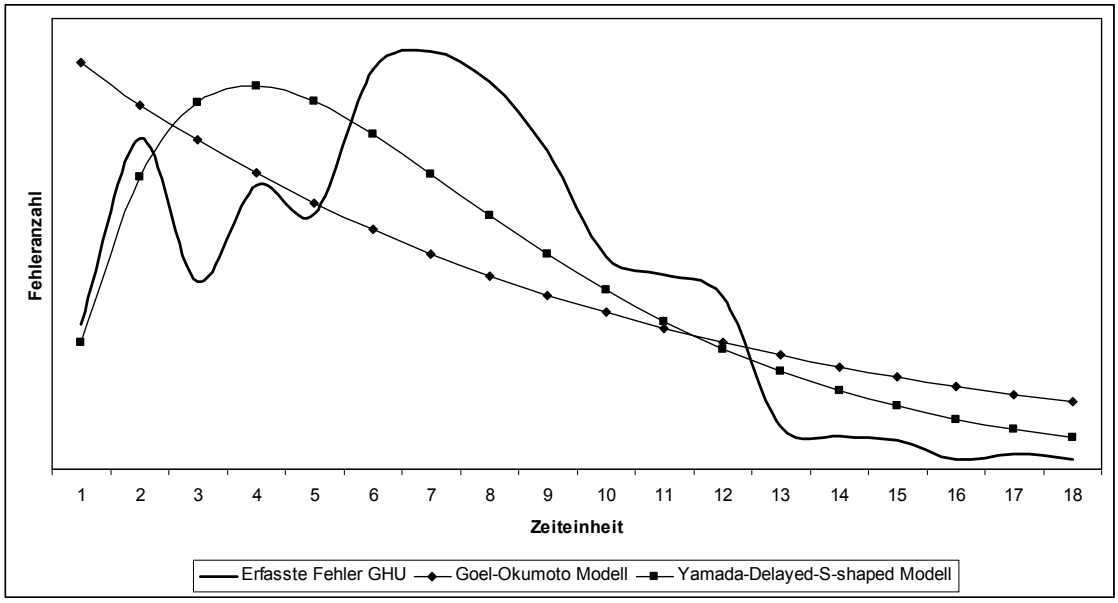
Um die Daten zu analysieren zeigt die Abbildung 5.1-1 zunächst die Kurve der erfassten Fehler und die der kumulierten Fehler. Die Kurve der kumulierten Fehler hat eine deutliche S-shaped-Form. Die erfassten Daten wurden als Inputdatei in SMERFS^3 eingetragen (zum Aussehen der Inputdatei siehe Kap. 4.6). Hierzu wurden die insgesamt erfassten ZE 1 bis 18 ausgewählt. Aufgrund dieser Daten wurde dann die Vorhersage für die ZE 19 bis 26 durchgeführt, für die noch keine realen Daten vorliegen. Da es sich bei den erfassten Fehlerdaten um Fehlerdaten von einer Integrationsstufe, die noch nicht abgeschlossen war, handelt, werden die vorhergesagten Daten mit den realen Danten nach Abgabe meiner Masterarbeit verglichen werden können.

**Abbildung 5.1-1 Fehlerkurve und kumulierte Fehlerkurve GHU**



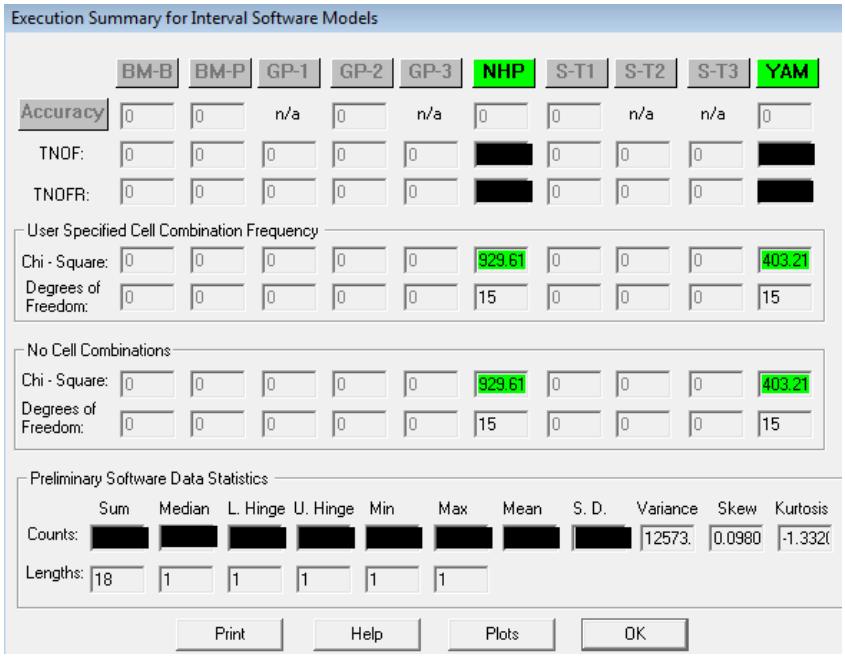

Die Inputdatei wurde dann in SMERFS^3 mit den Goel-Okumoto und Yamada-Delayed-S-shaped Modellen durchgeführt. In der folgenden Abb. 5.1-2 sind die realen Daten als „erfasste Fehler GHU“ eingetragen. Die geschätzte Kurve mit dem Yamada-Delayed-S-shaped Modell und mit dem Goel-Okumoto-Modell ist ebenfalls in der Abbildung eingetragen.

**Abbildung 5.1-2 Berechnung der NHPP Modelle mittels SMERFS^3 GHU**[7]




Wie man aus dem folgenden Screenshot in Abbildung 5.1-3 sieht (einige Felder sind aufgrund der Vertraulichkeit der Daten schwarz eingefärbt) erhält man sowohl mit dem Yamada-Delayed-S-shaped Modell als auch mit dem Goel-Okumoto-Modell (NHPP-Modell) gute Ergebnisse (grün markiert im Feld „Chi-Square“ in der Abb. 5.1-3).

**Abbildung 5.1-3 Screenshot SMERFS^3 GHU**




[7] Die in dieser Abbildung dargestellte Kurve der erfassten Fehler entspricht der Kurve der erfassten in der vorherigen Abbildung, nur der Maßstab ist anders.

Man sieht in der Abbildung 5.1-3, dass die Werte des Chi-Quadrat 929.81 für Goel-Okumoto und 403.21 für Yamada-Delayed-S-shaped Modell betragen. Jeweils haben die Modelle den Wert 15 als Freiheitsgrad. Beide Modelle sind in der oben genannten Abbildung 5.1-3 grün markiert und dies bedeutet, dass die Modelle zu den Inputdatei-Werten im Prinzip passen würden, da der Chi-Quadrat-Test die Güte der Anpassung der theoretischen an die empirische Verteilung darstellt. Aufgrund der Anonymisierung werde ich in der Masterarbeit keine exemplarische Berechnung von Chi-Quadrat darstellen, da die reale Fehleranzahl von jeder Zeiteinheit und die erwarteten Fehleranzahl eingegeben werden müssten (siehe Kap. 3.3.2).

Bei den SR-Modellen ist der Vergleich des Chi-Quadrat-Wertes mit der Chi-Quadrat-Tabelle (vgl. Bortz 1993, S. 699f.) nicht immer das ausschlaggebende Kriterium, ob die geschätzte Verteilung mit der realen übereinstimmt. Man sieht das man daran, dass SMERFS^3 ein Chi-Quadrat von „403“ als eine erfolgreiche Passung deklariert (siehe Abb. 5.1-3). Dies wird erklärlich, wenn man bedenkt, dass die Größe der Abweichungen zwischen beobachteten und geschätzten Fehlern sowie die Anzahl der Intervalle einen Einfluss auf die Größe des Chi-Quadrat-Wertes haben.

In der folgenden Tabelle 5.1-1 sind einige Schätzungen und Prognosen für die beiden Modelle dargestellt. Die Tabelle 5.1-1 ist in vier Quadranten unterteilt. Der linke obere Quadrant zeigt die Wahrscheinlichkeit der Fehlerentdeckung (P). Dabei bedeutet 0.29 bei Yamada-Delayed-S-shaped Modell, dass zu 29% weitere Fehler entdeckt werden. Die zwei verschiedenen Konfidenzintervalle bedeuten, dass für die Vorhersagen zwei unterschiedlich lange Datenreihen benutzt wurden, d.h. eine kürzere und eine längere. Der rechte obere Quadrant zeigt die gesamte Anzahl von Fehlern in der Software an, wie sie vom Modell berechnet wird. Zur Anonymisierung der Daten wurde nicht die Zahl angegeben, sondern die Prozentzahl. 100% entsprechen der Gesamtzahl der real gefundenen Fehler bis zur ZE 18 (für eine Lesehilfe vgl. Anhang B). In der Tabelle 5.1-1 sieht man im oberen rechten und unteren linken Quadranten, dass das Yamada-Delayed-S-shaped Modell als Gesamtfehleranzahl 103% einschätzt, d.h., dass es annimmt, dass in der ZE 18 noch 3% der Gesamtfehler in der Software sind (bei Goel-Okumoto sind es 117% als Gesamtfehleranzahl bzw. damit 17% Restfehler). Der rechte untere Quadrant zeigt Chi2 (Chi-Quadrat) (wie oben erklärt). Darüber hinaus kann man mit dem Modell für jede beliebige Zeitspanne vorhersagen lassen, wie viele Fehler voraussichtlich in dieser Zeitspanne gefunden werden. Im vorliegenden Beispiel wurde

die Zeitspanne von 10 ZE gewählt. Hier zeigt sich, dass bis zur 10. ZE voraussichtlich 91% der Restfehler für Yamada-Delayed-S-shaped Modell gefunden werden.

**Tabelle 5.1-1 Einschätzung und Prognose für eine nicht abgeschlossene IS GHU**

| | **Niedrigeres Konfidenz-intervall[8]** | **P** | **Höheres Konfidenz-intervall[9]** | **Niedrigeres Konfidenz-intervall** | **TNF** | **Höheres Konfidenz-intervall** |
|---|---|---|---|---|---|---|
| **Goel** | 0.09 | 0.10 | 0.11 | 112% | 117% | 122% |
| **YAM** | 0.28 | 0.29 | 0.30 | 100% | 103% | 107% |
| | **Niedrigeres Konfidenz-intervall** | **TNFR[10]** | **Höheres Konfidenz-intervall** | **CHI2** | **Pred** | |
| **Goel** | 12% | 17% | 22% | 929.61 | 65% | |
| **YAM** | 0 | 3% | 7.18% | 403.21 | 91% | |

P: Wahrscheinlichkeit der Fehlerendeckung

Pred: Im vorliegenden Fall: eingeschätzte Restfehleranzahl für die nächsten 10 Zeiteinheiten in Prozent der geschätzten Gesamtanzahl der Restfehler

TNF (total number of faults): Gesamtumfang der geschätzten Fehleranzahl in Prozent (im Vergleich mit realen Daten)

TNFR (total number of faults remaining): Restfehleranzahl in Prozent (Vergleich mit realen Daten)

Chi2: Chi-Quadrat Wert

In der folgenden Tabelle 5.1-2 werden die Vorhersagen dargestellt, wie viele Fehler in Prozent der übriggebliebenen Restfehler in einer ZE voraussichtlich gefunden werden (z.B. werden in ZE 19 voraussichtlich 21.74% der Restfehler gefunden werden). Darüber hinaus wird die Reliability angezeigt (vgl. Kap. 3.2.3.3). Eine Reliability von 0.93 in der 23. ZE bedeutet: Die Wahrscheinlichkeit, dass die Software in der 23. ZE einen festgelegten Zeitraum (0.01)[11] ohne Fehlerwirkung überlebt, liegt bei ca. 93%. Wäre der festgelegte Zeitraum beispielsweise ein Tag, hieße das, dass, wenn man nur bis zu 19. ZE getestet hätte, die Software zu 84% einen Tag fehlerfrei laufen würde. Dieser Wert würde sich um 19% erhöhen, wenn man bis zur 23. ZE weiter testen würde.

[8] In SMERFS als LCI genannt

[9] In SMERFS als UCI genannt

[10] TNFR wird wie folgt berechnet: (TNFR/TNF)*100

[11] Hier wird eine gewisse Zeit (z.B. Tage) der Zeiteinheit genommen um die Reliability zu messen und für einen bestimmten Zeitraum (z.B. ein Tag) das Testende zu bestimmen.

Diese Daten mit den Daten der obigen Tabelle 5.1-1 stellen wichtige Informationen dar, wann man mit dem Testen aufhören könnte. Beispielsweise könnte das vorher im Testkonzept festgelegte Ziel sein, mit dem Testen aufzuhören, wenn 90% aller eingeschätzten Restfehler gefunden wurden (ebenso könnte man dies als Anteil der Gesamtfehler berechnen). Im vorliegenden Beispiel würde man also nach ZE 21 mit dem Testen aufhören.

