

Statistical procedures for certification of software systems

Citation for published version (APA):

Corro Ramos, I. (2009). *Statistical procedures for certification of software systems*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven.
<https://doi.org/10.6100/IR654166>

DOI:

[10.6100/IR654166](https://doi.org/10.6100/IR654166)

Document status and date:

Published: 01/01/2009

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Statistical Procedures for Certification of Software Systems

THOMAS STIELTJES INSTITUTE
FOR MATHEMATICS



© Corro Ramos, Isaac (2009)

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-2098-5

NUR: 916

Subject headings: Bayesian statistics, reliability growth models, sequential testing,
software release, software reliability, software testing, stopping time, transition sys-
tems

Mathematics Subject Classification: 62L10, 62L15, 68M15

Printed by Printservice TU/e
Cover design by Paul Verspaget

This research was supported by the Netherlands Organisation for Scientific Research
(NWO) under project number 617.023.047.



Nederlandse Organisatie voor Wetenschappelijk Onderzoek

Statistical Procedures for Certification of Software Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 15 december 2009 om 16.00 uur

door

Isaac Corro Ramos

geboren te Sevilla, Spanje

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. K.M. van Hee

en

prof.dr. R.W. van der Hofstad

Copromotor:

dr. A. Di Bucchianico

CONTENTS

1	Introduction	1
1.1	Motivation	1
1.1.1	The importance of software testing	1
1.1.2	Software failure vs. fault	2
1.1.3	Black-box vs. model-based testing	3
1.1.4	When to stop testing	3
1.2	Goal and outline of the thesis	4
2	Probability Models in Software Reliability and Testing	9
2.1	Preliminaries	10
2.2	Stochastic processes	12
2.2.1	Counting processes	13
2.2.2	Basic properties	14
2.2.3	Property implications	17
2.3	Software testing framework	18
2.3.1	Common notation	18
2.3.2	Reliability growth models	19
2.3.3	Stochastic ordering and reliability growth	21
2.4	Classification of software reliability growth models	23
2.4.1	Previous work on model classification	23
2.4.2	Classification based on properties of stochastic processes	26
2.5	General order statistics models	27
2.5.1	Jelinski-Moranda model	30
2.5.2	Geometric order statistics model	32
2.6	Non-homogenous Poisson process models	33
2.6.1	Goel-Okumoto model	35
2.6.2	Yamada S-shaped model	36
2.6.3	Duane (power-law) model	37
2.7	Linking GOS and NHPP models	38
2.7.1	A note on NHPP-infinite models	40
2.8	Bayesian approach	41
2.9	Some other models	42
2.9.1	Schick-Wolverton model	42
3	Statistical Inference for Software Reliability Growth Models	45
3.1	Data description	45
3.2	Trend analysis	46
3.3	Model type selection	52
3.4	Model estimation	54
3.4.1	ML estimation for GOS models	56
	Jelinski-Moranda model	57

3.4.2	ML estimation for NHPP models	58
	Goel-Okumoto model	58
	Duane (power-law) model	59
3.5	Model validation	60
3.6	Model interpretation	63
4	A New Statistical Software Reliability Tool	65
4.1	General remarks about the implementation	65
4.2	Main functionalities	66
4.2.1	Data menu	69
4.2.2	Graphics menu	70
4.2.3	Analysis menu	71
4.2.4	Help menu	78
4.3	Two examples of applying reliability growth models in software development	78
4.3.1	Administrative software at an insurance company	78
4.3.2	A closable dam operating system	83
5	Statistical Approach to Software Reliability Certification	89
5.1	Previous work on software reliability certification	90
5.1.1	Certification procedure based on expected time to next failure	90
5.1.2	Certification procedure based on fault-free system	92
5.2	Bayesian approach	93
5.3	Bayesian release procedure for software reliability growth models with independent times between failures	97
5.3.1	Jelinski-Moranda and Goel-Okumoto models	99
	Case 1: N and λ deterministic	99
	Case 2: N known and fixed, λ Gamma distributed	100
	Case 3: N Poisson distributed, λ known and fixed (Goel-Okumoto model)	102
	Case 4: N Poisson and λ Gamma distributed (full Bayesian approach)	103
5.3.2	Run model	106
	Case 1: N and θ deterministic	107
	Case 2: N Poisson distributed, θ known and fixed	107
	Case 3: N known and fixed, θ Beta distributed	109
	Case 4: N Poisson and θ Beta distributed (full Bayesian approach)	110
6	Performance of the Certification Procedure	111
6.1	Jelinski-Moranda model	111
6.1.1	Case 1: N and λ deterministic	111
6.1.2	Case 2: N known and fixed, λ Gamma distributed	112
6.1.3	Case 3: N Poisson distributed, λ known and fixed (Goel-Okumoto model)	117

6.1.4	Case 4: N Poisson and λ Gamma distributed (full Bayesian approach)	121
6.2	Run model	123
6.2.1	Case 1: N and θ deterministic	124
6.2.2	Case 2: N Poisson distributed, θ known and fixed	124
6.2.3	Case 3: N known and fixed, θ Beta distributed	126
6.2.4	Case 4: N Poisson and θ Beta distributed (full Bayesian approach)	127
7	Model-Based Testing Framework	131
7.1	Labelled transition systems and a diagram technique for representation	132
7.2	Example of modelling software as a labelled transition system	134
7.3	Error distribution	135
7.3.1	Binomial distribution of error-marked transitions	137
7.3.2	Poisson distribution of error-marked transitions	139
7.4	Testing process	140
7.5	Walking Strategies	146
7.5.1	Walking function update for labelled transition systems	146
7.5.2	Walking function update for acyclic workflow transition systems	148
7.6	Common notation	152
8	Statistical Certification Procedures	155
8.1	Certification procedure based on the number of remaining error-marked transitions	155
8.2	Certification procedure based on the survival probability	157
8.3	Practical application	161
8.3.1	General setup	161
8.3.2	Performance of the stopping rules	162
9	Testing the Test Procedure	167
9.1	Generating random models	167
9.2	Quality of the procedure	170
9.3	Stresser: a tool for model-based testing certification	173
9.3.1	Creating labelled transition systems	173
9.3.2	Error distribution	173
9.3.3	Parameters of testing: walking strategy and stopping rule	175
9.3.4	Collecting results	175
9.3.5	Further remarks	176
	Summary	177
	Bibliography	179
	Index	191
	About the author	195

CHAPTER 1

INTRODUCTION

In this chapter we first give a brief overview of software testing theory. We emphasize on the different approaches to software testing found in the literature and in common problems being studied during the past four decades. Afterwards, we introduce the main goals of our research and the outline of this thesis.

1.1 Motivation

The main goal of this section is to provide a clear motivation to our work. We first discuss the importance of software testing in Section 1.1.1. A common problem in software testing theory is that the terminology used is often confusing. For that reason, we introduce in Section 1.1.2 consistent terminology that will be used in this thesis. In Section 1.1.3 we present the two main approaches to software testing (called *black-box* and *model-based* testing). Finally, in Section 1.1.4 we consider the decision problem when to stop testing and the role of statistical models in order to answer this question.

1.1.1 The importance of software testing

Our first goal is to answer the question *why software testing is important?* If a software user is asked about this, the answer would likely to be *because software often fails*. The study of software systems during the past decades has revealed that practically all software systems contain faults even after they have passed an acceptance test and are in *operational* use. Software faults are of a special nature since they are due to human design or implementation mistakes. Since humans are fallible (so are software developers), software systems will have faults. Software systems are becoming so complex that, even if the number of possible test cases is theoretically finite, which is not always the case (for example, if unbounded input strings are allowed, then the number of test cases is infinite), their execution takes unacceptable much time in practice. Hence, it is impossible from a practical, or even theoretical, point of view to test them exhaustively. Therefore, there it is most likely that complex software systems have faults. We can improve upon this situation by designing rigorous *test procedures*. A *test* can be defined as the act of executing software with test cases with the purpose of finding faults or showing correct software execution (cf. Jorgensen (2002)[Chapter 1]). A test case is associated with the software behaviour since after its execution testers are able to determine whether a software system has met the corresponding specifications or not. Testing the software against specific acceptance criteria or requirements is a way to determine whether the software meets the quality demands. In that sense, testing can be regarded as a procedure to *measure* the quality of the software. Testing also helps

to detect (and repair) faults in the system. As long as faults are found and repaired, the number of remaining faults should decrease (although during the repair phase new faults may be introduced), resulting in a more reliable system. Here testing can be regarded as a procedure to *improve* software quality. Sound test designs should include list of inputs and expected outputs and documentation of the performed tests. Tests must be checked in order to avoid test cases to be executed without prior analysis of the requirements or mistake test faults for real software faults. There is a vast literature on software testing starting in the 1970's, [Myers \(1979\)](#) being one of the first monographs on the field. For more recent ones we refer to [Beizer \(1990\)](#), [Jorgensen \(2002\)](#) or [Patton \(2005\)](#).

1.1.2 Software failure vs. fault

The definition of a software fault is a delicate matter since vague or confusing definitions are often found in the software testing literature. In this thesis, we adopt the following terminology: when a deviation of software behaviour from user requirements is observed we say that a *failure* has occurred. On the other hand, a *fault* (*error*, *bug*, etc.) in the software is defined as an erroneous piece of code that causes failure occurrence. For us, a software fault occurs when at least one of the following rules (cf. [Patton \(2005\)](#)[Chapter 1]) is true:

1. The software does not do something that its specifications says it should do.
2. The software does something that its specifications says it should not do.
3. The software is difficult to understand, hard to use, slow or (in the software tester's eyes) will be viewed by the end user as just plain "not right".

There are many types of software faults, each of them with their own impact on the use of software systems. Classifications of software faults provide insight into the factors that lead to programming mistakes and help to prevent these faults in the future. Faults can be classified in several ways according to different criteria: impact in the system (severity), difficulty and cost of repairing, frequency at which they occurred, etc. Taxonomies of software faults have been widely studied in the software testing literature (see e.g. [Basili and Perricone \(1984\)](#), [Beizer \(1990\)](#)[Chapter 2], [Du and Mathur \(1998\)](#), [Sullivan and Chillarege \(1991\)](#) and [Tsipenyuk et al. \(2005\)](#)). One of the main problems with this kind of classifications is that they are *ambiguous*. Most of the authors agree on that their classification schemes may not avoid this ambiguity since the interpretation of the categories is subjected to the point of view of the corresponding fault analyst. The following two classification schemes give a good overview about software fault taxonomies. One of the first classifications of software faults can be found in [Myers \(1979\)](#)[Chapter 3] where faults are classified into seven different categories: data reference (uninitialized variables, array references out of bounds, etc.), data-declaration (variables not declared, attributes of a variable not stated, etc.), computation (division by zero, computations on non-arithmetic variables, etc.), comparison (incorrect Boolean expressions, comparisons between variables of different type, etc.), control-flow (infinite loops,

module does not terminate, etc.), interface (number of input parameters differs from number of arguments, parameter and argument attributes do not match, etc.) and input/output (buffer size does not match record size, files do not open before use, etc.) faults. The second one is due to Basili and Perricone (1984) where software faults are classified into five categories: initialization (fail to initialize a data structure properly), control structure (incorrect path in the program flow), interface (associated with structures outside a module environment), data (incorrect use of a data structure) and computation (erroneous evaluation of the value of a variable). For an overview on different software fault classification schemes we refer to Jorgensen (2002)[Chapter 1]. In this thesis we abstract from the *type* of fault. We only distinguish whether there *is* a fault or not, where we consider a fault as the *deviation from user requirements* mentioned above.

1.1.3 Black-box vs. model-based testing

Two main approaches to software testing are found in the literature, *functional* and *structural* testing. Functional testing considers software systems as a, possibly stateful, function. The function is stateless if repeated application to the same input results in the same output. By stateful functions the result depends on the input and the history. In this case it is said that the system is treated as a *black-box* since no knowledge is assumed about the *internal structure* of the system. Thus, test cases are generated using only the specifications of the software system. The software is subjected to a set of inputs that generates their corresponding outputs which are verified for conformance to the specified behaviour. Note that black-box testing is a *user-oriented* concept since it concerns functionalities and not the implementation. One of the main advantages of black-box testing is precisely that test cases are independent of the implementation procedure. Thus, if the implementation changes, the test cases already generated are still valid. On the other hand, structural testing assumes that some details of the implementation (like programming style, control flow, database design or program code) are known to the tester and these may be used to generate test cases. Depending on the degree of knowledge about internal details of the system, different terms like *white-box*, *clear-box* or *model-based* are used for structural testing. The first two terms are usually used indistinctly to denote the situation where the tester has access to the program code. On the other hand, in model-based testing, test cases are generated based on models that describe part of the behaviour of the system. We are interested in model-based testing, in particular in models describing the control flow over the system components. The models used to describe the software are usually a certain type of *graphs*. Thus, model-based testing has a theoretical background in graph theory. Both black-box and model-based testing are useful but have limitations. For that reason, one should not look at them as two alternative strategies but as complementary.

1.1.4 When to stop testing

A major problem with software testing is to decide when to stop testing and release the software. Even for small software applications, the number of possible test cases

is often so large that, even if they are theoretically finite, which is not always the case, their execution takes unacceptable much time in practice. Since performing exhaustive testing is seldom feasible, statistical procedures to support the decision of stop testing and release the software (with certain statistical confidence) must be considered. Such statistical procedures are mainly based on stochastic models describing the failure detection process experienced during testing. These models are built upon certain assumptions about the failure detection process and usually depend on some parameters. Based on the failure information collected during testing, statistical models are used to estimate quantities like the remaining number of faults in the system, the future detection rate or the additional test effort needed to find a certain number of faults (see e.g. [Grottke \(2003\)](#), [Ohba \(1984\)](#) and [van Pul \(1992a\)](#)). Statistical procedures to support software release decisions are usually based on the optimization of a certain loss function that in general considers the trade-off between the cost of extra testing and the cost of undetected faults. Such procedures are developed from the software producer point of view. On the other hand, release decisions may be based on a *certification* criterion. Certification of certain properties of a system like fault-free system or probability of no failure in a given time period are certified with high probability. Note that unlike the first approach (optimization of loss function), certification procedures are developed from the point of view of software users. Producers must certify that their software performs according to certain reliability requirements. Statistical approaches to software reliability certification can be found in [Currit et al. \(1986\)](#) and [Di Bucchianico et al. \(2008\)](#). However, in these papers certification procedures are developed from a black-box approach. In this thesis we focus on statistical certification procedures for both black-box and model-based testing.

1.2 Goal and outline of the thesis

With this thesis we hope to contribute to the development of *new statistical procedures for certification of software systems* for both black-box and model-based testing. We have developed sequential test procedures to certify, with high confidence, that software systems do not have certain undesirable properties. In particular we focus on: *fault-free period after last failure observation* and *number of remaining faults in the system*. The procedures developed in the black-box context consider a large family of software reliability growth models: *semi-Markov* models with *independent times between failures*. In model-based testing we use a special class of *Petri nets* and finite state automata as the model of software. Practical application of our approach is supported with *software tools*. The main results of this thesis can be found in chapters 5 and 6 for black-box testing, and in chapters 7 and 8 for model-based testing. The remainder of this thesis is organized as follows.

In Chapter 2 we introduce the notation, basic definitions and properties to be used in this thesis, adopting the terminology from [Thompson \(1981\)](#). We emphasize the important role of *stochastic processes* in black-box testing. Based on their basic properties, we propose a classification scheme for software reliability growth models.

We also describe two popular families of models known as *General Order Statistics* (GOS) and *Non-homogeneous Poisson Process* (NHPP) models. We explain how these classes are related in a *Bayesian* way. Besides this, we introduce some of the most widely used models.

In Chapter 3 we present a step-by-step procedure to statistically analyze software failure data. For us statistical analysis of software reliability data should consist of data description, trend tests, initial model type selection, estimation of model parameters, model validation and model interpretation. This approach is similar to the one proposed in Goel (1985) but we provide more insight on each of the steps and give some examples of application. In particular, the problem of initial model selection is just mentioned as a necessary step in Goel (1985) but no explanation about how this should be done is given. In fact, this problem has not been studied in details in the software reliability literature, being Kharchenko et al. (2002) an exception. Moreover, an important step like trend analysis is not considered in Goel (1985). We also focus on model estimation and illustrate some common problems related to Maximum Likelihood (ML) estimation. In general, to obtain the ML estimators of model parameters *numerical optimization* is required. This is often a very difficult problem as shown in Yin and Trivedi (1999). Finally, we point out that further research is needed in this area, especially in problems regarding analysis of *interval-time* data and computation of *confidence intervals* for the parameters of the models.

In Chapter 4 we report on the status of a new software reliability tool to perform statistical analyses of software failure data based on the approach described in chapters 2 and 3. The new tool is a joint project of the Laboratory for Quality Software (LaQuSo) of the Eindhoven University of Technology (www.laquso.com), Refis (www.refis.nl) and the Probability and Statistics group of the Eindhoven University of Technology. This work has been partially presented in Boon et al. (2007), Brandt et al. (2007a) and Brandt et al. (2007b).

Statistical approaches to black-box certification of software systems have not been widely developed in the software reliability literature, Currit et al. (1986) and Di Bucchianico et al. (2008) being exceptions to this. We study their approaches and limitations in detail in Chapter 5. Moreover, in that chapter we present a sequential software release procedure where the certification criterion can be defined as *the next software failure is not observed in a certain time interval*. Our procedure is developed assuming that the failure detection process can be modelled as a *semi-Markov* software reliability growth model with *independent times between failures*. In particular, we consider one NHPP model (Goel-Okumoto), one GOS model (Jelinski-Moranda) and the software reliability model described in Di Bucchianico et al. (2008). In this way, we extend the work presented there with further results. Our procedure also certifies that under certain conditions the global risk taken in the whole procedure (defined as the probability to stop testing too early) can be controlled.

In Chapter 6 we study, via simulation, the performance of our certification procedure for the models considered in Chapter 5. Some of the results shown in chapters 5 and 6 have already been presented in Corro Ramos et al. (2009).

In Chapter 7 we introduce a general framework for model-based testing. Unlike black-box (functional) testing, we now exploit the structure of the system. We consider a model of software systems consisting of a set of components. However, we abstract from the testing of the components themselves (can be done by functional testing) but we concentrate on the control flow over the components. In particular, we use a model of *labelled transition systems* (a special class of Petri nets) where each transition in a labelled transition system represents a software component. We assume that the transitions can either have a correct or an erroneous behaviour. This erroneous behaviour represents the deviation from the requirements defined in Section 1.1.2. However, we do not specify *what* a fault is (in the sense that we do not classify it). For us a fault is a symbolic labelling of a transition. Transitions labelled as erroneous are called *error-marked*. We assume that the number of error-marked transitions is unknown and a fault can only be discovered by executing the corresponding error-marked transition. In this context a *test* is defined as the execution of a *run* through the system. A run is a path that either ends without discovering an erroneous transition (successful run), or it ends in an error-marked transition (failure run). Our main assumption is to consider testing to be *non-anticipative*, i.e., it does not depend on future observed transitions but it may depend on the past (test history). Under this assumption, we prove in Section 7.4 that the error-marking of transitions at the beginning (caused by the programmers) gives the same distribution as error-marking on the fly (when a transition is tested) and that this holds for all possible testing strategies. A testing strategy for labelled transition systems based on reduction techniques is described in Section 7.5. The main idea is that after each successful run (no failure is observed), we increase the probability of visiting unseen transitions. For that reason, we discard for the next run some already visited parts of the system. We show that after a finite number of updates all the transitions are visited, so that the updating procedure is exhaustive.

In Chapter 8 we describe two statistical certification procedures for the testing framework developed in Chapter 7. We consider the process where only the transitions observed for the first time are taken into account. We will refer to this as the *embedded process*. We provide two statistical stopping rules, that are independent of the underlying way of walking through the system, which allows us to stop earlier with a certain statistical reliability. The first rule is based on the probability of having a certain number of remaining error-marked transitions when we decide to stop testing and the second one is based on the survival probability of the system. Like in Chapter 5, we also prove that the global risk can be controlled. Finally, we illustrate our whole approach with an example. Some of the results shown in chapters 7 and 8 have been presented in [Corro Ramos et al. \(2008\)](#).

In chapters 7 and 8 we develop a test procedure for software systems assuming that these can be modelled as a special kind of labelled transition systems. We are interested in two *parameters* of our procedure: the *testing strategy* and the *stopping rule*. In Chapter 9 we discuss how these two parameters may influence the result of the whole test procedure and how to measure their *quality*. Since testing strategies and stopping rules can be very complex, we have no analytical methods to determine their quality in general. Therefore, we have to resort to empirical methods. In order

to do so, we need a population of labelled transitions systems that could be used as benchmark. Instead of fixing some finite set of labelled transition systems, we define a mechanism to generate an infinite population of labelled transition systems, each element having a certain probability of being generated. Based on this approach, we describe a procedure to test the quality of different test procedures. Finally, we present a software tool that can be used to study in an experimental way different test procedures for software systems that can be modelled as labelled transition systems.

CHAPTER 2

PROBABILITY MODELS IN SOFTWARE RELIABILITY AND TESTING

In this chapter we introduce the notation and basic definitions and properties to be used in this thesis. *Software reliability* is defined as the probability of software's successful operation during a certain period of time under specified conditions (see e.g. [Lyu \(1996\)](#)[Chapter 1]). Unlike in hardware reliability, where most of the faults are physical, software does not wear-out during its lifetime. Thus, software reliability will not change over time unless the code or the environmental conditions (for example, user behavior) change. Note that the concept of software reliability is *user-oriented* since it concerns the user expectations about the performance of the software. Since users must be satisfied, the system should be of high quality and thus highly reliable. In that sense, we can regard software reliability as an indicator of quality. Software reliability is hard to achieve in practice due to the complexity of software systems. Such complexity often makes it unfeasible to perform exhaustive testing in practice. Therefore, statistical procedures must be considered in order to support the decision to stop testing and release the software. It was already observed in the early 1970's that reports of failures observed during testing show that the failure detection process follows some patterns (cf. [Ohba \(1984\)](#)). To describe such patterns we use *black-box software reliability growth models*. As mentioned in Chapter 1, when a software system is considered as a black-box it means that we have no knowledge about the internal structure of the system. We discuss software reliability growth models in more detail in Section 2.3. Software reliability growth models are used in order to infer something (statistically) about the future behavior of the system. Inference on software reliability growth models is discussed in Chapter 3. Standard approaches include parameter estimation (allows the estimation of quantities like the expected time until next failure or the expected number of faults left), optimization of a certain loss function (usually based on the trade-off between the cost of future testing effort and the number of remaining faults in the system) and certification of certain properties of the system with high probability (like fault-freeness or probability of no failure in a given time period). Our research is mainly focused on the development of *certification procedures for software systems*.

This chapter is organized as follows. Basic terminology used in probability theory is established in Section 2.1. We are interested in the process of recording the number of observed failures in a software system during testing. We describe how this can be modelled as a *stochastic* process in Section 2.2. We also introduce some basic properties of stochastic processes in that section. In fact, we only consider those properties which are relevant for software reliability. The connection between stochastic processes and software reliability is made in Section 2.3. Some popular classification schemes of software reliability models are presented in Section 2.4. Moreover, we propose a classification scheme based on the basic properties

of stochastic processes studied in Section 2.2. We describe two of the most important families of software reliability models in sections 2.5 and 2.6. These classes are known as the class of *General Order Statistics* and *Non-homogeneous Poisson process* models, respectively. We explain how these classes are related in Section 2.7. A Bayesian approach to our problem is introduced in Section 2.8. Finally, in Section 2.9 we comment on software reliability models that have not been discussed in previous sections.

2.1 Preliminaries

Following the guidelines established in Thompson (1981), we introduce some notation and basic terminology to be used in this thesis. Basic concepts in probability theory will be related to software reliability throughout this chapter. We would like to emphasize that there is often confusion in the literature with certain notions of reliability theory. In particular, the concept of *failure rate* seems to be especially delicate. For example, in well-known software reliability books like Lyu (1996) or Xie et al. (2004), the term failure rate is used in a vague way, which may yield to confusion to the reader. For that reason, we introduce the following concepts with extra care.

We consider a *probability space*, denoted by $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is the *sample space*, usually considered here as \mathbb{R}_+ , \mathcal{F} is a σ -field of subsets of Ω and \mathbb{P} is a probability measure on (Ω, \mathcal{F}) . Any subset \mathcal{B} of Ω such that $\mathcal{B} \in \mathcal{F}$ is called an *event*. A *random variable* T is a function $T : \Omega \rightarrow \mathbb{R}$ such that $\{\omega \in \Omega : T(\omega) \leq t\} \in \mathcal{F}$, for all $t \in \mathbb{R}$. We often use the abbreviation $\{T \leq t\}$ for the corresponding event. All the definitions introduced in the remainder of this section are referred to a single *non-negative* random variable. The probability that T does not exceed $t \geq 0$ is given by the *cumulative distribution function* of T , denoted by $F(t)$ and defined as

$$F(t) = \mathbb{P}[T \leq t] = \int_0^t dF(x) , \quad (2.1)$$

where the above integral is considered in the sense of Riemann-Stieltjes (see e.g. Pitt (1985)[Chapter 6]). The cumulative distribution function of T has the following elementary properties: $F(t)$ is non-decreasing and continuous from the right with $F(\infty) = 1$ and $F(t) = 0$, for all $t < 0$. For details see e.g. Ross (2007)[p. 26]. When $F(t)$ is a step function on \mathbb{R}_+ we say that T is *discrete* and we define the *probability mass function* of T , denoted by $p(t)$, as follows:

$$p(t) = \mathbb{P}[T = t] , \quad (2.2)$$

for all $t = 0, 1, 2, \dots$. In this case, we can write the cumulative distribution function of T as

$$F(t) = \sum_{u=0}^t p(u) . \quad (2.3)$$

When $F(t)$ is *absolutely continuous* we say that T is *continuous*. In this case, the derivative of $F(t)$, denoted by $f(t)$, exists (almost everywhere) for all $t \geq 0$ and it is called the *density function* of T . Note that we can write the cumulative distribution function of T as

$$F(t) = \int_0^t f(u) \, du . \quad (2.4)$$

The *reliability function* or *survival function* of T , denoted by $S(t)$, gives the probability of surviving time $t \geq 0$, i.e.,

$$S(t) = \mathbb{P}[T > t] = 1 - F(t) . \quad (2.5)$$

The study of reliability functions is the core of reliability theory. The *residual lifetime* of T at time t , denoted by $R(t, x)$, is given by

$$R(t, x) = \mathbb{P}[T > t + x \mid T > t] = \frac{S(t + x)}{S(t)} , \quad (2.6)$$

for all $x \geq 0$ and we take $t \geq 0$ such that $S(t) > 0$. This concept is of special interest for us since it allows us to develop a certification procedure in Chapter 5. The *hazard rate* (also called *force of mortality*) of T is defined as follows:

$$h(t) = \lim_{\Delta \rightarrow 0} \frac{\mathbb{P}[t < T \leq t + \Delta \mid T \geq t]}{\Delta} . \quad (2.7)$$