**Tabelle 5.1-2 Vorhersage der Restfehler und der Reliability für IS GHU**

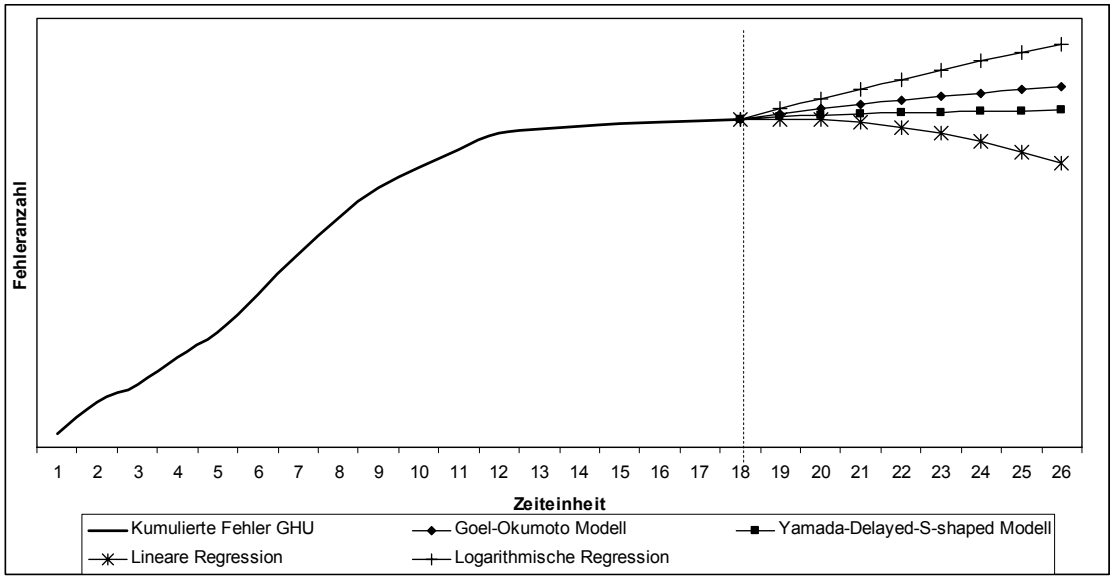
| **ZE** | **Anteil an Restfehlern Vorhersage** | **Reliabilty R(0.01) des Yamada-Delayed-S-shaped-Modells** $R(x/t) = e^{-a\left[(1+bt)e^{-bt} - (1+b(t+s))e^{b(t+s)}\right]}$ |
|---|---|---|
| 19 | 21.74% | 0.84 |
| 20 | 17.17% | 0.87 |
| 21 | 13.73% | 0.90 |
| 22 | 10.3% | 0.92 |
| 23 | 8.01% | 0.93 |

### *5.1.1.2 Analyse der Restfehlerdaten mit Regression*

Bivariate Regressionen dienen der Beschreibung des stochastischen Zusammenhangs zwischen zwei Variablen und der Vorhersage der Werte einer Variablen durch die andere Variable (vgl. Bortz 1993, S. 166f.). Dieses Beispiel wird auch mit Regressionen berechnet[12]. Es geht darum, eine Vorhersage zu treffen und zu prüfen, ob eine Vorhersage nicht nur mit SR-Modellen, sondern auch mit einfachen Regressionen möglich ist. In der folgenden Abbildung 5.1-4 sieht man die kumulierte Fehlerkurve bis zur ZE 18, ab der gestrichelten Linie sieht man die Schätzungen der verschiedenen Modelle und Regressionen. Es wird deutlich, dass die logarithmische Regression ab ZE 19 mehr Restfehler als die Goel-Okumoto und Yamada-Delayed-S-shaped Modelle vorhersagen. Für die lineare Regression sind alle Restfehler in der ZE 19 gefunden. Welche Kurve am besten zu den realen Daten passen würde, kann aufgrund der noch nicht vorliegenden Daten hier nicht entschieden werden. Am wahrscheinlichsten sind jedoch die Kurven der beiden SR Modelle. In Kapitel 5.3 wird das Ergebnis weitergehend interpretiert.

[12] Die Regressionsfunktion kann hier nicht angegeben werden, da sonst die Anonymität der Daten gefährdet wäre.

**Abbildung 5.1-4 Analyse der Daten mit Regressionen und NHPP-Modellen GHU**

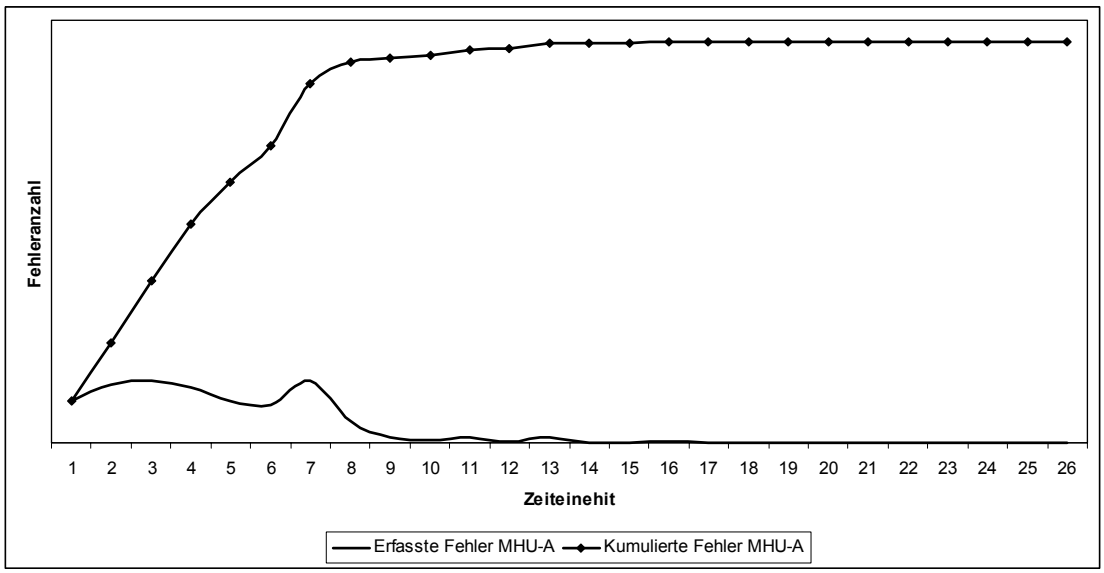



## 5.1.2 Analyse der Komponente „mittlere Head Unit“

### *5.1.2.1 Analyse der Fehlerdaten mit Software Reliability Modellen*

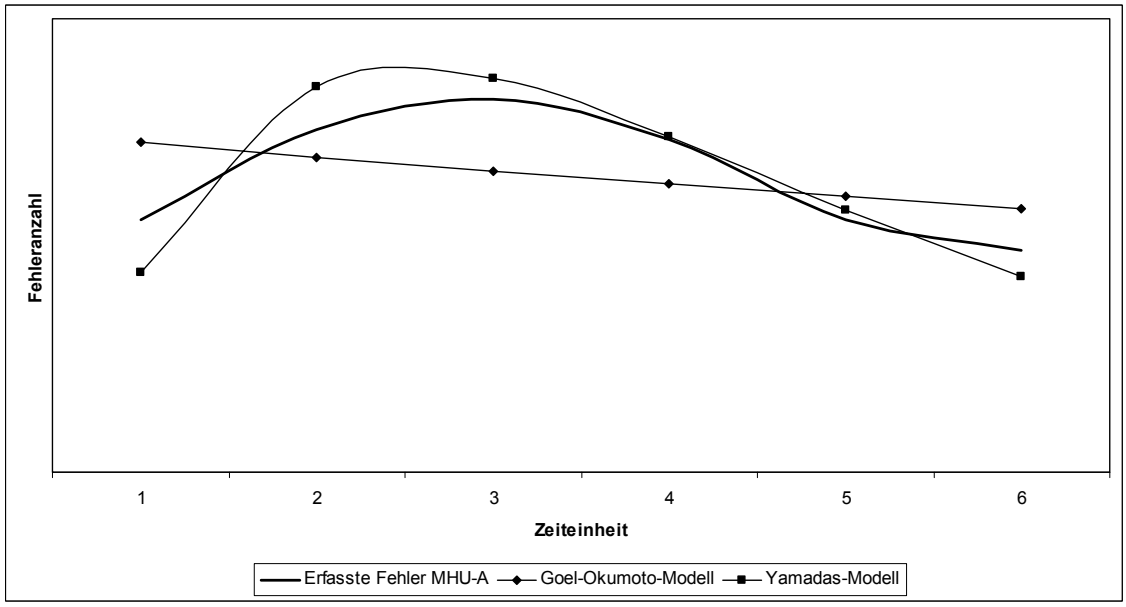
Um die Daten zu analysieren zeigt die Abbildung 5.1-5 die Kurve der kumulierten Fehler. Die Kurve der kumulierten Fehler könnte sowohl auf eine Goel-Okumoto- wie auf eine S-shaped-Form hindeuten (siehe Kap. 3.2.3). Die Daten der erfassten Kurve wurden als Inputdatei in SMERFS^3 eingetragen (zum Aussehen der Inputdatei siehe Kap. 4.6). Hierzu wurden die ZE 1 bis 6 für die Kurve MHU-A (A entspricht einer ausgewählten Integrationsstufe, die sich besonders für die SR-Analyse eignete) ausgewählt. Es wurden mit Absicht nicht alle vorhandenen Daten bis ZE 26 eingetragen, um die erfassten Daten zwischen ZE 7 bis ZE 26 mit der Vorhersage (auf Basis der ZE 1 bis 6) für diesen Zeitraum vergleichen zu können (vgl. zu diesem Vorgehen auch Wallace 2001). Durch diesen Vergleich der realen und vorhergesagten Daten zeigt sich die Genauigkeit der Vorhersage des Modells.

**Abbildung 5.1-5 Fehlerkurve und kumulierte Fehlerkurve MHU-A**




Die Input-Daten wurden dann in SMERFS^3 mit den Goel-Okumoto und Yamada-Delayed-S-shaped Modellen durchgeführt. In der folgenden Abbildung 5.1-6 sind die realen Daten als „erfasste Fehler MHU“ eingetragen. Die geschätzte Kurve mit dem Yamada-Delayed-S-shaped Modell und mit dem Goel-Okumoto Modell ist ebenfalls in der Abbildung eingetragen.

**Abbildung 5.1-6 Berechnung der NHPP Modelle mittels SMERFS^3[13] MHU-A**




[13] Die in dieser Abbildung dargestellte Kurve der erfassten Fehler entspricht der Kurve der erfassten in der vorherigen Abbildung, nur der Maßstab ist anders.

In der folgenden Tabelle 5.1-3 sind einige Schätzungen und Prognosen für die beiden Modelle für die kumulierten Fehler der Kurve MHU-A (Gesamte Schätzung und Prognose der beiden Modelle für alle Integrationsstufen in Kap. 5.3) dargestellt. Die Tabelle 5.1-3 ist in vier Quadranten unterteilt. Der linke obere Quadrant zeigt die Wahrscheinlichkeit der Fehlerentdeckung (P). Dabei bedeutet 0.5 bei Yamada-Delayed-S-shaped Modell, dass zu 50% weitere Fehler entdeckt werden. Die zwei verschiedenen Konfidenzintervalle bedeuten, dass für die Vorhersagen zwei unterschiedlich lange Datenreihen benutzt wurden, d.h. eine kürzere und eine längere. Der rechte obere Quadrant zeigt die gesamte Anzahl von Fehlern in der Software an, wie sie vom Modell berechnet wird. Zur Anonymisierung der Daten wurde nicht die Zahl angegeben, sondern die Prozentzahl. 100% entsprechen bei diesem und den folgenden Beispielen der Gesamtzahl der real gefundenen Fehler bis zur ZE 26. Prozentzahlen in der Nähe von 100% zeigen an, dass das Modell sehr gut passt. In der Tabelle 5.1-3 sieht man, dass die durch das Yamada-Delayed-S-shaped Modell eingeschätzte Gesamtfehleranzahl mit 92% sehr nahe der realen Anzahl kommt (bei Goel-Okumoto ist das mit 313% nicht der Fall). Der linke untere Quadrant zeigt die geschätzte Restfehleranzahl in Prozent der realen Restfehler bis zur ZE 26 an. Der rechte untere Quadrant zeigt Chi2 (Chi-Quadrat) (wie oben erklärt). Darüber hinaus kann man mit dem Modell für jede beliebige Zeitspanne vorhersagen lassen, wie viele Fehler voraussichtlich in dieser Zeitspanne gefunden werden. Im vorliegenden Beispiel wurde die Zeitspanne von 10 ZE gewählt. Hier zeigt sich, dass bis zur 10. ZE voraussichtlich 98% aller Restfehler gefunden werden.

**Tabelle 5.1-3 Einschätzung und Prognose für eine abgeschlossene IS MHU-A**

| | Niedrigeres Konfidenz-intervall | P | Höheres Konfidenz-intervall | Niedrigeres Konfidenz-intervall | TNF | Höheres Konfidenz-intervall |
|---|---|---|---|---|---|---|
| **Goel** | 0 | 0.04 | 0.10 | 74% | 313% | 687% |
| **YAM** | 0.43 | 0.50 | 0.57 | 80% | 92% | 104% |
| | **Niedrigeres Konfidenz-intervall** | **TNFR** | **Höheres Konfidenz-intervall** | **CHI2** | **Pred** | |
| **Goel** | undef. | 926% | 2365% | 10.54 | 36% | |
| **YAM** | 24% | 70% | 117% | 4.62 | 98% | |

P: Wahrscheinlichkeit der Fehlerendeckung
Pred: Im vorliegenden Fall: eingeschätzte Restfehleranzahl für die nächsten 10 Zeiteinheiten in Prozent der geschätzten Gesamtanzahl der Restfehler.
TNF(total number of faults): Gesamtumfang der geschätzten Fehleranzahl in Prozent (im Vergleich mit realen Daten)
TNFR (total number of faults remaining): Restfehleranzahl in Prozent (Vergleich mit realen Daten)
Chi2: Chi-Quadrat Wert

In der folgenden Tabelle 5.1-4 werden die Vorhersagen dargestellt, wie viele Fehler in Prozent der übriggebliebenen Restfehler in einer ZE voraussichtlich gefunden werden (z.B. werden in ZE 7 voraussichtlich 30.9% der Restfehler gefunden werden). Darüber hinaus wird die Reliability angezeigt (vgl. Kap. 3.2.3.3). Eine Reliability von 0.79 in der 7 ZE bedeutet: Die Wahrscheinlichkeit, dass die Software einen Zeitraum (0.01 einer Zeiteinheit) ohne Fehlerwirkung überlebt, liegt bei ca. 79%.

Diese Daten mit den Daten der obigen Tabelle (siehe Tab. 5.1-3) stellen wichtige Informationen dar, wann man mit dem Testen aufhören könnte. Beispielsweise könnte das vorher festgelegte Ziel sein, mit dem Testen aufzuhören, wenn 90% aller eingeschätzten Restfehler gefunden wurden (ebenso könnte man dies als Anteil der Gesamtfehler berechnen). Im vorliegenden Beispiel würde man also nach ZE 9 mit dem Testen aufhören.