Note that the hazard rate is well-defined: since $F(t)$ is continuous from the right, the limit in (2.7) always exists. Moreover, when $F(t)$ is absolutely continuous, the hazard rate of T (cf. [Rigdon and Basu \(2000\)](#)[Theorem 1]) is given by

$$h(t) = \frac{f(t)}{S(t)} , \quad (2.8)$$

for all $t \geq 0$, provided that $S(t) > 0$. It is also possible to express the cumulative distribution function of T in terms of the hazard rate (see e.g. [Rigdon and Basu \(2000\)](#)[Theorem 2]) as follows:

$$F(t) = 1 - e^{(-\int_0^t h(u) \, du)} . \quad (2.9)$$

The hazard rate is often called *failure rate* (of the random variable T) in the literature. However, this term can be confusing since for *stochastic processes* the failure rate of the *process* can also be defined, as we will see in Section 2.2. In any case, we will avoid to use the term “failure rate” as much as possible. Depending on the monotonicity of $h(t)$ the random variable T is said to have increasing or decreasing hazard rate. This is often also confusing in the literature since this kind of random variables have been usually called increasing (IFR) or decreasing failure rate (DFR), when increasing or decreasing *hazard rate* should be used. All the notation introduced in this section is gathered in Table 2.1. We may use a subindex, for example by writing $F_T(t)$, when we want to emphasize that any of the previous functions refer to a specific random variable, otherwise we simply drop the subindex.

Notation	Definition
T	non-negative random variable
$F(t)$	cumulative distribution function
$p(t)$	probability mass function
$f(t)$	density function
$S(t)$	reliability or survival function
$R(t, x)$	residual lifetime at time t
$h(t)$	hazard rate

Table 2.1: Common notation in probability theory.

2.2 Stochastic processes

Stochastic processes can be used to model the evolution in time of some process whose outcome is random. For example, think of recording the number of observed failures in a software system as a function of time. Formally, a stochastic process, denoted by $(Y(s))_{s \in S}$, is a collection of random variables indexed by the *ordered* set S , i.e., for a fixed $s \in S$, $Y(s)$ is a random variable. The set S is called the *index set*. When the index set S is an interval in \mathbb{R}_+ , the process is said to be a *continuous-time* process. In this case, the index set is often interpreted as time. If the index set S is a countable set (usually considered as the set of non-negative integers \mathbb{Z}_+), then the process is said to be a *discrete-time* process and the random variable $Y(s)$ is usually denoted by Y_s . The *state space* of the process is the set of all possible outcomes of the random variable $Y(s)$. The *Kolmogorov existence theorem* (see e.g. [Grimmett and Stirzaker \(1988\)](#)[p. 229]) states that under certain *consistency conditions* a stochastic process $(Y(s))_{s \in S}$ can be uniquely characterized by the set of all finite dimensional distributions, i.e., for any $n \geq 1$ and $0 < s_1 < \dots < s_n$, the distribution of the process $(Y(s))_{s \in S}$ is uniquely characterized by the joint distribution of $Y(s_1), \dots, Y(s_n)$. Further details and a proof of the theorem can be found in [Rao \(1995\)](#)[Section 1.2]. Repeated application of Bayes rule provides an alternative way to specify the joint distribution of $Y(s_1), \dots, Y(s_n)$ in terms of all the first order conditional distributions, i.e., the distribution of $Y(s_n)$ given $Y(s_1), \dots, Y(s_{n-1})$, for all $n \geq 1$, and $Y(s_0) = Y(0) = 0$. This way of characterizing the process is of special interest for *Markov processes*, as we will see in Section 2.2.2. Stochastic processes have been widely studied in probability theory, being [Cox and Miller \(1984\)](#), [Karlin and Taylor \(1975\)](#), [Rao \(1995\)](#) and [Ross \(1996\)](#) well-known monographs on this subject. Although there exist many types of stochastic processes, we consider in this thesis a special type of process known as *counting process*.

2.2.1 Counting processes

A *counting process* is a continuous-time stochastic process $(N(t))_{t \geq 0}$, with state space \mathbb{Z}_+ , where $t \mapsto N(t)$ is a *non-negative* and *non-decreasing* function of t . In this case $N(t)$ represents the number of *occurrences* that took place until time t . As a consequence, we can define an *occurrence time* of the process as the random point in \mathbb{R}_+ where $N(t)$ changes its state. For example, successive failure occurrences of a software system can be modelled as a stochastic counting process where the random variable $N(t)$, for t fixed, represents the total number of observed failures at time t . Although it is possible to consider counting processes where the initial state is different from zero, we always assume here that $N(0) = 0$. Moreover, we will assume that *two (or more) occurrences do not take place simultaneously*, thus every change of the state of $N(t)$ is of magnitude 1, and that in intervals of finite length only a finite number of occurrences may take place with probability 1. Since $N(t)$ is a non-decreasing function of t , we can define the i^{th} occurrence time of the process as the random variable

$$T_i = \inf \{t \geq 0 \mid N(t) = i\} , \quad (2.10)$$

for all $i \geq 1$. With the convention that $T_0 = 0$, we can also define the times between occurrences as $X_i = T_i - T_{i-1}$, for all $i \geq 1$. The collections of random variables $(T_n)_{n \geq 0}$ and $(X_n)_{n \geq 0}$ are discrete-time stochastic (but not counting) processes with state space \mathbb{R}_+ . It is clear that for any $n \geq 1$ the probability distribution of T_n determines the probability distribution of X_n and vice versa. Moreover, if we know the process $(T_n)_{n \geq 0}$, then we can define $N(t)$ as follows:

$$N(t) = \max \{i \in \mathbb{N} \mid T_i \leq t\} . \quad (2.11)$$

Therefore, the three processes $(N(t))_{t \geq 0}$, $(T_n)_{n \geq 0}$ and $(X_n)_{n \geq 0}$ are equivalent. Since $t \mapsto N(t)$ is right continuous, it follows that for any $0 < \ell_1 < \ell_2$ and $i \geq 1$, the events $\{N(\ell_1) < i \leq N(\ell_2)\}$ and $\{\ell_1 < T_i \leq \ell_2\}$ are equivalent. Thus, for arbitrary $n \geq 1$, $k \geq 1$ and $0 < \ell_1 < \dots < \ell_k$, the process modelled by $(N(t))_{t \geq 0}$, $(T_n)_{n \geq 0}$ or $(X_n)_{n \geq 0}$ can be characterized in an equivalent way by one of these probability distributions:

- the joint distribution of $N(\ell_1), \dots, N(\ell_k)$,
- the joint distribution of T_1, \dots, T_n ,
- the joint distribution of X_1, \dots, X_n .

Counting processes can also be classified in terms of the *mean-value function* of the process, denoted by $\Lambda(t)$, that is defined as the expected number of occurrences at time t , i.e.,

$$\Lambda(t) = \mathbb{E}[N(t)] . \quad (2.12)$$

Its derivative with respect to time is called the *intensity function* and it is denoted by $\lambda(t)$, i.e.,

$$\lambda(t) = \Lambda'(t) . \quad (2.13)$$

Conditions for the existence of $\lambda(t)$ are given in [Thompson \(1981\)](#). Since $\lambda(t)$ is the instantaneous rate of change of the expected number of occurrences with respect to time, it also represents the *occurrence rate of the system*. Another function of interest for the counting process is the following:

$$\mu(t) = \lim_{\Delta \rightarrow 0} \frac{\mathbb{P}[N(t + \Delta) - N(t) \geq 1]}{\Delta} . \quad (2.14)$$

Note that, if $\mu(t)$ exists, then $\Delta\mu(t)$ is the probability of having at least one occurrence in the interval $(t, t + \Delta]$ when $\Delta \rightarrow 0$. Conditions of the existence of $\mu(t)$ are given in [Thompson \(1981\)](#). Only when it is assumed that simultaneous occurrences do not take place and provided that the limit in (2.14) exists, it follows that $\mu(t) = \lambda(t)$. For a proof we also refer to [Thompson \(1981\)](#).

2.2.2 Basic properties

In this section we introduce some basic properties of stochastic processes (cf. [Karlin and Taylor \(1975\)](#)[Section 1.3]). Although the properties presented here are valid for general stochastic processes, we define them in terms of the processes $(N(t))_{t \geq 0}$, $(T_n)_{n \geq 0}$ and $(X_n)_{n \geq 0}$ introduced in Section 2.2.1. Stochastic processes can be classified into different types according to certain properties. In particular, we are interested in processes having the *Markov* property or some other properties related to it.

Semi-Markov process. Let us consider a discrete-time stochastic process $(Y_n)_{n \geq 0}$ with state space S . Suppose that the times where the process changes its state are given by the random variables $T_1 < T_2 < \dots$. The time spent in state Y_j is given by $X_j = T_j - T_{j-1}$, for all $j = 0, 1, 2, \dots$. The *bivariate stochastic process* $(Y_n, X_n)_{n \geq 0}$ is a *Markov renewal sequence* if and only if for any $n \geq 1$ it follows that

$$\begin{aligned} \mathbb{P}[Y_n = y_n, X_n \leq x_n \mid Y_1 = y_1, \dots, Y_{n-1} = y_{n-1}, X_1 = x_1, \dots, X_{n-1} = x_{n-1}] \\ = \mathbb{P}[Y_n = y_n, X_n \leq x_n \mid Y_{n-1} = y_{n-1}] . \end{aligned} \quad (2.15)$$

The process $(Y_n, X_n)_{n \geq 0}$ has the *Markov property*. Its definition is introduced after this one. The points T_n are called *Markov renewal moments* and X_n are often called *sojourn times*. Note that the sojourn times are conditionally independent given Y_1, \dots, Y_{n-1} . Since

$$N(t) = \sup \{i \mid T_i \leq t\} , \quad (2.16)$$

the stochastic process $(Z_t)_{t \geq 0}$ defined by

$$Z_t = Y_{N(t)} , \quad (2.17)$$

for all $t \geq 0$, is said to be a *Semi-Markov process*. Note that $(Z_t)_{t \geq 0}$ is a continuous-time stochastic process with state space S . Moreover, it is clear that Z_t is equal

to Y_j for all $t \in [T_j, T_{j+1})$. The successive states of $(Z_t)_{t \geq 0}$ forms a discrete-time *Markov* process (see e.g. [Kulkarni \(1995\)](#)[Theorem 9.1]). Note also that the process $(Z_t)_{t \geq 0}$ only changes its state at the Markov renewal moments and that a change in the state may imply that the process returns to previous states. The time spent in state Y_j is given by the random variable X_j . Its distribution may depend on the states Y_{j-1} and Y_j but not on all previous ones. In general, we will always consider $T_0 = 0$ and $Y_0 = 0$. Suppose now that we are interested in a counting process $(N(t))_{t \geq 0}$, as defined in Section 2.2.1. In this particular case, the process $(Y_n)_{n \geq 0}$ can be defined as

$$Y_n = n, \quad (2.18)$$

for all $n \in \mathbb{Z}_+$. The times where the process changes its state are given by $T_0 = 0 < T_1 < T_2 < \dots$ and the time spent in state $Y_j = j$ is given by $X_{j+1} = T_{j+1} - T_j$, for all $j = 0, 1, 2, \dots$. Therefore, the Semi-Markov process $(Z_t)_{t \geq 0}$ is now defined as

$$Z_t = Y_{N(t)} = N(t), \quad (2.19)$$

for all $t \geq 0$, where the random variable $N(t)$ represents the total number of observed failures at time t . A realization of a semi-Markov process $(N(t))_{t \geq 0}$ can be observed in Figure 2.1. Note that the process depicted in Figure 2.1 represents a counting

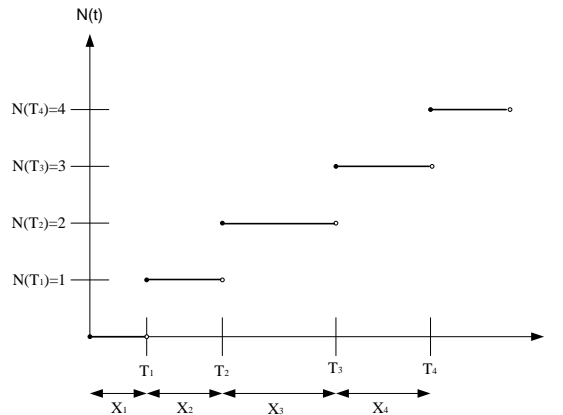


Figure 2.1: Realization of a semi-Markov process $(N(t))_{t \geq 0}$.

process since $N(t)$ is non-decreasing and every change on the state of $N(t)$ is positive and of magnitude 1. For that reason, and unlike for general semi-Markov processes, the process can never return to previous states. Further details on semi-Markov processes can be found in [Pyke \(1961\)](#).