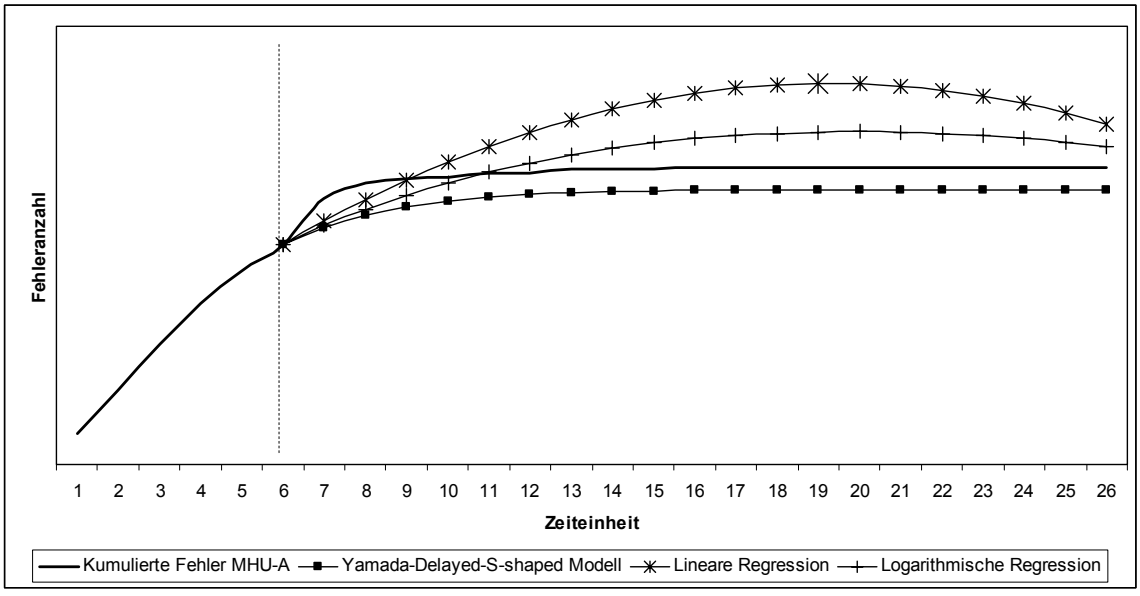
**Tabelle 5.1-4 Vorhersage der Restfehler und der Reliability für IS MHU–A**

| ZE | Anteil an Restfehlern Vorhersage | Reliabilty R(0.01) des Yamada-Delayed-S-shaped-Modells $R(x/t) = e^{-a\left[(1+bt)e^{-bt} - (1+b(t+s))e^{b(t+s)}\right]}$ |
|---|---|---|
| 7 | 30.9% | 0.79 |
| 8 | 22.89% | 0.85 |
| 9 | 14.80 | 0.90 |
| 10 | 10.30% | 0.93 |
| 11 | 6.87% | 0.95 |

### *5.1.2.2 Analyse der Restfehlerdaten mit Regression*

Dieses Beispiel wird ebenfalls mit Regressionen berechnet. Es geht darum, eine Vorhersage zu treffen und zu prüfen, ob eine Vorhersage nicht nur mit SR-Modellen, sondern auch mit einfachen Regressionen möglich ist. In der folgenden Abbildung 5.1-7 sieht man die kumulierte Fehlerkurve bis zur ZE 6, ab der gestrichelten Linie sieht man die Schätzungen der verschiedenen Modelle und Regressionen im Vergleich zur kumulierten Fehlerkurve. Hier wird deutlich, dass die lineare und logarithmische Regression keine gute Vorhersagekraft besitzen, wohingegen die vorhergesagte Kurve des Yamada-Delayed-S-shaped-Modells mit der Kurve der realen Daten übereinstimmt. In Kapitel 5.3 wird das Ergebnis weitergehend interpretiert.

**Abbildung 5.1-7 Analyse der Daten mit Regressionen und Yamadas-Modell MHU-A**

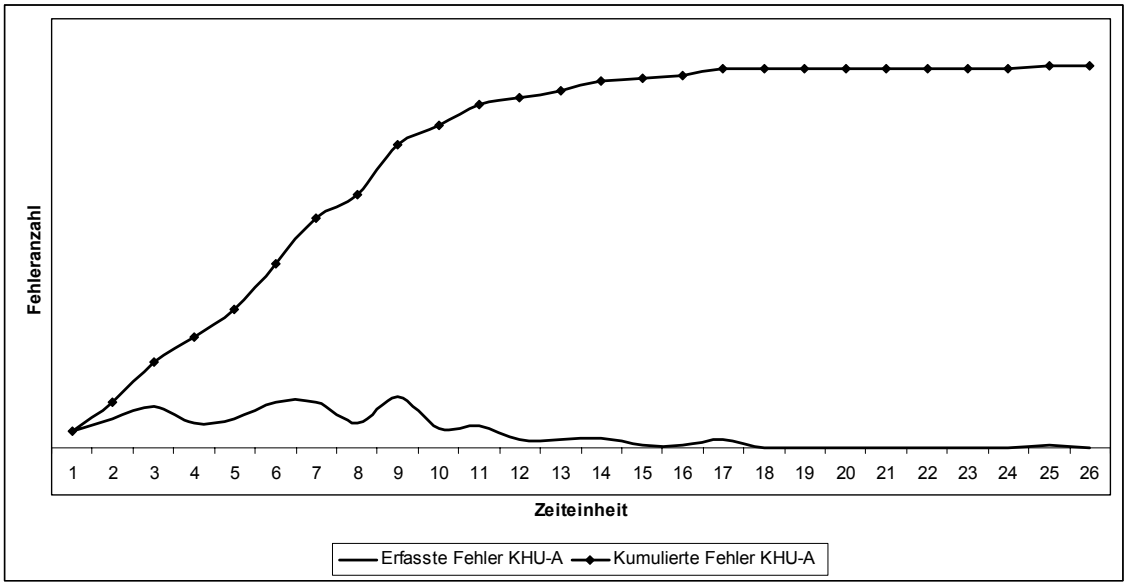


### 5.1.3 Analyse der Komponente „kleine Head Unit “

#### *5.1.3.1 Analyse der Fehlerdaten mit Software Reliability Modellen*

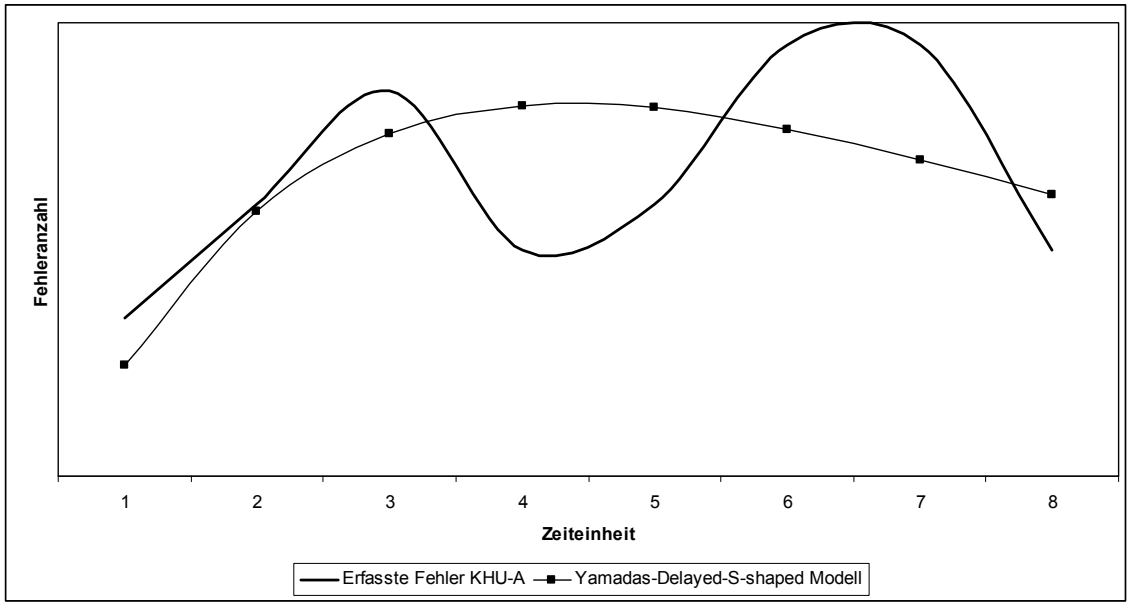
Um die Daten zu analysieren zeigt die Abbildung 5.1-8 die Kurve der kumulierten Fehler. Die Daten der erfassten Kurve wurden als Inputdatei in SMERFS^3 eingetragen (zum Aussehen der Inputdatei siehe Kap. 4.6). Hierzu wurden die ZE 1 bis 9 für die Kurve KHU-A ausgewählt. Es wurden mit Absicht nicht alle vorhandenen Daten beispielweise für die Kurve KHU-A bis ZE 26 eingetragen, um die erfassten Daten zwischen ZE 10 bis ZE 26 mit der Vorhersage (auf Basis der ZE 1 bis 8) für diesen Zeitraum vergleichen zu können. Durch diesen Vergleich der realen und vorhergesagten Daten zeigt sich die Genauigkeit der Vorhersage des Modells.

**Abbildung 5.1-8 Fehlerkurve und kumulierte Fehlerkurve KHU-A**




Die Input-Daten wurden dann in SMERFS^3 mit den Goel-Okumoto und Yamada-Delayed-S-shaped Modellen durchgeführt. Dabei ist das Goel-Okumoto Modell für diese Kurve nicht geeignet und wird daher im Folgenden nicht analysiert. In der folgenden Abbildung 5.1-9 sind die realen Daten als „erfasste Fehler KHU“ eingetragen. Die geschätzte Kurve mit dem Yamada-Delayed-S-shaped Modell ist ebenfalls in der Abbildung eingetragen.

**Abbildung 5.1-9 Berechnung des Yamadas-Modells mittels SMERFS^3 KHU-A**




In der folgenden Tabelle 5.1-5 sind einige Schätzungen und Prognosen für das Yamada-Delayed-S-shaped Modell für die kumulierten Fehler der Kurve KHU-A (Gesamte Schätzung und Prognose der beiden Modelle in Kap. 5.3) dargestellt. Die Tabelle 5.1-5 ist in vier Quadranten unterteilt. Der linke obere Quadrant zeigt die Wahrscheinlichkeit der Fehlerentdeckung (P). Dabei bedeutet 0.25 bei Yamada-Delayed-S-shaped Modell, dass zu 25% weitere Fehler entdeckt werden. Die zwei verschiedenen Konfidenzintervalle bedeuten, dass für die Vorhersagen zwei unterschiedlich lange Datenreihen benutzt wurden, d.h. eine kürzere und eine längere. Der rechte obere Quadrant zeigt die gesamte Anzahl von Fehlern in der Software an, wie sie vom Modell berechnet wird. Zur Anonymisierung der Daten wurde nicht die Zahl angegeben, sondern die Prozentzahl. 100% entsprechen der Gesamtzahl der real gefundenen Fehler bis zur ZE 26. Prozentzahlen in der Nähe von 100% zeigen an, dass das Modell sehr gut passt. In der Tabelle 5.1-5 sieht man, dass die durch das Yamada-Delayed-S-shaped Modell eingeschätzte Gesamtfehleranzahl mit 107.74% sehr nahe der realen Anzahl kommt. Der linke untere Quadrant zeigt die geschätzte Restfehleranzahl in Prozent der realen Restfehler bis zur ZE 26 an. Der rechte untere Quadrant zeigt Chi2 (Chi-Quadrat) (wie oben erklärt). Darüber hinaus kann man mit dem Modell für jede beliebige Zeitspanne vorhersagen lassen, wie viele Fehler voraussichtlich in dieser Zeitspanne gefunden werden. Im vorliegenden Beispiel wurde die Zeitspanne von 10 ZE gewählt. Hier zeigt sich, dass bis zur 10. ZE voraussichtlich 86% aller Restfehler gefunden werden.

**Tabelle 5.1-5 Einschätzung und Prognose für eine abgeschlossene IS KHU-A**

| | Niedrigeres Konfidenz-intervall | P | Höheres Konfidenz-intervall | Niedrigeres Konfidenz-intervall | TNF | Höheres Konfidenz-intervall |
|---|---|---|---|---|---|---|
| **YAM** | 0.16 | 0.25 | 0.35 | 67.82% | 107.74% | 147.65% |
| | **Niedrigeres Konfidenz-intervall** | **TNFR** | **Höheres Konfidenz-intervall** | **CHI2** | **Pred** | |
| **YAM** | 4.67% | 122.92% | 241.2% | 7.33 | 86.28% | |

P: Wahrscheinlichkeit der Fehlerendeckung
Pred: Im vorliegenden Fall: eingeschätzte Restfehleranzahl für die nächsten 10 Perioden in Prozent der geschätzten Gesamtanzahl der Restfehler
TNF (total number of faults): Gesamtumfang der geschätzten Fehleranzahl in Prozent (im Vergleich mit realen Daten)
TNFR (total number of faults remaining): Restfehleranzahl in Prozent (Vergleich mit realen Daten)
Chi2: Chi-Quadrat Wert

In der folgenden Tabelle 5.1-6 werden die Vorhersagen dargestellt, wie viele Fehler in Prozent der übriggebliebenen Restfehler in einer ZE voraussichtlich gefunden werden (z.B. werden in ZE 10 voraussichtlich 15.06% der Restfehler gefunden werden). Darüber hinaus wird die Reliability angezeigt (vgl. Kap. 3.2.3.3). Eine Reliability von 0.93 in der 12. ZE bedeutet: Die Wahrscheinlichkeit, dass die Software einen Zeitraum (0.01 einer Zeiteinheit) ohne Fehlerwirkung überlebt, liegt bei ca. 93%.

Diese Daten mit den Daten der obigen Tabelle (siehe Tab. 5.1-5) stellen wichtige Informationen dar, wann man mit dem Testen aufhören könnte. Beispielsweise könnte das vorher festgelegte Ziel sein, mit dem Testen aufzuhören, wenn 90% aller eingeschätzten Restfehler gefunden wurden (ebenso könnte man dies als Anteil der Gesamtfehler berechnen). Im vorliegenden Beispiel würde man also nach ZE 10 mit dem Testen aufhören.