Markov process. A stochastic process $(Y(s))_{s \in \mathcal{S}}$ has the *Markov property* if and only if, for any $n \geq 1$ and $0 < s_1 < \dots < s_n$, it holds that

$$\mathbb{P}[Y(s_n) \in \mathcal{B} \mid Y(s_1), \dots, Y(s_{n-1})] = \mathbb{P}[Y(s_n) \in \mathcal{B} \mid Y(s_{n-1})]. \quad (2.20)$$

In this case, the processes $(Y(s))_{s \in S}$ is said to be a *Markov process*. In particular, the discrete-time stochastic process $(T_n)_{n \geq 0}$ has the *Markov property* if and only if the probability distribution of T_n depends only on T_{n-1} . Note that, if T_{n-1} is given, then the distribution of X_n also depends only on T_{n-1} . The process $(N(t))_{t \geq 0}$ has the Markov property if and only if, for any $0 < \ell_1 < \dots < \ell_n$ and $n \geq 1$, the probability distribution of $N(\ell_n)$ conditional on $N(\ell_1), \dots, N(\ell_{n-1})$ depends only on $N(\ell_{n-1})$, i.e.,

$$\begin{aligned} \mathbb{P}[N(\ell_n) = k_n \mid N(\ell_1) = k_1, \dots, N(\ell_{n-1}) = k_{n-1}] \\ = \mathbb{P}[N(\ell_n) = k_n \mid N(\ell_{n-1}) = k_{n-1}] . \end{aligned} \quad (2.21)$$

As a consequence we can write,

$$\begin{aligned} \mathbb{P}[N(\ell_n) - N(\ell_{n-1}) = k \mid N(\ell_1) = k_1, \dots, N(\ell_{n-1}) = k_{n-1}] \\ = \mathbb{P}[N(\ell_n) - N(\ell_{n-1}) = k \mid N(\ell_{n-1}) = k_{n-1}] . \end{aligned} \quad (2.22)$$

Note that in this case it is more convenient to characterize the process in terms of all the first order conditional distributions since, by the Markov property, these ones depend only on the current state. If the process $(N(t))_{t \geq 0}$ is Markov, then the sojourn times of the process, given by X_n , for all $n \geq 1$, are exponentially distributed (see e.g. [Thompson \(1988\)](#)[p. 73]) although not necessarily identically distributed. Moreover, the converse is also true: if the sojourn times are exponentially distributed, then the process $(N(t))_{t \geq 0}$ is Markov (see e.g. [Kulkarni \(1995\)](#)[Theorem 6.1]). Therefore, a continuous time Markov process is a special case of semi-Markov process where the distributions of the sojourn times are exponentially distributed. Examples of Markov processes are *Birth/Death* processes (see e.g. [Karlin and Taylor \(1975\)](#)[p. 131]) and the *General Order Statistics* process (cf. [Thompson \(1988\)](#)[Chapter 10]).

Independent increments. A stochastic process $(Y(s))_{s \in S}$ has *independent increments* if and only if for any $0 \leq s_1 < s_2 < s_3 < s_4$, the random variables $Y(s_2) - Y(s_1)$ and $Y(s_4) - Y(s_3)$ are independent. In particular, the discrete-time stochastic process $(T_n)_{n \geq 0}$ has independent increments if and only if the times between failures are independent. The process $(N(t))_{t \geq 0}$ has independent increments if and only if for any $0 < \ell_1 < \dots < \ell_n$ and $n \geq 1$, it follows that

$$\begin{aligned} \mathbb{P}[N(\ell_n) - N(\ell_{n-1}) = k \mid N(\ell_1) = k_1, \dots, N(\ell_{n-1}) = k_{n-1}] \\ = \mathbb{P}[N(\ell_n) - N(\ell_{n-1}) = k] . \end{aligned} \quad (2.23)$$

Note that the Markov property is a weaker requirement than the independent increments property. This can be observed by comparing (2.23) and (2.22). An example of a counting process $(N(t))_{t \geq 0}$ with independent increments is the *Poisson* process (see e.g. [Thompson \(1988\)](#)[Chapter 6]). In fact, the *only* counting process that has independent increments are the Poisson processes (see e.g. [Rigdon and Basu \(2000\)](#)[Theorem 15]). In particular, *non-homogeneous Poisson processes* will be studied in details in Section 2.6.

Stationary increments. The discrete-time stochastic process $(T_n)_{n \geq 0}$ has *stationary increments* if and only if the distribution of $T_{n+k} - T_n$ is the same as the distribution of $T_{m+k} - T_m$ for any non-negative integer numbers n, m and k . In particular, for $k = 1$, this means that the times between failures are identically distributed. The counting process $(N(t))_{t \geq 0}$ has stationary increments if and only if the distribution of $N(t+\Delta) - N(t)$ is the same as the distribution of $N(s+\Delta) - N(s)$, for any non-negative real numbers t, s and Δ , i.e.,

$$\mathbb{P}[N(t + \Delta) - N(t) = k] = \mathbb{P}[N(s + \Delta) - N(s) = k] , \quad (2.24)$$

for all $k = 0, 1, 2, \dots$. This means that all the increments of the same length (denoted by Δ) has the same distribution. Examples of counting processes with stationary (and independent) increments are the *Binomial* process (see e.g. Larson and Shubert (1979)[p. 11]), the *homogeneous Poisson* process (see e.g. Thompson (1988)[Chapter 3]) and the *renewal* process (see e.g. Thompson (1988)[Chapter 5]). The stationary increments property characterizes *homogeneous* processes since the probability distribution of the increments of the process does not change in time.

2.2.3 Property implications

In this section we briefly summarize possible relationships between the properties of stochastic processes previously described. In fact, the stationary increments property is not included since the processes we are interested in do not have it, as we will see in Section 2.3.2. Based on those properties and on the relationships among them, we develop a classification scheme for software reliability growth models in Section 2.4.2. Table 2.2 summarizes all possible relationships. Arguments (or refer-

$(N(t))_{t \geq 0}$	$(T_n)_{n \geq 0}$
Semi-Markov	
$\uparrow_{(1)} \quad \not\downarrow_{(2)}$	
Markov	$\Rightarrow_{(3)}$ Markov
$\uparrow_{(4)} \quad \not\downarrow_{(5)}$	$\uparrow_{(6)} \quad \not\downarrow_{(7)}$
Independent Increments	$\not\Rightarrow_{(8)} \quad \not\Leftarrow_{(9)}$ Independent Increments

Table 2.2: Property implications for $(N(t))_{t \geq 0}$ and $(T_n)_{n \geq 0}$.

ences) to prove all these implications are given in the previous section. The following list just contains a brief list of arguments that can be used to prove all the implications.

- (1) Semi-Markov processes are a generalization of Markov processes.
- (2) This is true only when the sojourn times are exponentially distributed (memoryless).
- (3) The distribution of the sojourn times depends on the previous state of the process, thus on T_{n-1} .
- (4) The Markov property generalizes the independent increments property.
- (5) This is true only when the process is Poisson, as we will see in Section 2.6.
- (6) The Markov property generalizes the independent increments property.
- (7) This is true only when the sojourn times are exponentially distributed (memoryless).
- (8) We will see in Section 2.6 that the Poisson process does not have independent sojourn times in general.
- (9) We will see in Section 2.5.1 that for the Jelinski-Moranda model the sojourn times of the process are independent but the process $(N(t))_{t \geq 0}$ is not Poisson.

2.3 Software testing framework

All the concepts for general stochastic processes introduced in Section 2.2 are now put into a *software testing* framework. We first comment on the notation and then we discuss common assumptions in software reliability theory.

2.3.1 Common notation

After a software system has been developed it may contain a certain number of faults that will not change unless some of them are repaired or new ones are introduced. We denote by N the *unknown initial number of faults in a software system*. It can be assumed to be a random variable or deterministic. For example, for the class of models known as *General Order Statistics*, it is assumed to be unknown but *fixed* while for the class of models known as *non-homogeneous Poisson processes* it is considered as a random variable following a Poisson distribution. These two classes of models are studied in detail in sections 2.5 and 2.6, respectively. As mentioned in Section 2.2.1, the process of recording the number of *observed software failures during testing* as a function of time can be modelled as a counting stochastic process $(N(t))_{t \geq 0}$, where the random variable $N(t)$ represents the total number of observed failures at time t . The random variable T_i , representing the random time where the i^{th} software failure is observed, can be interpreted as the i^{th} occurrence time of the process $(N(t))_{t \geq 0}$. In this case, the random variables $T_1 < T_2 < \dots$ are called *failure times* while X_1, X_2, \dots are called *times between failures*. Both failure times and time between failures are usually not identically distributed and may not be independent, as we saw in Section 2.2.2. The mean-value function of the process,

denoted by $\Lambda(t)$, and the failure intensity of the process, denoted by $\lambda(t)$, are also considered here. All this notation will be used in this thesis and it is summarized in Table 2.3 so that the reader may refer back to this table should confusion about notation arise. In general, we use capital Latin letters to denote random variables

Notation	Definition
N	initial number of faults in a software system
n_0	realization of N (when considered random)
$N(t)$	number of failures on $(0, t]$
n	realization of $N(t)$
T_i	failure times
X_i	time between failures
ℓ_i	points on \mathbb{R}_+
$\Lambda(t)$	mean-value function of a process
$\lambda(t)$	intensity function of a process

Table 2.3: Common notation in black-box software reliability.

and the corresponding lower case to denote the realization of the random variable. An exception to this is N which can be random or deterministic depending on the model used. When N is considered to be random we will denote by n_0 the realization of N since n usually denotes the realization of the random variable $N(t)$.

2.3.2 Reliability growth models

Throughout Section 2.2.1 we have emphasized that certain processes, like the number of successive software failures observed during testing (as a function of time), can be modelled as a stochastic counting process. In fact, the choice of the type of stochastic process should depend on the characteristics of the *real* process that we want to include in our *probabilistic* model. Since in reality those characteristics are too many or simply too difficult to model, what is often done is that we first select an *easy-to-study* probabilistic model and then check what kind of assumptions make sense for this process. Our choice for counting processes is mainly justified for three reasons:

- it is a simple class within stochastic processes (and thus well-studied in probability theory),
- it incorporates a considerable number of assumptions,
- a large variety of models can be modelled as counting processes.

First note that independently of the probabilistic model used to describe the real process we must consider assumptions that apply to software testing in general. Assuming that the *software is tested under operational conditions* is considered as universal in the software testing community. Although software systems can reach an enormous level of complexity and can have millions of lines of code, they are *finite*. So is the number of faults in it. Therefore, N must be either a *fixed finite positive* number or a *random variable which is almost surely finite*. If we consider that the process of recording the number of observed software failures during testing can be modelled as a counting process $(N(t))_{t \geq 0}$, then all the assumptions considered for counting processes in Section 2.2.1 are also required in this context. Note that this choice of the counting process as stochastic model implies *immediate repair* of faults, i.e., the time spent repairing faults is not taken into account. If we want to include repairing times in our probabilistic model we need to consider a special type of stochastic process known as *alternating renewal* process (see e.g. Ansell and Phillips (1994)[Section 5.4.4]). However, we do not consider this kind of processes in this thesis. Note that immediate repair can be justified if time is measured as testing effort rather than as calendar time. We assume that *failure observations occur independently from each other*. However, practical experience shows that some failures may cover some other ones (cf. Ohba (1984)) so that failure observations are correlated. The probabilistic model described in Dai et al. (2005) incorporates failure correlation. Depending on the impact that faults have to the system, these can be classified into different *severity* levels. A special type of stochastic process known as *superimposed* process (see e.g. Thompson (1988)[Chapter 7]) considers different types of occurrences. Thus, it can be used to model different severity levels. However, we do not consider this kind of processes in this thesis. It is often assumed that after each failure observation at least one fault in the software is corrected, decreasing in that way the number of remaining faults in the system. This is often called *perfect repair*. If it is possible that the fault remains after reparation, then we speak of *imperfect repair*. Although perfect repair assumption reflects a desirable behaviour during the repairing process it seems to be little realistic since the same failure can be observed several times due to a wrong correction. Moreover, counting processes can incorporate imperfect repair as shown in Goel (1985). It is also often assumed that *faults are repaired without introducing new ones*. This assumption is even less realistic than perfect repair since practical experience also shows that reparation of software faults is in general a complicate process. Therefore, it is very likely that new faults are introduced during such a process. However, as we will see in Section 2.5, the whole class of General Order Statistics models is built upon this assumption. In general, for models like non-homogeneous Poisson process models, it is simply impossible to distinguish between *original* faults and faults *introduced during reparation*. Nevertheless, according to Goel (1985), the additional number of faults introduced during the repair process is very small compare with the total number of faults in the system. Thus, the consideration of this assumption has almost no practical effect. We could relax these two assumptions by considering that the repairing process is *effective* in the sense that the reliability of the system increases with time. For that reason, the models used to describe the failure detec-

tion process of a software system are usually called *reliability growth* models. Under the assumption of reliability growth it is clear that the counting process $(N(t))_{t \geq 0}$ will be *non-homogeneous*. For that reason, the stochastic processes considered here should not have the stationary-increments property introduced in Section 2.2.2 since they cannot model reliability growth. All the assumptions related to software testing and counting processes that we consider are shown in Table 2.4.