**Tabelle 5.1-6 Vorhersage der Restfehler und der Reliability für IS KHU-A**

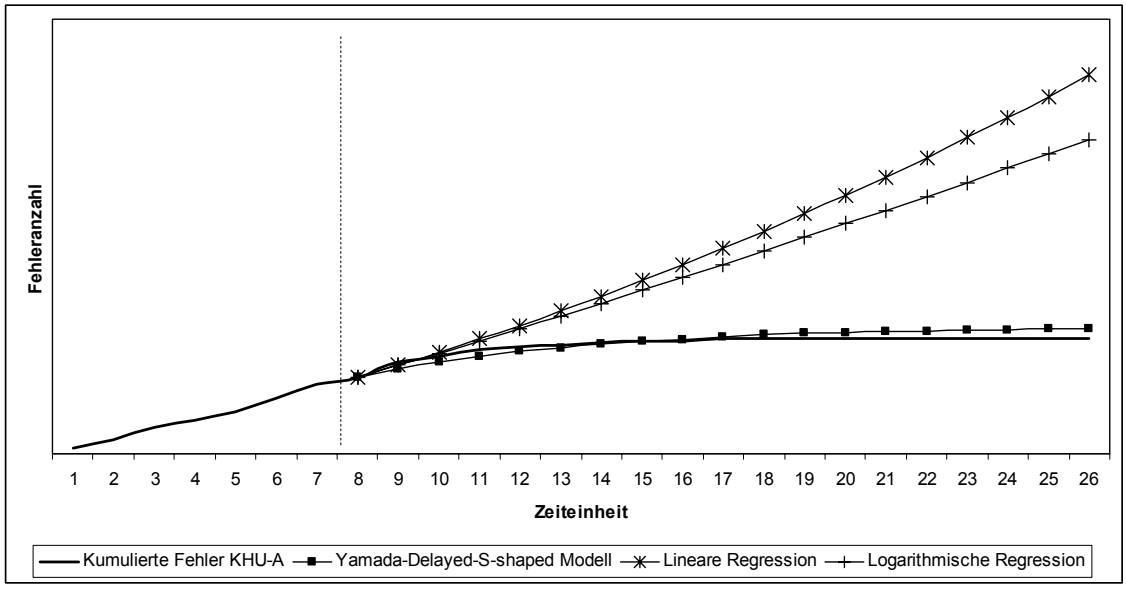
| **ZE** | **Anteil an Restfehlern Vorhersage** | **Reliabilty R(0.01) des Yamada-Delayed-S-shaped-Modells** $R(x/t) = e^{-a\left[(1+bt)e^{-bt} - (1+b(t+s))e^{b(t+s)}\right]}$ |
|---|---|---|
| 9 | 15.06% | 0.91 |
| 10 | 15.06% | 0.92 |
| 11 | 12.05% | 0.93 |

### *5.1.3.2 Analyse der Restfehlerdaten mit Regression*

Dieses Beispiel wird auch mit Regressionen berechnet. Es geht darum, eine Vorhersage zu treffen und zu prüfen, ob eine Vorhersage nicht nur mit SR-Modellen, sondern auch

mit einfachen Regressionen möglich ist. In der Abbildung 5.1-10 sieht man die kumulierte Fehlerkurve bis zur ZE 8, ab der gestrichelten Linie sieht man die Schätzungen der verschiedenen Modelle und Regressionen im Vergleich zur kumulierten Fehlerkurve. Hier wird deutlich, dass die lineare und logarithmische Regression keine gute Vorhersagekraft besitzen, wohingegen die vorhergesagte Kurve des Yamada-Delayed-S-shaped-Modells ähnlich mit der Kurve der realen Daten übereinstimmt. In Kapitel 5.3 wird das Ergebnis weitergehend interpretiert.

**Abbildung 5.1-10 Analyse der Daten mit Regressionen und Yamadas-Modell KHU-A**




## 5.1.4 Analyse der Komponente „Steuergerät für Telefonie“

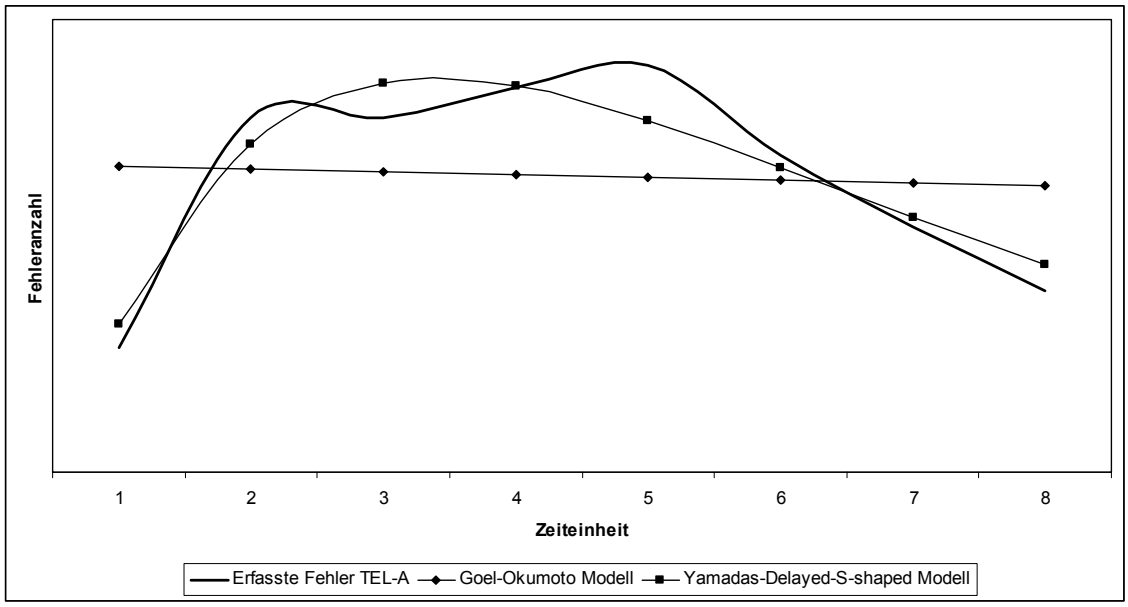
### *5.1.4.1 Analyse der Fehlerdaten mit Software Reliability Modellen*

Um die Daten zu analysieren zeigt die Abbildung 5.1-11 die Kurve der kumulierten Fehler. Die Daten der erfassten Kurve wurden als Inputdatei in SMERFS^3 eingetragen (zum Aussehen der Inputdatei siehe Kap. 4.6). Hierzu wurden die ZE 1 bis 8 für die Kurve TEL-A ausgewählt. Es wurden mit Absicht nicht alle vorhandenen Daten beispielweise für die Kurve bis ZE 26 eingetragen, um die erfassten Daten zwischen ZE 9 bis ZE 26 mit der Vorhersage (auf Basis der ZE 1 bis 8) für diesen Zeitraum vergleichen zu können. Durch diesen Vergleich der realen und vorhergesagten Daten zeigt sich die Genauigkeit der Vorhersage des Modells.

**Abbildung 5.1-11 Fehlerkurve und kumulierte Fehlerkurve TEL-A**


Die Input-Daten wurden dann in SMERFS^3 mit den Goel-Okumoto und Yamada-Delayed-S-shaped Modellen durchgeführt. In der folgenden Abbildung 5.1-12 sind die realen Daten als „erfasste Fehler TEL-A“ eingetragen. Die geschätzte Kurve mit dem Yamada-Delayed-S-shaped Modell und mit dem Goel-Okumoto-Modell ist ebenfalls in der Abbildung eingetragen.

**Abbildung 5.1-12 Berechnung der NHPP Modelle mittels SMERFS^3 TEL-A**



In der folgenden Tabelle 5.1-7 sind einige Schätzungen und Prognosen für die beiden Modelle für die kumulierten Fehler der Kurve TEL-A (Gesamte Schätzung und Prognose der beiden Modelle in Kap. 5.3) dargestellt. Das Goel-Okumoto Modell ist nicht geeignet, weil die Berechnung der Gesamtfehleranzahl und der Restfehler mit den realen Daten in keiner Weise übereinstimmt. Die Tabelle 5.1-7 ist in vier Quadranten unterteilt. Der linke obere Quadrant zeigt die Wahrscheinlichkeit der Fehlerentdeckung (P). Dabei bedeutet 0.34 bei Yamada-Delayed-S-shaped Modell, dass zu 34% weitere Fehler entdeckt werden. Die zwei verschiedenen Konfidenzintervalle bedeuten, dass für die Vorhersagen zwei unterschiedlich lange Datenreihen benutzt wurden, d.h. eine kürzere und eine längere. Der rechte obere Quadrant zeigt die gesamte Anzahl von Fehlern in der Software an, wie sie vom Modell berechnet wird. Zur Anonymisierung der Daten wurde nicht die Zahl angegeben, sondern die Prozentzahl. 100% entsprechen der Gesamtzahl der real gefundenen Fehler bis zur ZE 26. Prozentzahlen in der Nähe von 100% zeigen an, dass das Modell sehr gut passt. In der Tabelle 5.1-7 sieht man, dass die durch das Yamada-Delayed-S-shaped Modell eingeschätzte Gesamtfehleranzahl mit 115.3% sehr nahe der realen Anzahl kommt (bei Goel-Okumoto ist 1180%). Der linke untere Quadrant zeigt die geschätzte Restfehleranzahl in Prozent der realen Restfehler bis zur ZE 26 an. Der rechte untere Quadrant zeigt Chi2 (wie oben erklärt). Darüber hinaus kann man mit dem Modell für jede beliebige Zeitspanne vorhersagen lassen, wie viele Fehler voraussichtlich in dieser Zeitspanne gefunden werden. Im vorliegenden Beispiel wurde die Zeitspanne von 10 ZE gewählt. Hier zeigt sich, dass bis zur 10. ZE voraussichtlich 94.1% aller Restfehler gefunden werden.

**Tabelle 5.1-7 Einschätzung und Prognose für eine abgeschlossene IS TEL-A**

| | Niedrigeres Konfidenz-intervall | P | Höheres Konfidenz-intervall | Niedrigeres Konfidenz-intervall | TNF | Höheres Konfidenz-intervall |
|---|---|---|---|---|---|---|
| **Goel** | 0 | 0.009 | 0.04 | 88.2% | 1180% | 5153% |
| **YAM** | 0.30 | 0.34 | 0.39 | 103.2% | 115.3% | 127.4% |
| | **Niedrigeres Konfidenz-intervall** | **TNFR** | **Höheres Konfidenz-intervall** | **CHI2** | **Pred** | |
| **Goel** | 0 | 9252% | 42938% | 64.28 | 855% | |
| **YAM** | 127.1% | 229.5% | 332% | 5.88 | 94.1% | |

P: Wahrscheinlichkeit der Fehlerendeckung
Pred: Im vorliegenden Fall: eingeschätzte Restfehleranzahl für die nächsten 10 Zeiteinheit in Prozent der geschätzten Gesamtanzahl der Restfehler
TNF (total number of faults): Gesamtumfang der geschätzten Fehleranzahl in Prozent (im Vergleich mit realen Daten)
TNFR (total number of faults remaining): Restfehleranzahl in Prozent (Vergleich mit realen Daten)
Chi2: Chi-Quadrat Wert

In der folgenden Tabelle 5.1-8 werden die Vorhersagen dargestellt, wie viele Fehler in Prozent der übriggebliebenen Restfehler in einer ZE voraussichtlich gefunden werden (z.B. werden in ZE 9 voraussichtlich 26% der Restfehler gefunden werden). Darüber hinaus wird die Reliability angezeigt (vgl. Kap. 3.2.3.3). Eine Reliability von 0.61 in der 8. ZE bedeutet: Die Wahrscheinlichkeit, dass die Software einen Zeitraum (0.01 einer Zeiteinheit) ohne Fehlerwirkung überlebt, liegt bei ca. 61%. Diese Daten mit den Daten der obigen Tabelle (siehe Tab. 5.1-7) stellen wichtige Informationen dar, wann man mit dem Testen aufhören könnte. Beispielsweise könnte das vorher festgelegte Ziel sein, mit dem Testen aufzuhören, wenn 90% aller eingeschätzten Restfehler gefunden wurden (ebenso könnte man dies als Anteil der Gesamtfehler berechnen). Im vorliegenden Beispiel würde man also nach ZE 14 mit dem Testen aufhören.

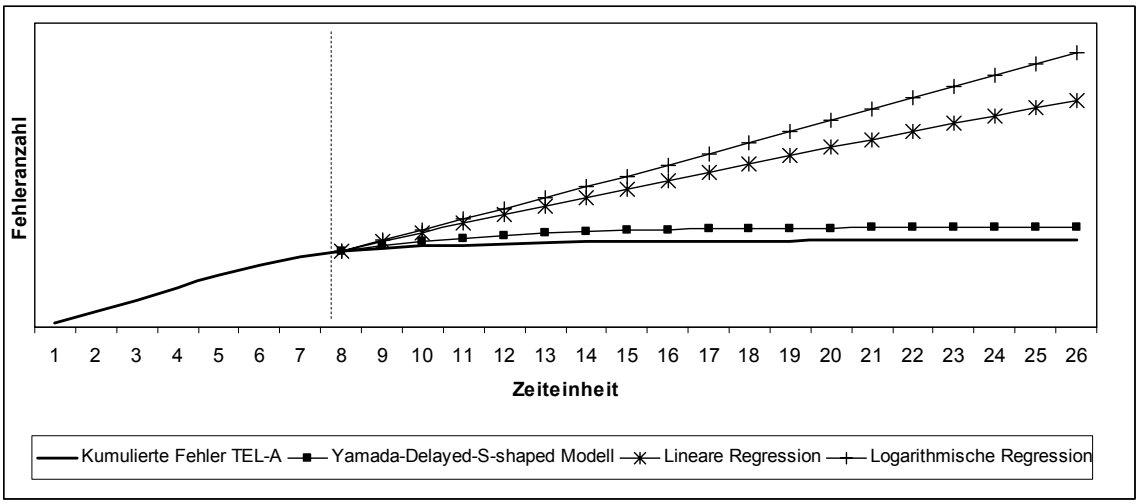
**Tabelle 5.1-8 Vorhersage der Restfehler und der Reliability für IS TEL-A**

| **ZE** | **Anteil an Restfehlern Vorhersage** | **Reliabilty R(0.01) des Yamada-Delayed-S-shaped-Modells** $R(x/t) = e^{-a\left[(1+bt)e^{-bt} - (1+b(t+s))e^{b(t+s)}\right]}$ |
|---|---|---|
| 9 | 22.82% | 0.68% |
| 10 | 17.63% | 0.74% |
| 11 | 14.52% | 0.79 |
| 12 | 10.89% | 0.83 |
| 13 | 8.29% | 0.87 |
| 14 | 6.22% | 0.90 |

#### *5.1.4.2 Analyse der Restfehlerdaten mit Regression*

Dieses Beispiel wird auch mit Regressionen berechnet. Es geht darum, eine Vorhersage zu treffen und zu prüfen, ob eine Vorhersage nicht nur mit SR-Modellen, sondern auch mit einfachen Regressionen möglich ist. In der Abbildung 5.1-13 sieht man die kumulierte Fehlerkurve bis zur ZE 8, ab der gestrichelten Linie sieht man die Schätzungen der verschiedenen Modelle und Regressionen im Vergleich zur kumulierten Fehlerkurve. Hier wird deutlich, dass die lineare und logarithmische Regression keine gute Vorhersagekraft besitzen, wohingegen die vorhergesagte Kurve des Yamada-Delayed-S-shaped-Modells mit der Kurve der realen Daten übereinstimmt. In Kapitel 5.3 wird das Ergebnis weitergehend interpretiert.

**Abbildung 5.1-13 Analyse der Daten mit Regressionen und Yamadas-Modell TEL-A**




## *5.2 Interpretation und Vergleich der Ergebnisse*

### 5.2.1 Interpretation und Vergleich der Ergebnisse in Bezug auf die Problemschwere

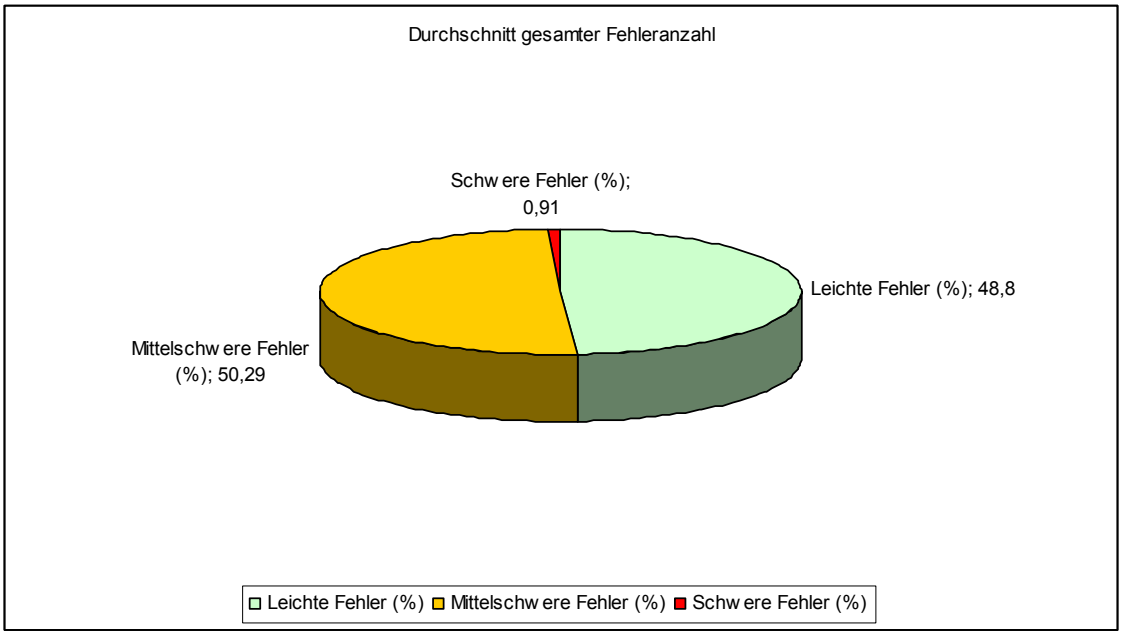
Als erstes wird hier der Anteil der verschiedenen Problemschweren an der Gesamtfehleranzahl aller vier untersuchten Komponenten im Vergleich dargestellt (siehe folgende Tab. 5.2-1), hierbei handelt es sich um eine fehlerbasierte Metrik (siehe Kap. 2.2.4.3).

**Tabelle 5.2-1 Anteil der Fehlerarten nach Problemschwere für die Komponenten**

| | Leichte Fehler (%) | Mittelschwere Fehler (%) | Schwere Fehler (%) |
|---|---|---|---|
| **GHU** | 34.8 | 62.6 | 2.6 |
| **MHU** | 64.3 | 35.3 | 0.4 |
| **KHU** | 60.7 | 39.1 | 0.2 |
| **TEL** | 35.39 | 63.77 | 0.84 |
| GHU: Große Head Unit<br>MHU: Mittlere Head Unit | | KHU: Kleine Head Unit<br>TEL: Steuergerät für Telefonie | |

Bei der Bewertung nach Fehlerschwere sieht man, dass die Fehlerschwere der Komponenten unterschiedlich variiert hat. Wenn man den gesamten Durchschnitt betrachtet, waren 48.80% leichte Fehler, 50.29% der Fehler mittelschwer und nur 0.91% waren schwere Fehler (vgl. folgende Abb. 5.2-1).

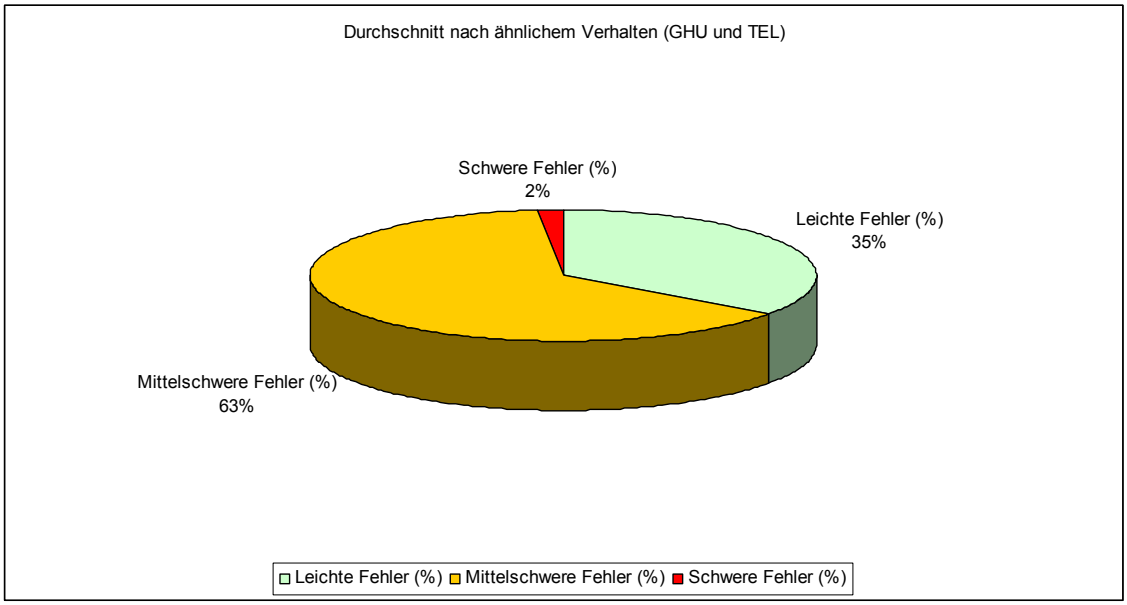
**Abbildung 5.2-1 Problemschwere im Durchschnitt**

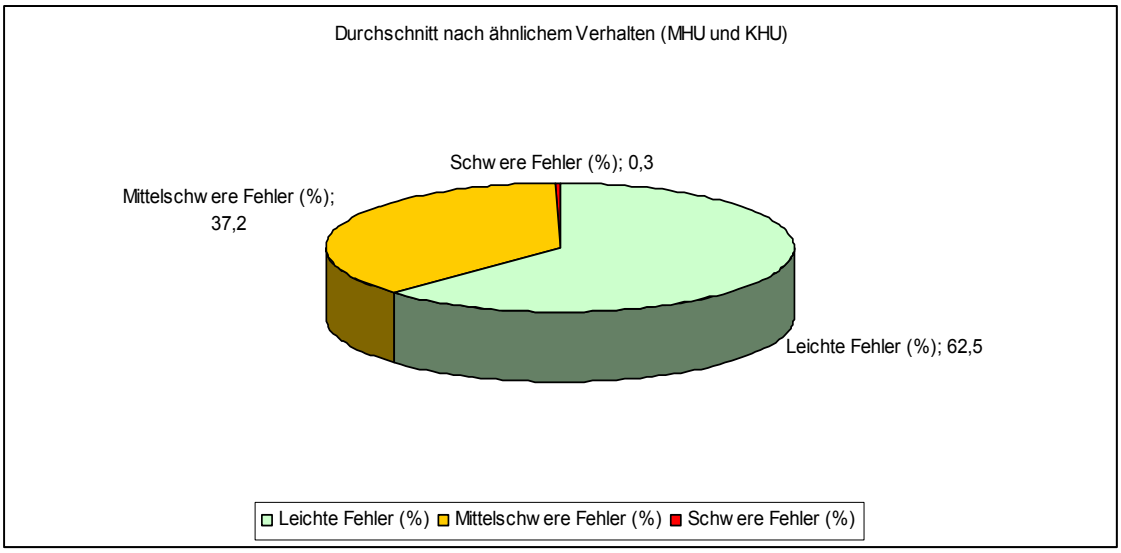



Wichtig ist aber zu betrachten, dass die Komponenten GHU und TEL ein ähnliches Verhalten zeigten und zwar waren bei diesen beiden Komponenten im Durchschnitt 35.1% der Fehler leicht, 63.19% waren mittelschwer, und 1.72% waren schwer. Dagegen zeigen die MHU und KHU das Verhältnis genau andersrum und zwar waren 62.5% leichte Fehler, 37% waren mittelschwer und 0.3% waren schwer (vgl. folgende Abb. 5.2-2). Dies lässt sich durch die unterschiedliche Komplexität der Komponenten erklären: die große Head Unit und das Telefoniesteuergerät sind sehr komplex da sie mehrere Teilsysteme beinhalten und zeigen daher vermutlich mehr schwere und

mittelschwere Fehler, wohingegen die mittlere und kleine Head Unit weniger komplex sind und mehr leichte Fehler zeigen.

**Abbildung 5.2-2 Problemschwere nach ähnlichen Komponenten**




Aufgrund der unterschiedlichen Anteile nach verschiedenen Problemschweren wurden in der vorliegenden Arbeit alle Problemschweren gemeinsam mit SRM berechnet, um die Vergleichbarkeit zu gewährleisten. Für die Praxis kann es jedoch – je nach Zielen – relevant sein, nur bestimmte Problemschweren mit SRM zu berechnen. Beispielsweise könnte man bei sehr komplexen Komponenten wie der großen Head Unit auch nur die mittelschweren und schweren Fehler analysieren und vorhersagen, da diese fast 2/3 aller Fehler ausmachen. Bei weniger komplexen Komponenten wie der mittleren und kleinen

Head Unit wäre dies jedoch nicht richtig, da hier die leichten Fehler – die in ihrer Masse jedoch auch nachteilig sind – 2/3 aller Fehler ausmachen. Neben den Prozentanteilen sind auch die absoluten Zahlen für diese Entscheidungen relevant, da es einen Unterschied macht, ob man Kurven für 2% von 100 oder von 10.000 Fehlern berechnet.

### 5.2.2 Interpretation und Vergleich der Ergebnisse in Bezug auf die Passung der Modelle

Die Modellauswahl beinhaltet wie in Kapitel 3.3 beschrieben zwei Schritte: der erste Schritt bestand in der generellen Auswahl von Modellen, die für die Fragestellung und die Daten passend waren. Im zweiten Schritt wird – nachdem die ausgewählten Modelle auf die Daten angewandt und berechnet wurden – das von diesen Modellen am besten passende ausgewählt. Der zweite Schritt betrachtet die deskriptive und voraussagende Stärke eines Modells in Bezug auf die speziellen Daten. In der vorliegenden Arbeit wird nach Wallace (2001) diese deskriptive und voraussagende Stärke eines Modells dadurch berechnet, dass die geschätzte Gesamtfehleranzahl und die vorhergesagte Restfehleranzahl mit den realen Anzahlen verglichen werden. Für Chi-quadrat gilt, je kleiner die Werte sind, desto besser ist das Modell geeignet, vergleichbar zu anderen Modellen mit dem gleichen Datenbestand (vgl. Pham 2006, S. 182).

Im Folgenden werden die Ergebnisse der Modellberechnungen in Bezug auf die Schätzung, Vorhersage und Testbeendigung miteinander verglichen.

#### *5.2.2.1 Vergleich der Ergebnisse der Beispiele in Bezug auf die Schätzung der Gesamtfehler*

In der folgenden Tabelle 5.2-2 sind die Ergebnisse für die abgeschlossenen Integrationsstufen (IS) zusammengefasst. Da nur die abgeschlossenen Integrationsstufen einen Vergleich mit den realen Daten erlauben, wurden nur diese ausgewählt, um die deskriptive Stärke der Modelle in Bezug auf die Schätzung der Gesamtfehler zu vergleichen. Je näher der Prozentwert von TNF 100%, desto näher liegt der geschätzte Wert an dem realen Wert (=100%). In der Tabelle 5.2-2 ist jeweils das besser passende Modell markiert. Insgesamt kann man sagen, dass meist mit dem Yamada-Delayed-S-shaped Modell die bessere Schätzung vorgenommen wird. Nur bei der Kurve MHU-B ist das Goel-Okumoto besser geeignet. Dies zeigt sich auch bei der Betrachtung der Kurve (vgl. Abb. C-1). Sie ist deutlich konkav und hat keine Einschwingphase. Die Schätzungen liegen sehr nah an den realen Daten (+/-16%). In

Bezug auf die Reliabilitätsfunktion unterscheiden sich die Komponenten. Die Komponente KHU-B würde beispielsweise zu 60% einen festgelegten Zeitraum fehlerfrei überstehen, wohingegen die Komponente MHU-B dies zu 96% tun würde.

**Tabelle 5.2-2 Vergleich der Ergebnisse der abgeschlossenen Integrationsstufen**

| Gerät | IS | Goel-Okumoto Modell | | | Yamada-Delayed-S-shaped Modell | | | Yamada-Delayed-S-shaped Reliabilty R(0.01) (%) |
|---|---|---|---|---|---|---|---|---|
| | | CHI2 | In Prozent | | CHI2 | In Prozent | | |
| | | | P | TNF | | P | TNF | |
| **MHU** | MHU-A | 10.5 | 4 | 313 | 4.6 | 50 | 92 | 72 |
| | MHU-B | 11.5 | 22 | 103 | 9.2 | 67 | 85 | 96 |
| **KHU** | KHU-A | n.g | n.g | n.g | 7.3 | 25 | 107.7 | 89 |
| | KHU-B | 99.1 | 2.8 | 415 | 7.3 | 33.4 | 116 | 60 |
| **TEL** | TEL-A | 64.2 | 0.9 | 1180 | 5.8 | 34 | 115.3 | 61 |
| | TEL-B | 47.9 | 3.3 | 382 | 17.9 | 37.7 | 110.1 | 77.6 |

MHU: Mittlere Head Unit
KHU: Kleine Head Unit
TEL: Steuergerät für Telefonie
P: Wahrscheinlichkeit der Fehlerendeckung
n.g: nicht geeignet

### *5.2.2.2 Vergleich der Ergebnisse in Bezug auf Restfehler*

Ein weiteres Kriterium für die deskriptive Stärke der Modelle ist die Schätzung der Restfehler. In den beiden folgenden Tabellen 5.2-3 und 5.2-4 erfolgt diese Schätzung getrennt für die abgeschlossenen und noch nicht abgeschlossenen Integrationsstufen. In der ersten Tabelle 5.2-3 werden die geschätzten Restfehler mit den realen Restfehlern verglichen. Dabei sieht man deutlich, dass wieder meistens Yamada-Delayed-S-shaped Modell am besten passt. Zum Teil wirkt es, als ob die Restfehlerabschätzung weit entfernt von 100% (den realen Daten) ist. Dies ist zwar bei den letzten drei Integrationsstufen so, jedoch muss man sich hier in Erinnerung rufen, dass zur Anonymisierung der Daten die realen Restfehler mit 100 gleichgesetzt wurden. Wenn diese z.B. 10 realen Restfehlern entsprechen, wären 229% nur ca. 23 geschätzte Fehler. Diese wäre trotzdem noch akzeptabel und besser als die Berechnung mit Regressionen. Bei der Kurve MHU-A ist die Berechnung von Yamada-Delayed-S-shaped Modell mit 70% am nächsten an 100% , aber auch die logarithmische Regression liegt mit 146% recht nah an 100%, auch der Determinationskoeffizient ist mit 0.71 recht hoch.