Assumptions
Software is tested under operational conditions
The unknown initial number of faults in the system, denoted by N , is fixed and finite or a random variable (almost surely finite)
The number of failures observed at time t , denoted by $N(t)$, is non-decreasing and non-negative
$N(0) = 0$
Simultaneous failures do not occur
In finite length intervals only a finite number of failures may occur
Immediate repair
Independent failure observations
All faults of the same severity
Perfect or imperfect repair
New faults may be introduced during reparation
Effective repair process (reliability growth)

Table 2.4: Software reliability assumptions for counting processes.

After selecting an appropriate stochastic process to describe the real process, the next step consists of studying whether the stochastic process has some properties like those introduced in Section 2.2.2. Based on those properties we can classify the process into different categories. For example, for the counting process $(N(t))_{t \geq 0}$ we can consider the whole class of Markov processes and, within this class, different software reliability growth models arise when different functional forms for $N(t)$ are assumed. This is studied in details in the next section.

2.3.3 Stochastic ordering and reliability growth

The concept of *stochastic order* is of special interest in reliability theory since it can be used to illustrate the idea of *reliability growth*. A random variable U is said to be *stochastically less* than V if $S_U(u) \leq S_V(u)$, for all $u \geq 0$, where $S_U(u)$ and $S_V(u)$ denote the reliability function of U and V , respectively. The following lemma gives

a characterization of stochastic ordering of the times between failures of a Markov counting process in terms of its mean-value function.

Lemma 2.1. *Let $X^{(n)} = (X_1, \dots, X_n)$ be the times between failures of a Markov counting process with mean-value function $\Lambda(t)$ and intensity function $\lambda(t)$. Let $S_{X_n|X^{(n-1)}=x^{(n-1)}}(z | x^{(n-1)})$ denote the reliability function of X_n given X_1, \dots, X_{n-1} , for all $n \geq 1$. If $\Lambda(t)$ is concave for all $t \geq 0$, then*

$$S_{X_n|X^{(n-1)}=x^{(n-1)}}(z | x^{(n-1)}) \leq S_{X_{n+1}|X^{(n)}=x^{(n)}}(z | x^{(n)}),$$

for all $z \geq 0$.

Proof. First note that for any Markov counting process the times between failures are exponentially distributed, as mentioned in Section 2.2.2. Thus, if $t_{n-1} = \sum_{i=1}^{n-1} x_i$, then we can write the reliability function of X_n given X_1, \dots, X_{n-1} (cf. Thompson (1988)[p. 74]) as follows:

$$\begin{aligned} S_{X_n|X^{(n-1)}}(z | x^{(n-1)}) &= \mathbb{P} \left[X_n \geq z \mid X^{(n-1)} = x^{(n-1)} \right] \\ &= e^{-\Lambda(t_{n-1}+z) - \Lambda(t_{n-1})}. \end{aligned}$$

Therefore, X_{n-1} will be (conditionally) stochastically less than X_n if and only if

$$\Lambda(t_n + z) - \Lambda(t_{n-1} + z) \leq \Lambda(t_n) - \Lambda(t_{n-1}). \quad (2.25)$$

Note that if $\Lambda(t)$ is concave for all $t \geq 0$, then $\lambda(t)$ is monotone decreasing. Therefore, we may write (2.25) as

$$\int_{t_{n-1}+z}^{t_n+z} \lambda(u) du \leq \int_{t_{n-1}}^{t_n} \lambda(u) du.$$

Noting that the left-hand side of the above inequality can be written as

$$\int_{t_{n-1}}^{t_n} \lambda(z + u) du,$$

and using that $\lambda(t)$ is monotone decreasing, condition (2.25) holds and therefore the desired stochastic order follows. \square

Note that, in general, not all software reliability growth models have concave mean-value function for all $t \geq 0$. However, if we assume that the software is improved due to the effect of testing and fault reparation, then we can also assume that at some point in time the mean-value function will be concave. This is the case of the S-shaped mean-value function (cf. Section 2.6.2) as can be observed in Figure 2.2.

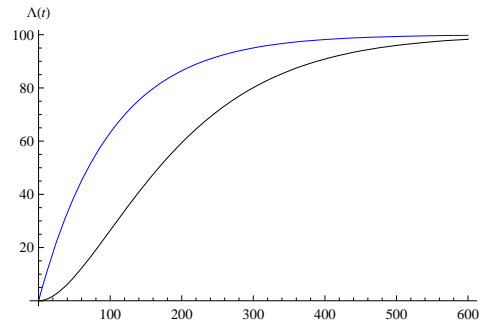


Figure 2.2: Typical shapes of mean-value functions of software reliability growth models. Concave shapes indicate reliability growth.

2.4 Classification of software reliability growth models

The abundance of software reliability models (more than 200 known models according to [Singpurwalla and Wilson \(1994\)](#)) often makes it difficult to select appropriate models for specific problems (this will be studied in more detail in Section 3.3). Studying differences among models and their relationships may help with this task. The development of model classification schemes provides a general overview of existing software reliability models. Moreover, it facilitates the study of the relationships between different models and establishes a good basis for model comparisons.

2.4.1 Previous work on model classification

One of the first classification schemes found in the literature is due to [Ramamoorthy and Bastani \(1982\)](#). In that paper, software reliability models are classified into four different types according to the phase of software development as follows:

- Testing and debugging phase: faults are repaired without introducing new ones, increasing in that way the reliability of the system (*reliability growth models*).
- Validation phase: faults are not corrected and may lead to the rejection of the software. This types of models describe systems for critical applications that must be highly reliable.
- Operational phase: system inputs are selected using a certain probability distribution for a certain (random) period of time.
- Maintenance phase: during this phase activities like fault correction or improvement of implemented features may take place and may modify the reliability of the system. The updated reliability can be estimated using the models for the validation phase.

The models discussed in this thesis belong to the class of *testing and debugging phase* models, therefore *software reliability growth* models. In general we will assume that software reliability growth models are related to a stochastic counting process $(N(t))_{t \geq 0}$, as mentioned in the previous section. Software reliability growth models arise when (implicitly or explicitly) different functional forms for $N(t)$ are assumed.

In [Musa and Okumoto \(1983\)](#) software reliability growth models are classified according to the following attributes:

- Time domain: calendar versus execution time.
- Category: the number of failures that may be experienced in infinite time is finite or infinite.
- Type: probability distribution of $N(t)$ (Poisson, binomial, etc.).
- Class (finite number of failures only): functional form of the failure intensity in terms of time (Exponential, Weibull, etc.).
- Family (infinite number of failures only): functional form of the failure intensity in terms of expected number of failures (geometric, power-law, etc.).

For example, the attributes of the Jelinski-Moranda model (cf. [Jelinski and Moranda \(1972\)](#)) are finite category, binomial type and exponential class. This model will be described in details in Section 2.5.1. Note that from a probabilistic point of view “time domain” is not of special interest since the process $(N(t))_{t \geq 0}$ is defined independently from the way that time is measured. The attribute “category” discriminates between two types of counting processes depending on whether $\mathbb{E}[N(t)] \rightarrow \infty$, as $t \rightarrow \infty$, or not. In particular, we are interested in models for which the above limit is finite. The preference for this type of models is explained in Section 2.7. In any case, we also discuss models with a possibly infinite number of observed failures in Section 2.7.1. Note also that this attribute has further implications. If we assume that software systems always contain a finite number of faults, as explained in Section 2.3.2, then for those models where $\mathbb{E}[N(t)] \rightarrow \infty$, as $t \rightarrow \infty$, holds, the only possibility is that either repair is imperfect or new faults are introduced during reparation. The attribute “type” is often difficult to characterize. Some models are described in terms of the distribution of the failure times or the corresponding times between failures. In that case, the probability distribution of $N(t)$ is not easy to compute in general. Most of the known models consider a binomial or a Poisson distribution for $N(t)$. We will study a class of models of each type in sections 2.5 and 2.6, respectively. Attributes “class” and “family” distinguish between different types of models depending on the functional form of the mean-value function of the process or the probability distribution of the failure times. This classification scheme, although it is quite complete, is criticized in [Kharchenko et al. \(2002\)](#) mainly due to a lack of systematization and connection between the sets of attributes.

A different approach is considered in [Goel \(1985\)](#). As mentioned in Section 2.2, if a software reliability growth model can be characterized in terms of a counting process $(N(t))_{t \geq 0}$, then it can also be described in an equivalent way in terms

of the stochastic process defined by the failures times $(T_n)_{n \geq 0}$ or by the times between failures $(X_n)_{n \geq 0}$. Depending on which of these stochastic processes is chosen for the probabilistic model we speak of *time between failure* and *failure count* models. In fact, strictly speaking, time between failure implies that the process $(T_n)_{n \geq 0}$ is not considered at all, although it is equivalent to $(X_n)_{n \geq 0}$. Thus, this classification scheme is wrong in nature since it does not represent two different types of models but two different types of *characterization* because the processes $(N(t))_{t \geq 0}$ and $(T_n)_{n \geq 0}$ are also equivalent. For example, if the times between failures are exponentially distributed, then, in general, it may be more *convenient* to describe the process in terms of the process $(X_n)_{n \geq 0}$. However, this does not mean that it can *only* be described in this way. This is already discussed in [Singpurwalla and Wilson \(1994\)](#) where software reliability growth models are “classified” according to the *modelling strategy* employed to define the model into *type I models* (models described in terms of the failure times of the process) and *type II models* (models described in terms of the number of observed failures). The most popular class of type I models is the class of the *General Order Statistics* (GOS) models and the most popular class of type II models is the class of *non-homogeneous Poisson process* (NHPP) models. These two classes will be studied in details in sections 2.5 and 2.6, respectively. Moreover, according to [Singpurwalla and Wilson \(1994\)](#), most of the existing software reliability models can be included into these two classes.

The last classification scheme we discuss here is due to [Gokhale et al. \(1996\)](#). Software reliability models are classified into four different types as follows:

- Semi-Markov models: the total number of faults in the system is unknown but finite and fixed. The number of remaining faults in the system at time $t > 0$ is a semi-Markov chain.
- Homogeneous Markov models: the total number of faults in the system is unknown but finite and fixed. The number of remaining faults in the system at time $t > 0$ is a homogeneous Markov chain.
- Non-homogeneous Markov models: the total number of faults in the system is assumed to be a random variable. The number of discovered faults in the system at time $t > 0$ is a non-homogeneous Markov chain.
- Other models: those models that cannot be classified into the previous types.

As mentioned in Section 2.2.2, semi-Markov processes are a generalization of Markov processes. Thus, the second type of models described here is a subclass of the first one. In fact, we can consider here two main groups of models depending on the nature of N : fixed or random. We should emphasize again that the process $(N(t))_{t \geq 0}$ is defined independently from the fact that N is considered to be fixed or a random variable. For that reason, we consider that software reliability growth models should not be classified attending by the nature of N as a primary criterion. Note also that the first two types of models are described in terms of the number of remaining faults in the system. In this case, the process considered is denoted by $(R(t))_{t \geq 0}$, where $R(t) = N - N(t)$. Note that this approach is only valid when N is

fixed, finite and $N(t) \rightarrow N$, as $t \rightarrow \infty$. However, as mentioned in Section 2.3, this is not always the case. For that reason, we find it more convenient to classify the models in terms of the “original” process $(N(t))_{t \geq 0}$ as it is done for the third type of models described here.

2.4.2 Classification based on properties of stochastic processes

In Section 2.2.2 we described some basic properties of stochastic processes. Afterwards, in Section 2.3, we explained how the number of observed failures of a software system at time t can be modelled as a counting process $(N(t))_{t \geq 0}$ or, equivalently, as a stochastic process $(T_n)_{n \geq 0}$, representing the failure times of the software during testing. For that reason, we propose to classify software reliability growth models according to the properties of the stochastic processes $(N(t))_{t \geq 0}$ and $(T_n)_{n \geq 0}$ *independently on the way that the process is characterized*. Therefore, we consider models where $(N(t))_{t \geq 0}$ may have the semi-Markov, Markov or independent increments properties, and $(T_n)_{n \geq 0}$ may have the Markov or independent increments properties. Some of the most well-known models are classified according to these properties in Table 2.5. In order to interpret Table 2.5 properly we should keep in

$(N(t))_{t \geq 0} \backslash (T_n)_{n \geq 0}$	Markov	ind. incr.
semi-Markov		Schick and Wolverton (1978) Shanthikumar (1981) etc.
Markov	GOS models (cf. Section 2.5)	Jelinski and Moranda (1972) Moranda (1979) Goel and Okumoto (1979) Musa (1979) Goel (1985) Abdel-Ghaly et al. (1986) Xie (1990) etc.
ind. incr.	NHPP models (cf. Section 2.5) Duane (1964) Goel and Okumoto (1978) Musa and Okumoto (1984) Yamada et al. (1984) Xie and Zhao (1993) etc.	HPP (no growth)

Table 2.5: Software reliability growth models classification.

mind the property implications explained in Section 2.2.3. For example, the semi-Markov property generalizes the Markov property, which, in turn, generalizes the

independent increments property. Thus, the models in Table 2.5 are classified into the most restrictive class they belong to since, in this case, they are all semi-Markov. Different software reliability growth models of the same type (same cell in Table 2.5) are considered depending on the functional form adopted for the process $(N(t))_{t \geq 0}$ or $(T_n)_{n \geq 0}$. In particular, these are classified based on the functions $\Lambda(t)$, or $\lambda(t)$ or the probability distribution of $N(t)$, for the process $(N(t))_{t \geq 0}$, and the functions $F_{T_i}(t)$, $f_{T_i}(t)$ or $h_{T_i}(t)$ (or the corresponding function for X_i), for all $i \geq 1$, for the process $(T_n)_{n \geq 0}$.