**Tabelle 5.2-3 Vergleich der Restfehler (abgeschlossene IS)**

| Gerät | IS (Kurve) | NHPP-Modelle (%) | | Regression | | | |
|---|---|---|---|---|---|---|---|
| | | Goel-Okumoto | Yamada-Delayed-S-shaped | Lineare (%) | $R^2$ | Logarithmisch (%) | $R^2$ |
| **MHU** | MHU-A | 926 | 70 | 208 | 0.65 | 146 | 0.71 |
| | MHU-B | 116 | 23 | 56.3 | 0.71 | 150.7 | 0.55 |
| **KHU** | KHU-A | n.g | 122.92 | n.b. | 0.17 | n.b. | 0.26 |
| | KHU-B | 4667 | 332 | 1709 | 0.04 | n.b. | 0.01 |
| **TEL** | TEL-A | 9252 | 229.5 | n.b. | 0.004 | n.b. | 0.044 |
| | TEL-B | 2539 | 187.1 | 1097 | 0.04 | n.b | 0.0001 |

MHU: Mittlere Head Unit
KHU: Kleine Head Unit
TEL: Steuergerät für Telefonie
$R^2$: Determinationskoeffizient/Bestimmtheitsmaß[14]
n.g: nicht geeignet
n.b.: nicht bestimmbar

Bei der folgenden Tabelle. 5.2-4 kann man keinen Vergleich zwischen geschätzten und realen Daten ziehen, da die Integrationsstufen noch nicht abgeschlossen sind. Dies ist hier nur zur Dokumentation aufgeführt, wie es auch in der Praxis aussehen würde. Aus den obigen Erfahrungen kann man annehmen, dass das Yamada-Delayed-S-shaped Modell am ehesten die richtigen Ergebnisse liefern würde.

[14] Determinationskoeffizient oder Bestimmtheitsmaß ist definiert als ein Maß der Statistik für den Anteil der erklärten Varianz eines Zusammenhangs. Beträgt beispielsweise R2 = 0.4, dann heißt das, dass 40% der Streuung der einen Variablen durch lineare Abhängigkeit von der anderen Variablen erklärt werden kann

**Tabelle 5.2-4 Vergleich der Restfehler (nicht abgeschlossene IS)**

| Gerät | IS (Kurve) | NHPP-Modelle (%) | | Regression | | | |
|---|---|---|---|---|---|---|---|
| | | Goel-Okumoto | Yamada-Delayed-S-shaped | Lineare (%) | $R^2$ | Logarithmisch (%) | $R^2$ |
| **GHU** | GHU* | 17 | 3 | 0.2 | 0.5 | 68.7 | 0.22 |
| **MHU** | MHU-C* | 6 | 1 | 0 | 0.53 | 9.1 | 0.38 |
| **KHU** | KHU-C* | 2 | 0.2 | 0 | 0.79 | 0 | 0.80 |
| **TEL** | TEL-C* | 4.7 | 0.7 | 0 | 0.77 | 3.62 | 0.57 |
| | TEL-D* | 3.6 | 0.5 | 0 | 0.75 | 0.63 | 0.64 |

GHU: Große Head Unit
MHU: Mittlere Head Unit
KHU: Kleine Head Unit
TEL: Steuergerät für Telefonie
$R^2$: Determinationskoeffizient/Bestimmtheitsmaß
n.g: nicht geeignet
n.b.: nicht bestimmbar

Hier wird die Wichtigkeit von Grafikdarstellungen für die Praxis deutlich. Als Beispiel verweise ich hier noch einmal auf die nicht abgeschlossene Integrationsstufe GHU (vgl. Kap. 5.1.1.1). Hier sieht man in der Abbildung 5.1-4 „Analyse der Daten mit Regressionen und NHPP-Modellen GHU“ klar, dass Yamada-Delayed-S-shaped Modell am wahrscheinlichsten der realen Kurve entsprechen wird.

### 5.2.3 Interpretation der Ergebnisse in Bezug auf Testbeendigung

Wichtig ist, dass die SR zur Unterstützung des Testmanagements gedacht ist. Deswegen ist es auch wichtig, den Zeitpunkt für die Beendigung des Testens nach vorher festgelegten Kriterien zu bestimmen. Als Beispiel in der Masterarbeit wird das Kriterium, dass die Software zu 90% für einen bestimmten Zeitraum fehlerfrei läuft, angenommen. In der folgenden Tabelle 5.2-5 kann man sehen, dass nur die Yamada-Delayed-S-shaped Bestimmung der Reliabilität sinnvoll ist (siehe auch vorherige Abbildungen in Kap. 5.1 beim Vergleich mit den realen Daten). Bei den linearen und logarithmischen Regressionen bekommen wir keine guten Ergebnisse.

* Die untersuchten IS sind nicht abgeschlossen, deswegen sind ihre TNFR nicht mit realen Daten vergleichbar (z.B. 17 in GHU NHPP bedeutet, dass 17% Restfehler noch zu finden sind.)

**Tabelle 5.2-5 Vergleich der Bestimmungen des Testendes**

| Gerät | IS | Testende in Zeiteinheit | | |
|---|---|---|---|---|
| | | Yamada-Delayed-S-shaped (90% der reliability) | Lineare | Logarithmische |
| **GHU** | GHU * | 3 | 1' | 44 |
| **MHU** | MHU-A | 3 | 13 | 14 |
| | MHU-B | 0[15] | 6 | 11 |
| | MHU-C * | 0 | 0 | 10 |
| **KHU** | KHU-A | 1 | n.b. | n.b. |
| | KHU-B | 7 | 30 | n.b. |
| | KHU-C * | 0 | 0 | 0 |
| **TEL** | TEL-A | 6 | n.b. | n.b. |
| | TEL-B | 4 | 22 | n.b. |
| | TEL-C * | 0 | 0 | 6 |
| | TEL-D * | 0 | 0 | 4 |

GHU: Große Head Unit
MHU: Mittlere Head Unit
KHU: Kleine Head Unit
TEL: Steuergerät für Telefonie
n.b.: nicht bestimmbar

Die Bestimmung der Reliability ist wichtig, um die Fehlerfolgenkosten und Qualitätssicherung in Balance zu halten (siehe Kap. 2.2.4.2.).

## *5.3 Zusammenfassung*

Im Folgenden werden zusammenfassend die Fragestellungen (vgl. Kap. 4.2.2) – die in 5.1 und 5.2 ausführlich dargestellt wurden – beantwortet.

*1. Welche SR-Modelle liefern für die jeweiligen Komponenten die besten Einschätzungen und Vorhersagen?* Hierzu wurden das Goel-Okumoto und Yamada-Delayed-S-shaped Modell miteinander verglichen. Die Ergebnisse der SRM sowie die

' Bei den Regressionen ist nur bestimmbar, wenn die geschätzte gesamte kumulierte Fehleranzahl einen hohen Punkt erreicht und dann sinkt sie (z.B. Fehleranzahl bis ZE 10 ist 100, bis ZE 12 ist 102 dann bis ZE 13 ist 102,5). Man geht davon aus, dass es bis ZE 12 sinnvoll wäre weiter zu testen.

[15] Eine Null bedeutet hier, dass man nicht weiter testen muss.

Grafiken der kumulierten Fehlerkurven sind kongruent: sowohl in den Grafiken erschienen die Kurven als S-shaped-Kurven als auch das Yamada-Delayed-S-shaped Modell (das auf einem S-shaped-Verhalten beruht) war im Vergleich meistens besser geeignet. Dies gilt auch im Vergleich der unterschiedlich komplexen Komponenten (vgl. Kap. 5.2.2.1).

*2. Liefern für die Komponenten auch Regressionen gute Ergebnisse?* Hierzu wurden Regressionen mit SR-Modellen vergleichen. Die Vorhersagen mit Regressionen waren nicht so präzise wie die Vorhersagen des Yamada-Delayed-S-shaped Modells (vgl. Tabellen zum Vergleich in Kap. 5.2.2.2).

*3. Wie gut ist die Software (wie fehlerfrei)?* Hierzu wurden die Restfehler pro Zeitintervall geschätzt und vorhergesagt und – sofern möglich (d.h. Integrationsstufe abgeschlossen) – mit den beobachteten realen Daten verglichen (vgl. Kap. 5.2.2.2).

*4. Wann kann man mit dem Testen aufhören?* Wichtig ist dazu der Einsatz der Reliabilityfunktion, um vorhersagen zu können, zu welcher Wahrscheinlichkeit die Software in einem bestimmten Zeitraum fehlerfrei laufen wird. Dies stellt also ein Testendekriterium dar (vgl. Kap. 5.2.3).

# 6 Zusammenfassung

In diesem abschließenden Kapitel soll zunächst Allgemeines zum Vorgehen bei der Anwendung von Software Reliability Modellen zusammengefasst werden. Danach wird ein Ausblick auf den Einsatz von Software Reliability Modellen zur Unterstützung des Testmanagements gegeben.

Zusammenfassend lässt sich sagen, dass es wichtig ist, vor Anwendung der Software Reliability Modelle eine ausführliche Voranalyse der Daten durchzuführen (vgl. Kap. 4). Voranalysen der Fehlerdaten in Bezug auf die Problemschwere ermöglichen es, bestimmte Fehlerarten für die Software Reliability Modell Analyse auszuwählen und durchzuführen (vgl. Kap. 5.2.1). Nur durch diese Voranalysen ist es dann möglich, die richtigen Daten – z.B. in Bezug auf die Problemschwere und Integrationsstufe – sowie die zu den Daten passenden Software Reliability Modelle auszuwählen. Hierzu muss man die Daten in Bezug auf Fehlerdatentypen (z.B. Intervalldaten), Art des Testprozesses und Art der Testobjekte untersuchen und diese mit den Anforderungen der Software Reliability Modelle vergleichen (vgl. Kap. 3.3.1). Dabei ist es hilfreich, wenn das Fehlermanagementsystem möglichst viele Informationen bereitstellt, wichtig wäre beispielsweise die Dokumentation der Anzahl der involvierten Tester sowie die genauen Testzeiten (um festzustellen, ob tatsächlich keine Fehler auftraten oder nur nicht getestet wurde): “Similarly two people conducting tests instead of one person changes the interval size. The lesson is that test schedule information and staffing levels should be required information in the tracking system and should be made available to the analyst.” (Wallace 2001, o.A.). Insgesamt lässt sich sagen, dass es gut ist, wenn möglichst viele Daten vorliegen, da, wenn mehr Zeitintervalle berücksichtigt werden, das Ergebnis besser wird.

Um sich für Software Reliability Modelle entscheiden zu können, muss man einen Einblick in Software Reliability und Software Reliability Modelle haben (vgl. Kap. 3). Für ein besseres Verständnis von dem Qualitätskriterium Zuverlässigkeit ist es notwendig, sich zuvor allgemein mit Softwarequalität und verschiedenen Maßnahmen der Qualitätssicherung von Software beschäftigt zu haben (vgl. Kap. 2).

Aus den Ergebnissen (vgl. Kap. 5) können einige Punkte für den Einsatz von Software Reliability Modellen zur Unterstützung des Testmanagements abgeleitet werden.

Obwohl die untersuchten Komponenten unterschiedlich komplex waren, war das Yamada-Delayed-S-shaped Modell am besten geeignet und nicht die Regressionen. Hier zeigte sich der Mehrwert des Einsatzes von spezifischen Software Reliability Modellen. Mit dem Yamada-Delayed-S-shaped Modell waren genaue Berechnungen, Vorhersagen und der Vergleich mit vorher festgelegten Zielen (z.B. 90% Zuverlässigkeit) möglich. Dieses Modell erfüllt die Eigenschaften eines guten Software Reliability Modells: Es gibt (1) eine gute Prognose des zukünftigen Fehlerwirkungsverhaltens in Form von Restfehlern in Abhängigkeit von der Zeit, berechnet (2) nützliche Quantitäten wie zum Beispiel Gesamtfehler-/Restfehleranzahl, ist (3) einfach, weil es mit einem Werkzeug unterstützt werden kann, (4) breit anwendbar, da es sich für verschieden komplexe Komponenten eignet und basiert (5) auf soliden Vermutungen (vgl. auch Musa 1999, S. 260). Quantitäten wie die Zuverlässigkeitsfunktion, die die Wahrscheinlichkeit einer Software in einem bestimmten Zeitraum fehlerfrei zu laufen beschreibt, können als Testendekriterien wichtige Informationen für den Testmanager bereit stellen. Dabei sind gerade auch Testmetriken in Form von Grafiken für die Praxis wichtig, da sie auf einen Blick verständlich sind, den Vergleich mit typischen Kurven erlauben und damit auch die Modellwahl weiter absichern (z.B. Einschwingphase ja/nein). Sie sollten jedoch durch die genauen Berechnungen ergänzt werden. Obwohl die Werkzeuge die Berechnung vereinfachen, müssen die Quantitäten jedoch verstanden und interpretiert werden: „Analysts can perform the technique within reasonable amount of time and without a deep understanding of the mathematics. The two major issues are data reduction and interpretation of results." (Wallace 2001). Zusammenfassend lässt sich sagen, dass für eine gute Anwendung von Software Reliability Engineering in der Praxis weiterhin zeitliche und finanzielle Ressourcen sowie qualifizierte Mitarbeiter notwendig sind. Die vorliegende Arbeit konnte einen profunden Überblick über die theoretischen Grundlagen geben sowie exemplarisch und vergleichend die Anwendung in der Praxis aufzeigen.

# Literaturverzeichnis

(Birolini 2007): Birolini, A.: Reliability Engineering – Theory and Practice, Springer, 5. Aufl., Springer, Berlin 2007.

(Blischke; Murthy 2000): Blischke, W.-R.; Murthy, P.: Reliability – modelling, prediction, and optimization, Wiley, New York 2000.

(Bortz 1993): Bortz, J.: Statistik für Sozialwissenschaftler, Springer, Berlin 1993.

(Elbel; Mäckel o.Jg.): Elbel, B.; Mäckel, O.: Softwarezuverlässigkeit effizient bewerten – ein Erfahrungsbericht aus der Praxis. http://www.item.uni-bremen.de/itg-fg5/workshop_15/paper/elbel_softwarezuverlaessigkeit.pdf, o.Jg., Abruf am 2008-05-04.