As mentioned throughout this chapter, we are considering a simple class of software reliability models based on the assumption that they can be modelled as counting processes. However, it should be noted that our classification scheme is also valid independently of the type of stochastic process considered. Thus, in general, software reliability growth models can be classified according to the following characteristics:

- type of stochastic process (counting process, alternating renewal process (cf. [Ansell and Phillips \(1994\)](#)[Section 5.4.4]), superimposed process (cf. [Thompson \(1988\)](#)[Chapter 7]), ...),
- properties of the process (semi-Markov process, Markov process, ...),
- functional form of the process (exponential times between failures, power-law mean-value function, ...).

Model classification establishes a good basis for model selection, as we will see in Section 3.3. Systematic approaches to use model assumptions and data requirements for initial model selection facilitates this task although they are not easy to find in the literature (see e.g. [Kharchenko et al. \(2002\)](#)). In the remainder of this chapter we study general features of some of the most widely used software reliability growth models paying special attention to GOS and NHPP models.

2.5 General order statistics models

In this section we describe an important class of software reliability growth models known as *General Order Statistics* (GOS). The main assumption for this class of models is that the times between failures of a software system can be defined as the differences between two consecutive *order statistics*.

Definition 2.1 (Order statistics). *Suppose that Z_1, \dots, Z_m are m independent and identically distributed (i.i.d.) random variables. Then, the random variable*

$$Z_{(1)} = \min \{Z_1, \dots, Z_m\}$$

is called the first order statistic, the random variable

$$Z_{(m)} = \max \{Z_1, \dots, Z_m\}$$

is called the m^{th} order statistic and the random variable

$$Z_{(i)} = \min \{ \{Z_1, \dots, Z_m\} \setminus \{Z_{(1)}, \dots, Z_{(i-1)}\} \} ,$$

for all $i = 2, \dots, m - 1$, is called the i^{th} order statistic.

In case of GOS models, we assume that the initial number of faults in a software system, denoted by N , is unknown but *fixed* and *finite*. Thus, for any $n \leq N$, we can interpret the first n failure times $T_1 < T_2 < \dots < T_n$ as the *first n order statistics* of a random sample Z_1, Z_2, \dots, Z_N . If we assume that software is tested until a failure is observed, then the random variable Z_i represents the time at which the i^{th} fault occurs and $T_i = Z_{(i)}$ represents the time at which one among the N initial faults is the i^{th} to be observed. The times between failures are defined in this case as the difference of two order statistics, i.e.,

$$X_i = T_i - T_{i-1} = Z_{(i)} - Z_{(i-1)} ,$$

for all $i \geq 1$. In general, the times between failures of GOS models are *not independent nor identically distributed*, as shown in Pyke (1965)[Section 2.2]. In Section 2.5.1 we will see that in the particular case of the Exponential order statistics the times between failures are independent although not identically distributed. The next results show basic properties about the distribution of the order statistics that are needed in order to properly define specific GOS models. Since these results are well-known in probability theory we skip the proofs here. For further details we refer to David and Nagaraja (2003)[Chapter 2].

Theorem 2.1 (Distribution of the order statistics). *Let Z_1, \dots, Z_n be i.i.d. continuous random variables with common distribution function $F(z)$ and density function $f(z)$. The density function of $Z_{(i)}$ is given by*

$$f_{(i)}(z) = \frac{n!}{(i-1)!(n-i)!} f(z) (F(z))^{i-1} (1-F(z))^{n-i} .$$

From Theorem 2.1 one can easily deduce that the cumulative distribution function of the i^{th} order statistic $Z_{(i)}$ is given by

$$F_{(i)}(z) = \mathbb{P} [Z_{(i)} \leq z] = \sum_{k=i}^n \binom{n}{k} (F(z))^k (1-F(z))^{n-k} . \quad (2.26)$$

Note that (2.26) represents the cumulative distribution function of a binomial distribution with parameters n and $F(z)$. As mentioned in Section 2.2, the events $\{N(z) \geq i\}$ and $\{Z_{(i)} \leq z\}$ are equivalent. Therefore, for $t > 0$ fixed, the random variable $N(t)$ follows a binomial distribution. For that reason, in the software reliability literature, the class of GOS models is often called the class of *binomial models* (see the classification in Musa and Okumoto (1983) discussed in the previous section). The next two results characterize the joint distribution of order statistics. We skip the proofs here and refer to David and Nagaraja (2003)[Chapter 2] for further details.

Theorem 2.2 (Joint distribution of two order statistics). *Let Z_1, \dots, Z_n be i.i.d. continuous random variables with common distribution function $F(z)$ and density function $f(z)$. The joint density function of $Z_{(i)}$ and $Z_{(j)}$, $i < j$, is given by*

$$f_{(i,j)}(z_1, z_2) = \frac{n!f(z_1)f(z_2)(F(z_1))^{i-1}(F(z_2) - F(z_1))^{j-i-1}(1 - F(z_2))^{n-j}}{(i-1)!(j-i-1)!(n-j)} .$$

Corollary 2.3 (Joint distribution of all order statistics). *Let Z_1, \dots, Z_n be i.i.d. continuous random variables with common density function $f(z)$. Then, the joint density function of $Z_{(1)}, Z_{(2)}, \dots, Z_{(n)}$ is given by*

$$f_{(1,\dots,n)}(z_1, \dots, z_n) = n!f(z_1) \dots f(z_n) .$$

GOS models are usually characterize in terms of the probability distribution of the failure times. However, as observed in (2.26), it is also possible to describe them in terms of the probability distribution of $N(t)$. The next result shows that, independently on the way that the process is characterized, for the class of GOS models the stochastic processes $(N(t))_{t \geq 0}$ and $(T_n)_{n \geq 0}$ have both the Markov property.

Lemma 2.2 (GOS classification). *The stochastic processes $(N(t))_{t \geq 0}$ and $(T_n)_{n \geq 0}$ defined by the general order statistics are both Markov processes.*

Proof. Let us first show that $(N(t))_{t \geq 0}$ is a Markov process. If $F(z)$ denotes the common cumulative distribution function of Z_1, \dots, Z_N , then for any $n \geq 1$ and $0 < \ell_1 < \dots < \ell_n$, it follows that

$$\begin{aligned} & \mathbb{P}[N(\ell_n) = k_n \mid N(\ell_1) = k_1, \dots, N(\ell_{n-1}) = k_{n-1}] \\ &= \binom{N - k_{n-1}}{k_n - k_{n-1}} \left(\frac{F(\ell_n) - F(\ell_{n-1})}{1 - F(\ell_{n-1})} \right)^{k_n - k_{n-1}} \left(\frac{1 - F(\ell_n)}{1 - F(\ell_{n-1})} \right)^{N - k_n} \\ &= \mathbb{P}[N(\ell_n) = k_n \mid N(\ell_{n-1}) = k_{n-1}] . \end{aligned} \quad (2.27)$$

Note that (2.27) is precisely the Markov property for $(N(t))_{t \geq 0}$. We prove now that $(T_n)_{n \geq 0}$ is also a Markov process. As a generalization of Theorem 2.2 (cf. David and Nagaraja (2003)[p. 12]), we can write the joint density function of the first n order statistics as

$$f_{T_1, \dots, T_n}(t_1, \dots, t_n) = \frac{N!}{(N-n)!} f(t_1) \dots f(t_n) (1 - F(t_n))^{N-n} , \quad (2.28)$$

for all $t_i \geq 0$. Therefore, the conditional density function of T_n given $T_1 = t_1, \dots, T_{n-1} = t_{n-1}$ can be written as

$$\begin{aligned} f_{T_n|T_1, \dots, T_{n-1}}(t_n) &= \frac{f_{T_1, \dots, T_n}(t_1, \dots, t_n)}{f_{T_1, \dots, T_{n-1}}(t_1, \dots, t_{n-1})} \\ &= (N - n + 1)f(t_n) \frac{(1 - F(t_n))^{N-n}}{(1 - F(t_{n-1}))^{N-n+1}} \\ &= f_{T_n|T_{n-1}}(t_n) , \end{aligned} \quad (2.29)$$

for all $t_i \geq 0$ such that $f_{T_1, \dots, T_{n-1}}(t_1, \dots, t_{n-1}) > 0$ (in fact, it only depends on t_{n-1}). Since the Jacobian determinant for the change of variables $X_i = T_i - T_{i-1}$ is equal to 1, it follows that

$$f_{X_1, \dots, X_n}(x_1, \dots, x_n) = f_{T_1, \dots, T_n}(x_1, \dots, x_1 + \dots + x_n). \quad (2.30)$$

Hence, we can write (2.29) as

$$f_{X_n|T_1, \dots, T_{n-1}}(x_n) = f_{X_n|T_{n-1}}(x_n). \quad (2.31)$$

Finally, note that (2.31) is precisely the Markov property for $(T_n)_{n \geq 0}$. \square

Different GOS models arise when one considers different distributions for the failure times. The most well-known GOS model is based on the Exponential distribution and it is due to [Jelinski and Moranda \(1972\)](#). For a detailed study of this model we refer to [Miller \(1986\)](#). Another two popular GOS models consider the Weibull and Pareto distribution for the order statistics. These two models are studied in [Abdel-Ghaly et al. \(1986\)](#) and [Raftery \(1987\)](#) among others. In the remainder of this section we describe in details the Jelinski-Moranda model and comment its discrete version when the geometric distribution is considered.

2.5.1 Jelinski-Moranda model

The Jelinski-Moranda model was first introduced as a software reliability growth model in [Jelinski and Moranda \(1972\)](#). The distribution of the order statistics is the Exponential distribution. The assumptions for this model can be found in [Lyu \(1996\)](#)[Chapter 3] and [Xie et al. \(2004\)](#)[Chapter 4]. However, as explained in Section 2.1, these assumptions are often presented in the literature in an unclear way. The main assumptions for the Jelinski-Moranda model are the following:

- (1) The number of initial faults in the system is unknown but finite and fixed.
- (2) All faults are of the same type.
- (3) Immediate and perfect repair of faults.
- (4) Faults are detected independently of each other.
- (5) The times between failures are exponentially distributed with parameter proportional to the number of remaining faults.
- (6) The hazard rate remains constant over the interval between fault occurrences.

One of the most widely discussed assumptions of the Jelinski-Moranda model is (2) since it implies that each repaired fault reduces the hazard rate of the new time between failure by a constant $\lambda > 0$. This idea is depicted in Figure 2.3. However, practical experience shows that some faults are easier to detect than others (cf. [Ohba \(1984\)](#), [Yamada and Osaki \(1984\)](#)). The Jelinski-Moranda model is usually characterized in terms of the distribution of the times between failures as follows. Suppose

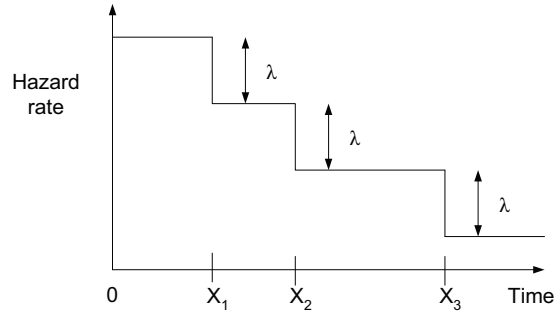


Figure 2.3: Jelinski-Moranda model hazard rate. It remains constant between failure observations and decreases by a factor λ after a fault is repaired.

that Z_1, \dots, Z_N are N i.i.d. random variables having an Exponential distribution with parameter $\lambda > 0$. Thus,

$$f_i(z_i) = \lambda e^{-\lambda z_i}$$

is the density function of Z_i for all $z_i \geq 0$ and $i = 1, \dots, N$. Let $Z_{(i)}$ be the i^{th} order statistic and define $Z_{(0)} = 0$. Note that $T_i = Z_{(i)}$, for all $i = 1, \dots, N$. According to Corollary 2.3 we have in this case that

$$f_{(1, \dots, N)}(z_1, \dots, z_N) = N! \lambda^N e^{-\lambda \sum_{i=1}^N z_i} . \quad (2.32)$$