(Farr 1996): Farr, W.-H: Software Realibility Modeling Survey, In: Lyu, M.R.: Handbook of Software Reliability Engineering, IEEE Computer Society, Los Alamitos - California 1996, Kap. 3, 71-117.

(Farr 2002a): Farr, W.-H.: Software Reliability and Smerfs^3 – A Methodology and Tool for Software Reliability, Assesment, http://www.slingcode.com/smerfs/downloads/NASA_Training_Mar2002.zip, 2002-03-14, Abruf am 2007.11.12.

(Farr 2002b): Farr, W.-H.: Smerfs and Smerfs^3, http://www.slingcode.com/smerfs/downloads/ComparingSMERFSVwithSMERFS3.zip, Abruf am 2007.11.12.

(Grimm 2005): Grimm, K.: Software-Technologie im Automobil. In: Liggesmeyer, P.; Rombach, D. (Hrsg.): Software Engineering eingebetteter Systeme – Grundlage – Methodik – Anwendungen, München 2005, S. 407-430.

(Hoffmann 2008): Hoffmann, D. W.: Software-Qualität, Springer, Heidelberg 2008.

(Huang; Lyu; Kuo 2003): Huang, Ch.-Y.; Lyu, M. R; Kuo, S.-Y..: Poisson Process Models for Software Reliability Estimation. In: IEEE Transations on software engineering , Vol. 29. Nro. 3. March 2003, S. 261-269. http://www.cse.cuhk.edu.hk/~lyu/paper_pdf/01183936.pdf, Abruf am 2008-04-18.

(Huang et al. 2000): Huang, Ch.-Y.; Kuo, S.-Y.; Lyu, M.R.; Lo, J.-H.: Quantitative Software Reliability Modeling from Testing to Operation. In: 11th International Symposium on Software Reliability Engineering ISSRE (2000), S. 72-82.

(ITWissen 2007): Das große Online Lexikon für Informationstechnologie,. http://www.itwissen.info , Abruf am 11-11-2007

(Yin; Trivedi 1999): Yin, L.; Trivedi, K.S.: Confidence Interval Estimation of NHPP-Based Software Reliability Models. In: 10th International Symposium on Software Reliability Engineering ISSRE (1999), S. 6-11.

(Kapur; Garg 1999): Kapur, P.K.; Garg, R.B.: Contribution to Hardware and Software Reliability, World Schientific, Singapur 1999.

(Leitch 1995); Leitch, R.D.: Reliability Analysis for Engineers – An Introduction, Oxford University Press, New York 1995.

(Liggesmeyer 2002): Liggesmeyer, P.: Software-Qualität – Testen, Analysieren und Verifizieren von Software, Spektrum Akademischer Verlag, Heidelberg 2002.

(Liggesmeyer 2005a): Liggesmeyer, P.: Einleitung und Überblick. In: Liggesmeyer, P.; Rombach, D. (Hrsg.): Software Engineering eingebetteter Systeme – Grundlage – Methodik – Anwendungen, München 2005, S. 1-12.

(Liggesmeyer 2005b): Liggesmeyer, P.: Prüfung eingebetteter Software. In: Liggesmeyer, P.; Rombach, D. (Hrsg.): Software Engineering eingebetteter Systeme – Grundlage – Methodik – Anwendungen, München 2005, S. 205-225.

(Liggesmeyer; Mäckel 1999): Liggesmeyer, P.; Mäckel, O.: Statistische Messung und Prognose von Zuverlässigkeit. In: Saglietti, F.; Goerigk, W.(Hrsg.): Sicherheit und Zuverlässigkeit software-basierter Systeme – Beiträge zum GI-Workshop in Bad Honnef, München 1999, S. 210-221.

(Lyu 1996): Lyu, M.R.: Handbook of Software Reliability Engineering, IEEE Computer Society, Los Alamitos - California 1996.

(Mellis 2001): Mellis, W.: Testen von Software. In: Mertens, P. (Hrsg.): Lexikon der Wirtschaftsinformatik, Springer, Berlin 2001, S. 471-472.

(Lyu; Schönwälder 1998): Lyu, M.R.;Schönwälder, J.: Web-CASRE: a web-based tool for software reliability modeling. In: 9th International Symposium on Software Reliability Engineering ISSRE (1998), S. 151-160.

(M-SRAT o.Jg.): M-STAT: Metrics-based Software Reliability Assesment Tool. http://www.rel.hiroshima-u.ac.jp/tools/msrat/, o.Jg., Abruf am 2008-05-06.

(Musa 1999): Musa, J.D.: Software reliability engineering – more reliable software, faster development and testing, McGraw-Hill, New York 1999.

(NAVSEA 2002): Naval Sea Systems Command: SMERFS^3 , Supporting, Analyses und Software & Hardware, http://www.slingcode.com/smerfs/downloads/UsingSmerfs3.zip, Abruf am 2008-02-08.

(Pham 2006): Pham, H.: System Software Reliability, Springer, London 2006.

(Rätzmann 2004): Rätzmann, M.: Software-Testing & Internationalisierung, 2. Aufl., Galileo Computing, Bonn 2004.

(Rinke et al. 2006): Rinke, T.; Bauer, T.; Metzger, A.; Robinson-Mallett, C.; Eschbach, R.; Pohl, K.: Einsatz von Modellen für das risikominimierende, anforderungsbasierte Testen von Softwaresystemen (ranTEST). http://www.softwarefoerderung.de/projekte/rantest/beitrag_ranTEST.pdf, 2006-06-26, Abruf am 2008-06-19.

(Salzmann; Stauner 2005): Salzmann, Ch.; Stauner, T.; Automative Software – Methoden & Technologien. http://www4.informatik.tu-muenchen.de/lehre/vorlesungen/ase/ss05/,2005-06-14, Abruf am 2008-06-11. (Zugriff nur für berechtigte Personen)

(Schneider 2007): Schneider, K.: Abenteuer Software Qualität – Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement, d.punkt.verlag, Heidelberg 2007.

(Spillner et al. 2006): Spillner, A.; Roßner, T.; Winter, M.; Linz, T.: Praxiswissen Softwaretest - Testmanagement, dpunkt.verlag, Heidelberg 2006.

(Spillner; Linz 2005): Spillner, A.; Linz, T.: Basiswissen Softwaretest, 3. Aufl., dpunkt.verlag, Heidelberg 2005.

(Stahlknecht; Hasenkamp 2005): Stahlknecht, P.; Hasenkamp, U.: Einführung in die Wirtschaftsinformatik, 11. Aufl., Springer, Heidelberg 2005.

(Thaller 1990): Thaller, G.E.: Software-Qualität – Entwicklung Test Sicherung, Sybex, Düsseldorf 1990.

(Tosteson; Demidenko 2002): Tosteson, T.; Demidenko, E.: Maximum Likelihood. In: Ruggeri, F.; Kenett, R.; Faltin, F. (Hrsg.): Encyclopedia of Statistics in Quality and Reliability, Volumen 3, Chichester 2007, S. 1057-1059.

(Wallace 2001): Wallace, D.R.: Practical Software Reliability Modeling, http://www.slingcode.com/smerfs/downloads/PraticaleModelingUsingSMERFS3.zip, 2003-01-11, Abruf am 2008-05-08.

(Wallace; Laing 2002): Wallace, D.R.; Laing, V.: Software Reliability Modeling: Traditional and Non-Parametric, http://www.nasa.gov/centers/ivv/ppt/172514main_Dolores_and_Victor_Software_Reliability_Modeling_Traditional_and_Non-Parametric.ppt, 1997-08-24, Abruf am 2008-05-08.

(Xie 1991): Xie, M.: Software Reliability Modelling, Wold Scientific, Singapore 1991.

(Zhang; Pham 2001): Zhang, X.; Pham, H.: A software reliability model with testing coverage and imperfect debugging. In: Pham, H. (Hrsg.): Recent Advances in Reliability and Quality Engineering, World Scientific, Singapore 2001, S. 17-31.

# Anhang

## *A Software Reliability Engineering Werkzeuge*

**Tabelle A-1 Übersicht über weitere SRE-Software**

| | CASRE[16] | RAT[17] |
|---|---|---|
| **Name** | **Computer Aided Software Reliability Estimation** | **Reliability Assessment Tool** |
| **Beschreibung** | **CASRE ist ein Software Werkzeug (tool) für die Messung und die Evaluation der Software Zuverlässigkeit.** | **Das Siemens eigene Werkzeug RAT unterstützt verschiedene Software Zuverlässigkeitsmodelle.** |
| **Unterstützte Modelle** | **Das Werkzeug unterstützt u.a. folgende Modelle:**<br>**- Musa Basic**<br>**- Littlewood/Verrall**<br>**- Jelinski-Moranda**<br>**- NHPP** | **Das Werkzeug unterstützt u.a. folgende Modelle:**<br>**- Goel-Okumoto**<br>**- Musa-Okumoto**<br>**- Yamada S-shape**<br>**- Geometrisches Moranda** |
| **Vorteile** | **CASRE ermöglich einen Text Editor zu benutzen und Input-Dateien (ASCII) zu kreieren. Die Parameter können mit maximum likelihood oder Methode der kleinsten Quadrate eingeschätzt werden.** | **- Verlässlichkeit kann leicht geprüft werden.**<br>**- Die Datenerfassung erfordert keinen Zusatzaufwand.**<br>**- Die Modellierung kann unmittelbar begonnen werden.** |
| **Nachteile** | **Das kann man nur mit dem Betriebssystem Windows benutzen.** | **Es ist nicht möglich, die Datenerfassung zu beeinflussen.**<br>**Ausfalldaten basieren auf Kalenderzeit.**<br>**Die Daten sind nicht immer geeignet strukturiert.** |
| **Registration** | **erforderlich** | **Nur bei Siemens** |
| **Download** | **Das ist eine freie Software, wenn Buch von Lyu (1996) gekauft wird. Die Dokumentation zum Werkzeug ist kostenpflichtig.** | **Nicht verfügbar** |
| **Link** | http://www.openchannelsoftware.com/projects/CASRE_3.0 | keine |

[16] Lyu und Schönwälder (1998, S. 151ff.) beschreiben detailliert technisch das CASRE-Werkzeug.

[17] Liggesmeyer und Mäckel (1999, S. 210ff.) beschreiben detailliert das RAT-Wekzeug.

## *B Lesehilfe zum besseren Verständnis der anonymisierten Daten*

Das folgende Beispiel erläutert, wie die absoluten Zahlen in Prozentzahlen dargestellt werden, um die Anonymisierung zu gewährleisten. Die realen Zahlen eines fiktiven Beispiels seien mit 100 für die Gesamtfehleranzahl und 10 für die Restfehler gegeben. In der Tabelle B-1 sieht man die berechneten absoluten Zahlen. Das heißt, Goel-Okumoto berechnet als Gesamtfehleranzahl 200, Yamadas-Modell berechnet 98. Als Restfehler berechnet Goel-Okumoto 80, Yamadas-Modell 5. In der Tabelle B-2 sieht man dann, wie diese Zahlen in Prozentzahlen wiedergegeben werden. Die berechneten absoluten Zahlen wurden in Relation zu den realen Zahlen gesetzt. So ergibt sich für das Goel-Okumoto-Modell ein Wert von 200% für die Gesamtfehleranzahl, da die absolute reale Zahl 100 beträgt. Die in der Arbeit angegebenen Prozentzahlen zeigen also die Relation zwischen realen Daten und theoretisch berechneten Daten, d.h. je näher die Prozentwerte bei 100 sind, desto besser ist die Prognose. Im vorliegenden Beispiel wären die Prognosen von Yamadas-Modell näher an 100%.

**Tabelle B-1 Einschätzung und Prognose für eine abgeschlossene IS (absolute Zahl)**

| | **Niedrigeres Konfidenz-intervall** | **P** | **Höheres Konfidenz-intervall** | **Niedrigeres Konfidenz-intervall** | **TNF** | **Höheres Konfidenz-intervall** |
|---|---|---|---|---|---|---|
| **Goel** | 0 | 0.04 | 0.10 | | 200 | |
| **YAM** | 0.43 | 0.50 | 0.57 | | 98 | |
| | **Niedrigeres Konfidenz-intervall** | **TNFR** | **Höheres Konfidenz-intervall** | **CHI2** | **Pred** | |
| **Goel** | | 80 | | | 3 | |
| **YAM** | | 5 | | | 9 | |

P: Wahrscheinlichkeit der Fehlerendeckung

Pred: Im vorliegenden Fall: eingeschätzte Restfehleranzahl für die nächsten 10 Zeiteinheiten in Prozent der geschätzten Gesamtanzahl der Restfehler

TNF (total number of faults): Gesamtumfang der geschätzten Fehleranzahl in Prozent (im Vergleich mit realen Daten)

TNFR (total number of faults remaining): Restfehleranzahl in Prozent (Vergleich mit realen Daten)

Chi2: Chi-Quadrat Wert

Fiktives Beispiel reale Daten: TNF: 100; TNFR: 10

**Tabelle B-2 Einschätzung und Prognose für eine abgeschlossene IS (Prozent)**

| | Niedrigeres Konfidenz-intervall | P | Höheres Konfidenz-intervall | Niedrigeres Konfidenz-intervall | TNF | Höheres Konfidenz-intervall |
|---|---|---|---|---|---|---|
| **Goel** | 0 | 0.04 | 0.10 | | 200% | |
| **YAM** | 0.43 | 0.50 | 0.57 | | 98% | |
| | **Niedrigeres Konfidenz-intervall** | **TNFR** | **Höheres Konfidenz-intervall** | **CHI2** | **Pred** | |
| **Goel** | | 800% | | | 30% | |
| **YAM** | | 50% | | | 90% | |

# *C Abbildungen zur Datenanalyse (nicht ausführlich besprochene Integrationsstufen)*

**Abbildung C-1 Analyse der Daten mit Regressionen und NHPP-Modellen MHU-B**

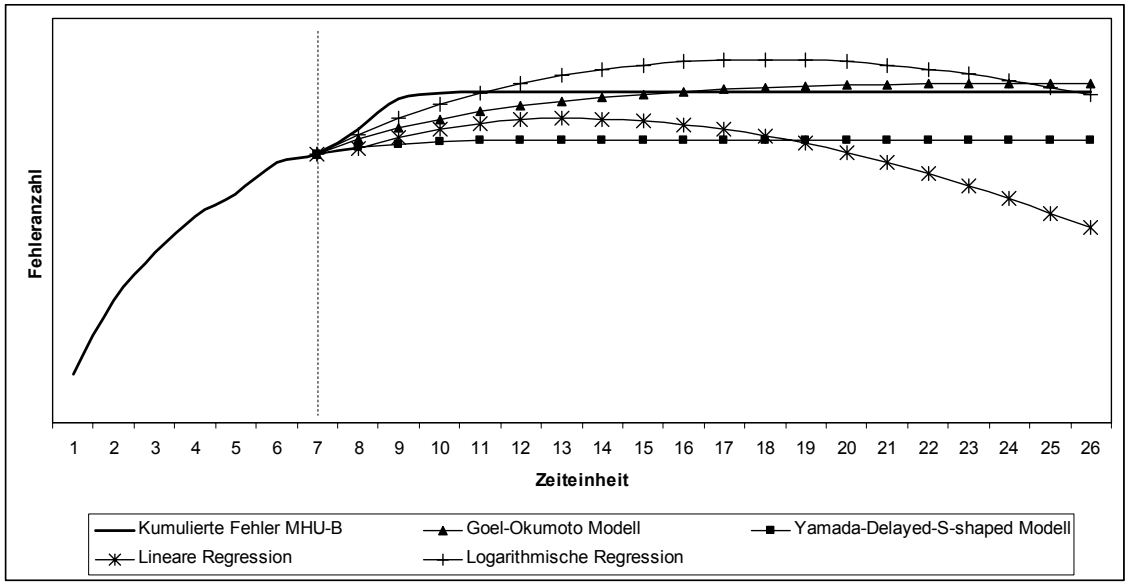



**Abbildung C-2 Analyse der Daten mit Regressionen und Yamadas-Modell MHU-C**

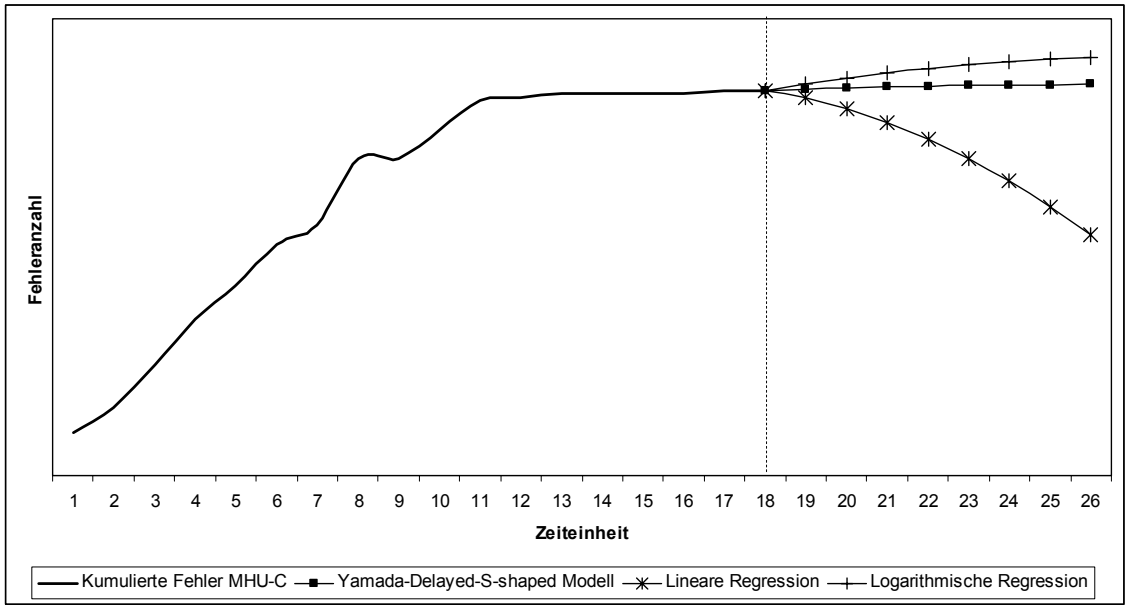


**Abbildung C-3 Analyse der Daten mit Regressionen und Yamadas-Modell KHU-B**

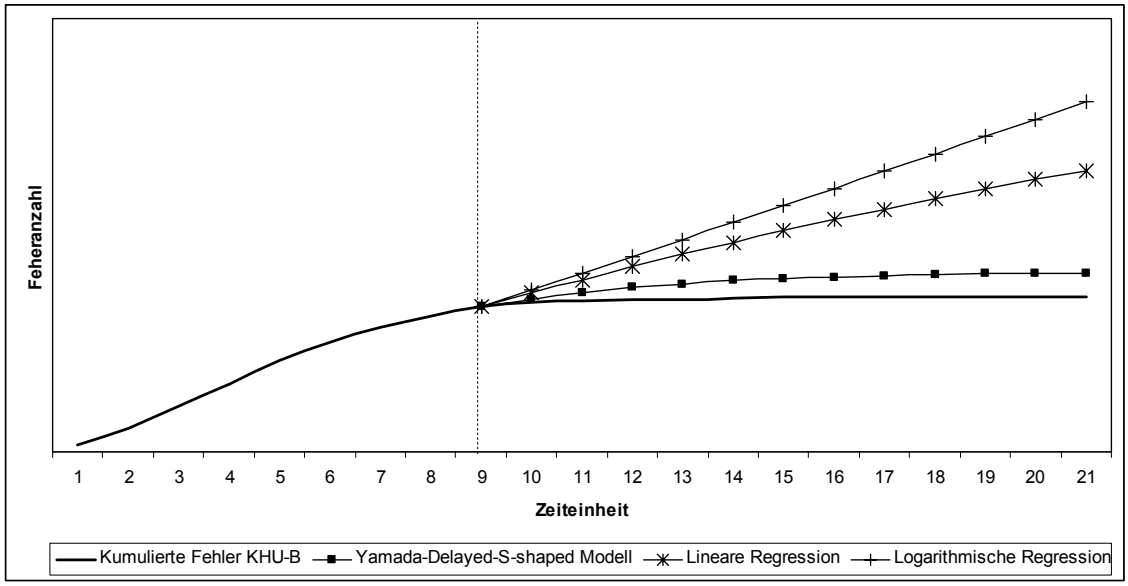



**Abbildung C-4 Analyse der Daten mit Regressionen und NHPP-Modellen KHU-C**

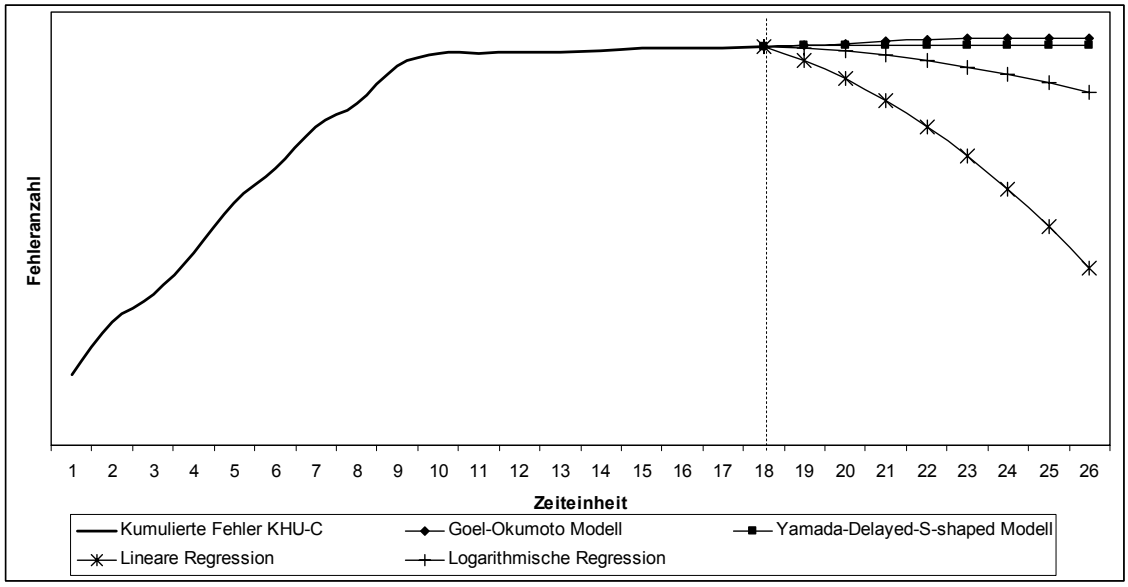


**Abbildung C-5 Analyse der Daten mit Regressionen und Yamadas-Modell TEL-B**

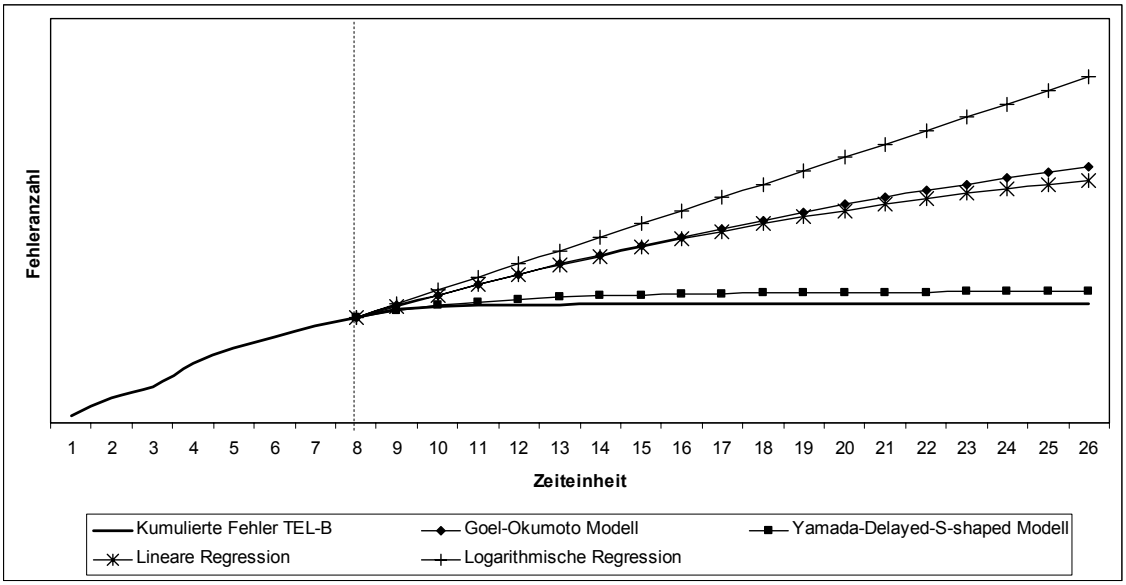



**Abbildung C-6 Analyse der Daten mit Regressionen und Yamadas-Modell TEL-C**

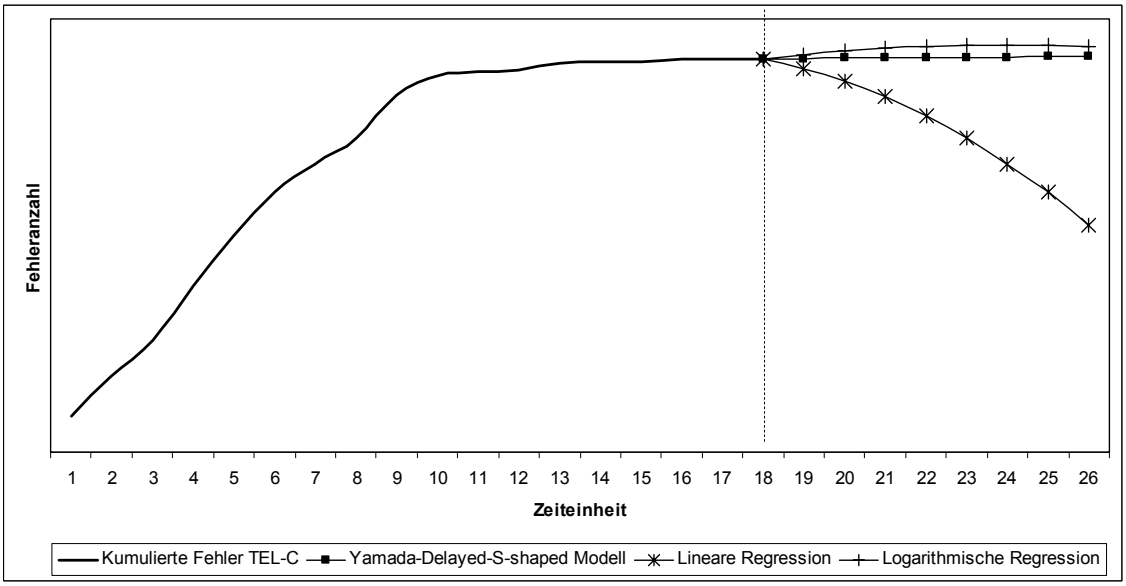
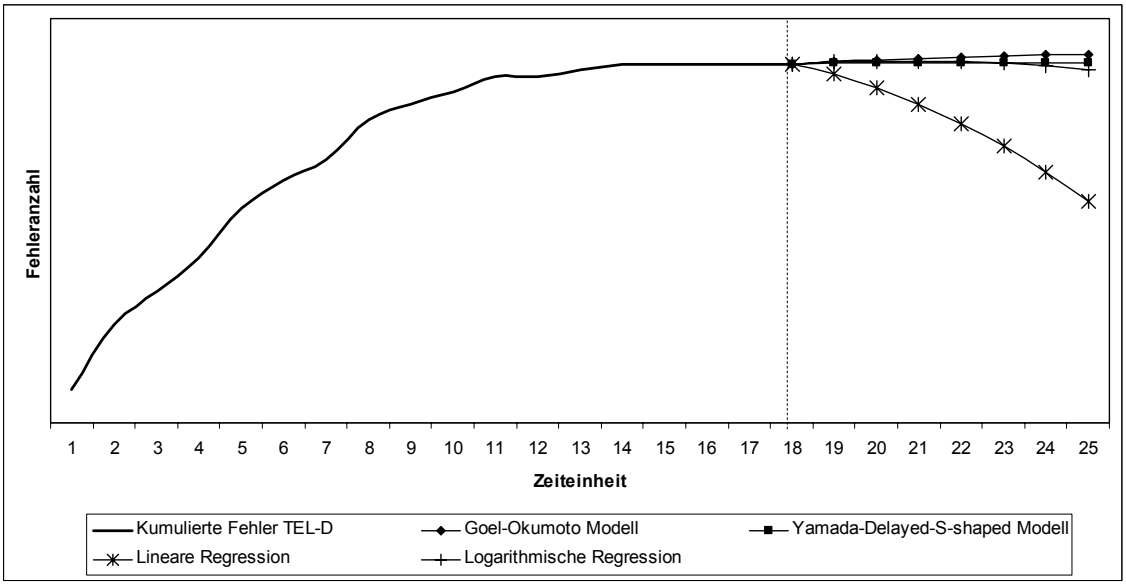


**Abbildung C-7 Analyse der Daten mit Regressionen und Yamadas-Modell TEL-D**



# Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben kann.

München, 28.07.2008