The times between failures are given by $X_i = Z_{(i)} - Z_{(i-1)}$, for all $i = 1, \dots, N$, and $Z_{(0)} = 0$. Note that the Jacobian determinant for this change of variables is equal to 1. Note also that

$$\sum_{i=1}^N Z_i = \sum_{i=1}^N Z_{(i)} = \sum_{i=1}^N (N - i + 1) X_i .$$

Since $f_{(1, \dots, N)}(z_1, \dots, z_N)$ in (2.32) only depends on $\sum_{i=1}^N z_i$, it follows that

$$f_{(1, \dots, N)}(z_1, \dots, z_N) = f(x_1, \dots, x_N) = N! \lambda^N e^{-\lambda \sum_{i=1}^N (N - i + 1) x_i} .$$

Note also that $f(x_1, \dots, x_N)$ is the joint density function of X_1, \dots, X_N , since the marginal density function of X_i can be obtained by integrating $f(x_1, \dots, x_N)$ with respect to X_j , for all $j \neq i$, and it is given by

$$f_i(x_i) = (N - i + 1) \lambda e^{-(N - i + 1) \lambda x_i} . \quad (2.33)$$

Moreover, we may write

$$\begin{aligned} f(x_1, \dots, x_N) &= N! \lambda^N e^{-\lambda \sum_{i=1}^N (N-i+1)x_i} \\ &= \prod_{i=1}^N (N-i+1) \lambda e^{-(N-i+1)\lambda x_i} \\ &= \prod_{i=1}^N f_i(x_i). \end{aligned}$$

Thus, the times between failures are independent exponentially distributed random variables with parameter $(N-i+1)\lambda$ for all $i = 1, \dots, N$. Since the times between failures are exponentially distributed, the hazard rate after the i^{th} failure detection is constant and equal to the parameter of the distribution, i.e.,

$$h_i(x) = (N-i+1)\lambda,$$

for all $i = 1, \dots, N$ and $x > 0$. We proved in Lemma 2.2 that for all GOS models the processes $(N(t))_{t \geq 0}$ and $(T_n)_{n \geq 0}$ are Markov. In particular, for the Jelinski-Moranda model, the times between failures are independent. Therefore, the process $(T_n)_{n \geq 0}$ has also independent increments. However, this does not imply that the process $(N(t))_{t \geq 0}$ also has independent increments (see Section 2.2.3). In fact, we can write (2.27) as follows:

$$\begin{aligned} &\mathbb{P}[N(\ell_n) = k_n \mid N(\ell_{n-1}) = k_{n-1}] \\ &= \binom{N - k_{n-1}}{k_n - k_{n-1}} \left(1 - e^{-(\ell_n - \ell_{n-1})\lambda}\right)^{k_n - k_{n-1}} \times \\ &\quad \times e^{-(N - k_n)(k_n - k_{n-1})(\ell_n - \ell_{n-1})\lambda}. \end{aligned} \quad (2.34)$$

As a consequence of Theorem 2.1, we know that the random variable $N(t)$ follows a binomial distribution with parameters N and $F(t) = 1 - e^{-\lambda t}$. Thus,

$$\mathbb{P}[N(\ell_n) = k_n] = \binom{N}{k_{n-1}} (1 - e^{-\lambda \ell_n})^{k_n} e^{-(N - k_n)\lambda \ell_n}. \quad (2.35)$$

Since (2.34) and (2.35) are not equal, it follows that the process $(N(t))_{t \geq 0}$ does not have independent increments.

2.5.2 Geometric order statistics model

Discrete-time software reliability growth models have not received as much attention as continuous-time models, Di Bucchianico et al. (2008), Fries and Sen (1996), Okamura et al. (2004) and Yamada et al. (1986) being some of the exceptions. For this kind of models time can be considered as the number of runs or test cases needed to observe a failure. A natural discrete version of the Jelinski-Moranda model appears when one considers Geometric order statistics instead of Exponential ones. Since the Geometric distribution is also memoryless, it can be seen as the discrete version

of the Exponential distribution. For that reason, all the assumptions and remarks for the Jelinski-Moranda model are also valid here. If $p > 0$ denotes the parameter of the Geometric distribution, then, following the same as steps as for the Jelinski-Moranda model, we can characterize the Geometric order statistics model by the probability mass function of the times between failures as follows:

$$q_i(x_i) = p(N - i + 1)(1 - p)^{(N-i+1)x_i-1}, \quad (2.36)$$

for all $i = 1, \dots, N$. For more details on this model we refer to [Okamura et al. \(2004\)](#).

2.6 Non-homogenous Poisson process models

In this section we describe the class of software reliability growth models known as *non-homogeneous Poisson processes* (NHPP). The main assumption for this class of models is that the counting process $(N(t))_{t \geq 0}$ is a *Poisson process*.

Definition 2.2. A counting process $(N(t))_{t \geq 0}$ is said to be a non-homogeneous Poisson process (NHPP) with intensity function $\lambda(t)$, for all $t \geq 0$, if the following properties hold:

- (1) $N(0) = 0$.
- (2) $(N(t))_{t \geq 0}$ has independent increments: for any $0 \leq t_1 < t_2 < t_3 < t_4$ the random variables $N(t_2) - N(t_1)$ and $N(t_4) - N(t_3)$ are independent.
- (3) The random variable $N(t_2) - N(t_1)$ has a Poisson distribution with mean $\Lambda(t_2) - \Lambda(t_1)$, for all $0 \leq t_1 < t_2$, i.e.,

$$\mathbb{P}[N(t_2) - N(t_1) = k] = e^{-(\Lambda(t_2) - \Lambda(t_1))} \frac{(\Lambda(t_2) - \Lambda(t_1))^k}{k!},$$

for all $k = 0, 1, 2, \dots$, where $\Lambda(t) = \int_0^t \lambda(x) dx$ is the mean-value function of the process.

Note that $\Lambda(t)$ and $\lambda(t)$ are the mean-value function and the intensity function of the process, respectively, as defined in Section 2.2.1. When the intensity function is constant, then the Poisson process is said to be *homogeneous* and it is denoted by HPP. Note that unlike the HPP, the times between failures of an NHPP are neither independent nor identically distributed (see e.g. [Thompson \(1988\)](#)[p. 57]). Only the increments are independent although not identically distributed. In fact, the *only* counting process that has independent increments is the Poisson process (cf. [Rigdon and Basu \(2000\)](#)[Theorem 15]). The next result shows an important property of the Poisson process. This result is used for example in the construction of trend tests as we will see in Section 3.2 and relates NHPP models with GOS models. Since this result is well-known in probability theory, we skip the proof here. For further details, we refer to [Rigdon and Basu \(2000\)](#)[Theorem 25].

Theorem 2.4. *Conditional on the event $\{T_n = t_n\}$, the failure times of an NHPP $T_1 < T_2 < \dots < T_{n-1}$ are distributed as $n - 1$ order statistics from a distribution with cumulative distribution function*

$$F(y) = \begin{cases} 0, & y \leq 0 \\ \Lambda(y)/\Lambda(t_n), & 0 < y \leq t_n \\ 1, & y > t_n \end{cases}$$

Note that when the process is homogeneous it follows that $\Lambda(t) = t$. Therefore, in that case $T_1 < T_2 < \dots < T_{n-1}$ are distributed as $n - 1$ order statistics from a *Uniform* distribution on $(0, t_n)$ (cf. [Rigdon and Basu \(2000\)](#)[Theorem 21]). Note that we obtain a similar result when the failure times of an NHPP are $T_1 < T_2 < \dots < T_{N(t)}$, where $N(t)$ is the number of failures at time $t > 0$. In this case, conditional on the event $\{N(t) = n\}$, the failure times $T_1 < T_2 < \dots < T_n$ are distributed as n order statistics from a distribution with cumulative distribution function $\Lambda(y)/\Lambda(t)$, on the interval $(0, t)$ (cf. [Rigdon and Basu \(2000\)](#)[Theorem 26]). As done for GOS models in Lemma 2.2, we now classify the whole class of NHPP models according to the properties introduced in Section 2.2.2. Note that, by Definition 2.2, the process $(N(t))_{t \geq 0}$ has independent increments, therefore it is Markov. The next lemma shows that the process $(T_n)_{n \geq 0}$ is also Markov. Note that $(T_n)_{n \geq 0}$ does not have independent increments since this would imply that X_i are independent for all $i \geq 1$. The times between failures are not independent in general for NHPP as shown in [Thompson \(1988\)](#)[p. 57].

Lemma 2.3 (NHPP classification). *The stochastic process $(T_n)_{n \geq 0}$ defined by an NHPP has the Markov property.*

Proof. The joint density function of T_1, \dots, T_n (see e.g. [Thompson \(1988\)](#)[p. 56]) is given by

$$f_{T_1, \dots, T_n}(t_1, \dots, t_n) = \lambda(t_1) \dots \lambda(t_n) e^{-\Lambda(t_n)}, \quad (2.37)$$

for all $t_i \geq 0$. Therefore, the conditional density function of T_n given $T_1 = t_1, \dots, T_{n-1} = t_{n-1}$ can be written as

$$\begin{aligned} f_{T_n|T_1, \dots, T_{n-1}}(t_n) &= \frac{f_{T_1, \dots, T_n}(t_1, \dots, t_n)}{f_{T_1, \dots, T_{n-1}}(t_1, \dots, t_{n-1})} \\ &= \lambda(t_n) e^{-(\Lambda(t_n) - \Lambda(t_{n-1}))} \\ &= f_{T_n|T_{n-1}}(t_n), \end{aligned} \quad (2.38)$$

for all $t_i \geq 0$ such that $f_{T_1, \dots, T_{n-1}}(t_1, \dots, t_{n-1}) > 0$ (in fact, it only depends on t_{n-1}). Since the Jacobian determinant for the change of variables $X_i = T_i - T_{i-1}$ is equal to 1, it follows that

$$f_{X_1, \dots, X_n}(x_1, \dots, x_n) = f_{T_1, \dots, T_n}(x_1, \dots, x_1 + \dots + x_n). \quad (2.39)$$

Hence, we can write (2.29) as

$$f_{X_n|T_1, \dots, T_{n-1}}(x_n) = f_{X_n|T_{n-1}}(x_n). \quad (2.40)$$

Finally, note that (2.40) is precisely the Markov property for $(T_n)_{n \geq 0}$. \square

Different NHPP models arise when one considers different mean-value or failure intensity functions. NHPP models can be further classified depending on whether $\Lambda(t)$ converges to a finite value or to infinite when $t \rightarrow \infty$ (see attribute ‘‘category’’ in the classification scheme by [Musa and Okumoto \(1983\)](#)). We will refer to these subclasses as *NHPP-finite* and *NHPP-infinite*, respectively. A relationship between NHPP-finite and GOS models is presented in [Langberg and Singpurwalla \(1985\)](#). It is shown that when N in GOS models is considered to be a Poisson random variable, an equivalent NHPP model is obtained. We study this relationship in details in Section 2.7. Some properties of NHPP-infinite models are discussed in Section 2.7.1. The asymptotic behaviour for both types of NHPP models is given by the following result. For a proof we refer to [Thompson \(1988\)](#)[p. 57].

Theorem 2.5 (Asymptotic behaviour of the NHPP). *Let $(N(t))_{t \geq 0}$ be an NHPP with mean-value function $\Lambda(t)$, for all $t \geq 0$.*

1. *If $\lim_{t \rightarrow \infty} \Lambda(t) = \Lambda_0 < \infty$, then, in distribution,*

$$N(t) \longrightarrow Y ,$$

where Y is a Poisson distributed random variable with mean Λ_0 .

2. *If $\lim_{t \rightarrow \infty} \Lambda(t) = \infty$, then, in distribution,*

$$N(t)/\Lambda(t) \longrightarrow 1$$

and

$$(N(t) - \Lambda(t))/\sqrt{\Lambda(t)} \longrightarrow Z ,$$

where Z is a standard normal distributed random variable.

There exist many NHPP models in the literature. For a description of some of them we refer to [Musa et al. \(1987\)](#)[Chapter 11], [Lyu \(1996\)](#)[Chapter 3] and [Xie et al. \(2004\)](#)[Section 4.5]. In the remainder of this section we describe in details two NHPP-finite models (Goel-Okumoto and Yamada S-shaped) and one NHPP-infinite model (Duane).

2.6.1 Goel-Okumoto model

Presented in [Goel and Okumoto \(1978\)](#), this is the most well-known NHPP model. Due to the important role that this model has played on the software reliability modelling history, it is often called ‘‘the’’ NHPP model. Assumptions (2), (3) and (4) for the Jelinski-Moranda model are also valid for the Goel-Okumoto model.

Assumption (1) is not valid here since N is considered as a Poisson random variable. The mean-value function and the intensity function are given by

$$\Lambda(t) = a(1 - e^{-bt}) ,$$

and

$$\lambda(t) = abe^{-bt} ,$$

respectively, for all $t \geq 0$, where $a > 0$ and $b > 0$ are the parameters of the model. The parameter a is the expected number of faults to be eventually detected while b is the rate at which each individual fault will be detected during testing. In fact, it follows that

$$\lim_{t \rightarrow \infty} \Lambda(t) = a .$$

A typical plot of $\Lambda(t)$ for the Goel-Okumoto model can be observed in Figure 2.4 where $\Lambda(t)$ is plotted when $a = 11$ and $b = 0.14$. Note that a determines the

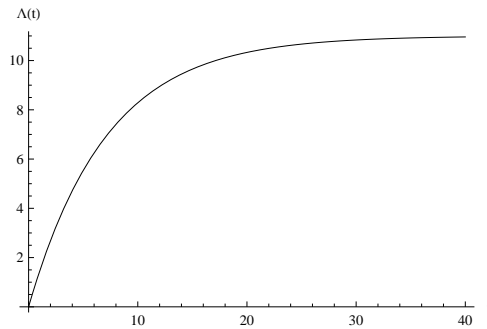


Figure 2.4: Goel-Okumoto model mean-value function when $a = 11$ and $b = 0.14$.

scale and b the *shape* of the mean-value function. We will see in Section 2.7 that assuming a Poisson distribution for N in the Jelinski-Moranda model one obtains the Goel-Okumoto model.

2.6.2 Yamada S-shaped model

The so-called S-shaped model was presented in Yamada and Osaki (1984). It receives the name S-shaped because the curve of the mean-value function is often S-shaped (in comparison with the exponential-shaped mean-value function of the Goel-Okumoto model). This can be observed in Figure 2.5 where $\Lambda(t)$ is plotted when $a = 11$ and $b = 0.14$. The S-shaped NHPP model has mean-value function

$$\Lambda(t) = a(1 - (1 + bt)e^{-bt}) ,$$

and intensity function

$$\lambda(t) = ab^2te^{-bt} ,$$

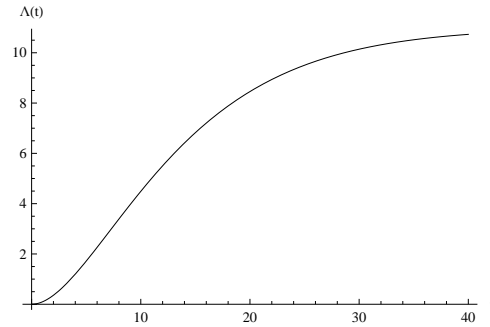


Figure 2.5: Yamada S-shaped model mean-value function when $a = 11$ and $b = 0.14$.

respectively, for all $t \geq 0$, where $a > 0$ and $b > 0$ are the parameters of the model. The parameters of the model have the same interpretation as in the Goel-Okumoto model. Note that also in this case it follows that

$$\lim_{t \rightarrow \infty} \Lambda(t) = a .$$

Therefore, this is also an NHPP-finite model.

2.6.3 Duane (power-law) model

This NHPP model was first introduced for hardware reliability in [Duane \(1964\)](#). It is also known as the power-law or the Crow model after [Crow \(1974\)](#). The assumptions for this model (cf. [Lyu \(1996\)](#)[Chapter 3]) are the same as those for the Goel-Okumoto model but an infinite number of failures in infinite time is allowed. The mean-value function and the intensity function are given by

$$\Lambda(t) = \left(\frac{t}{a} \right)^b ,$$

and

$$\lambda(t) = \frac{b}{a} \left(\frac{t}{a} \right)^{b-1} ,$$

respectively, for all $t \geq 0$, where $a > 0$ and $b > 0$ are the parameters of the model. The parameter a is the scale parameter, while b is the shape parameter. A typical plot of $\Lambda(t)$ for the Duane model can be observed in [Figure 2.6](#) where $\Lambda(t)$ is plotted when $a = 100$ and $b = 0.5$, $b = 1$ and $b = 2$. Note that if $b = 1$, then the failure intensity of the process is constant. Thus, in that case the process is an HPP. On the other hand, if $b < 1$, then $\lambda(t)$ decreases, meaning that the failures tend to occur more infrequently. Therefore, the system shows reliability growth. Finally, if $b > 1$, then the reliability of the system decreases with time. Note that in this case it follows that

$$\lim_{t \rightarrow \infty} \Lambda(t) = +\infty .$$

Therefore, this is an NHPP-infinite model.

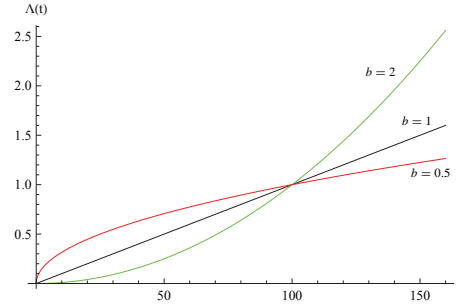


Figure 2.6: Duane model mean-value function for $b = 0.5$, $b = 1$ and $b = 2$.

2.7 Linking GOS and NHPP models

As mentioned in Section 2.5, the main assumption for the class of GOS models is that the times between failures of a software system are defined as the differences between two consecutive order statistics from a random sample of fixed and finite length N . Theorem 2.4 already provides a way of connecting NHPP models with GOS models. In this section we show that if N is considered to follow a Poisson distribution with finite mean, then the counting process $(N(t))_{t \geq 0}$ is an NHPP. In particular, by Lemma 2.5, we will have that

$$\lim_{t \rightarrow \infty} N(t) = N.$$

Let us consider that the first $n \geq 1$ observed failure times of a software system, denoted by $T_1 < T_2 < \dots < T_n$ and defined as in (2.10), are the first n order statistics of a random sample Z_1, \dots, Z_N with common cumulative distribution function $F(t)$, for all $t \geq 0$. Suppose now that N is a Poisson random variable with finite mean $\theta > 0$. Then,

$$\mathbb{P}[N = n_0] = \frac{\theta^{n_0} e^{-\theta}}{n_0!}, \quad (2.41)$$

for all $n_0 = 0, 1, 2, \dots$. For $t \geq 0$ fixed, the random variable $N(t)$ given $N = n_0$ is binomially distributed with parameters n_0 and $F(t)$, as shown in (2.26). Therefore,

$$\begin{aligned} \mathbb{P}[N(t) = n] &= \sum_{n_0=0}^{\infty} \mathbb{P}[N(t) = n \mid N = n_0] \mathbb{P}[N = n_0] \\ &= \sum_{n_0=0}^{\infty} \binom{n_0}{n} (F(t))^n (1 - F(t))^{n_0-n} \frac{\theta^{n_0} e^{-\theta}}{n_0!}. \end{aligned} \quad (2.42)$$

Note that for $n_0 < n$ it follows that

$$\mathbb{P}[N(t) = n \mid N = n_0] = 0. \quad (2.43)$$

Thus, we can write (2.42) as

$$\begin{aligned}\mathbb{P}[N(t) = n] &= \frac{(\theta F(t))^n}{n!} e^{-\theta} \sum_{n_0=n}^{\infty} \frac{(\theta(1-F(t)))^{n_0-n}}{(n_0-n)!} \\ &= \frac{(\theta F(t))^n}{n!} e^{-\theta F(t)},\end{aligned}\quad (2.44)$$

where the last equality comes from the Taylor expansion of $x \mapsto e^x$ in the form

$$\sum_{n_0=n}^{\infty} \frac{(\theta(1-F(t)))^{n_0-n}}{(n_0-n)!} = e^{\theta(1-F(t))} . \quad (2.45)$$

Hence, $N(t)$ follows a Poisson distribution with mean

$$\Lambda(t) = \mathbb{E}[N(t)] = \theta F(t) . \quad (2.46)$$

Thus, the process $(N(t))_{t \geq 0}$ is Poisson with mean-value function $t \mapsto \Lambda(t)$. For further details we refer to [Langberg and Singpurwalla \(1985\)](#). Moreover, by [Theorem 2.5](#), if $N(t)$ is Poisson distributed and such that

$$\lim_{t \rightarrow \infty} \Lambda(t) = \Lambda_0 < \infty , \quad (2.47)$$

then $N(t)$ converges in distribution to a Poisson random variable with mean Λ_0 . In this case, we have that

$$\lim_{t \rightarrow \infty} \Lambda(t) = \lim_{t \rightarrow \infty} \theta F(t) = \theta < \infty . \quad (2.48)$$

Therefore, as $t \rightarrow \infty$, it follows that $N(t)$ converges in distribution to N . Moreover, since N is assumed to follow a Poisson distribution, we can also make predictions about the initial number of faults in the system given that we have already observed n of them. This can be done by computing the conditional probability mass function of N given T_1, \dots, T_n . Note first that the joint density function of T_1, \dots, T_n given $N = n_0$ (see e.g. [Kuo and Yang \(1996\)](#)) is given by

$$L(t_1, \dots, t_n; n_0) = \left(\prod_{i=1}^n f(t_i) \right) \frac{n_0!}{(n_0-n)!} (1-F(t_n))^{n_0-n} . \quad (2.49)$$

Thus, application of Bayes formula, (2.43) and (2.49) yields

$$\begin{aligned}\mathbb{P}[N = n_0 \mid T_1 = t_1, \dots, T_n = t_n] &= \frac{L(t_1, \dots, t_n; n_0) \mathbb{P}[N = n_0]}{\sum_{k=n}^{\infty} L(t_1, \dots, t_n; k) \mathbb{P}[N = k]} \\ &= \frac{\frac{\theta^{n_0} e^{-\theta}}{(n_0-n)!} \left(\prod_{i=1}^n f(t_i) \right) (1-F(t_n))^{n_0-n}}{\sum_{k=n}^{\infty} \frac{\theta^k e^{-\theta}}{(k-n)!} \left(\prod_{i=1}^n f(t_i) \right) (1-F(t_n))^{k-n}} .\end{aligned}\quad (2.50)$$

Note that the factors $e^{-\theta}$ and $\prod_{i=1}^n f(t_i)$ cancel out. Therefore, the conditional probability mass function of N depends only on T_n . We can use (2.45) in the denominator of (2.50) to write

$$\begin{aligned} \mathbb{P}[N = n_0 \mid T_1 = t_1, \dots, T_n = t_n] &= \mathbb{P}[N = n_0 \mid T_n = t_n] \\ &= \frac{\theta^{n_0-n}}{(n_0-n)!} e^{-\theta(1-F(t_n))} (1-F(t_n))^{n_0-n}. \end{aligned} \quad (2.51)$$

In particular, we are interested in the case where the distribution of the order statistics is the Exponential, i.e.,

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

for all $t \geq 0$ and $\lambda > 0$ fixed. In this case, the corresponding GOS model is the Jelinski-Moranda, as we saw in Section 2.5.1. Since N follows a Poisson distribution with mean θ , then by (2.46) we have that

$$\mathbb{E}[N(t)] = \theta(1 - e^{-\lambda t}),$$

which corresponds to the mean-value function of the Goel-Okumoto model, as described in Section 2.6.1. Moreover, we can write the conditional probability mass function of N given $T_1 = t_1, \dots, T_n = t_n$, as defined in (2.51), as follows:

$$\mathbb{P}[N = n_0 \mid T_n = t_n] = \frac{\theta^{n_0-n}}{(n_0-n)!} e^{-(\theta e^{-\lambda t_n} + (n_0-n)\lambda t_n)}. \quad (2.52)$$

The above relationship between the Jelinski-Moranda and Goel-Okumoto models will be used later in chapters 5 and 6.

2.7.1 A note on NHPP-infinite models

When the assumption that the repairing process is not perfect or that new faults can be introduced are considered, it may be possible to experience an infinite number of failures in infinite time. Such a behaviour can be modelled as an NHPP-infinite model (cf. Kuo and Yang (1996)). The Duane model described in Section 2.6.3 and the model presented in Musa and Okumoto (1984) are two examples of NHPP-infinite models. Although NHPP-infinite models cannot be described in terms of General Order Statistics, it is possible to define the stochastic process $(T_n)_{n \geq 0}$ in terms of *Record Value Statistics* (RVS). Suppose that Z_1, Z_2, Z_3, \dots is a sequence of i.i.d. random variables with common distribution function $F(t)$, for all $t \geq 0$. If we define

$$R_1 = \min \{i \mid Z_i > 0\}$$

and

$$R_{k+1} = \min \{i \mid Z_i > Z_{R_k}\},$$

for all $k = 1, 2, \dots$, then the random variables $Z_{R_1} < Z_{R_2} < \dots$ define a sequence of *record values* where Z_{R_i} is called the i^{th} *record value statistic* for all $i \geq 1$. Unlike

order statistics, the sequence of record values can be *infinite* as shown in Glick (1978). In this case, assuming immediate repair and that simultaneous failures do not occur, we can interpret $T_i = Z_{R_i}$ as the i^{th} failure time of an NHPP where

$$\Lambda(t) = -\log(1 - F(t)) ,$$

and

$$\lambda(t) = \frac{f(t)}{1 - F(t)} ,$$

as it is shown in Dwass (1964) and Resnick (2007)[Section 4.1]. Therefore, it follows that $\Lambda(t) \rightarrow \infty$, as $t \rightarrow \infty$. Thus, when the process $(N(t))_{t \geq 0}$ is NHPP-infinite, GOS cannot be used to describe the process $(T_n)_{n \geq 0}$, but this can be described in terms of RVS. Note that in this case N (the initial number of faults in the system) does not play a role like in case of GOS models. In fact, it can be thought as if N would be infinite.

2.8 Bayesian approach

We have seen in Section 2.2.1 that the failure detection process of a software system can be modelled in an equivalent way as one of the following stochastic processes: $(N(t))_{t \geq 0}$, $(T_n)_{n \geq 0}$ and $(X_n)_{n \geq 0}$. Such processes usually depend on some parameter φ that is unknown. For example, we can write

$$\mathbb{P}_\varphi [(X_1, \dots, X_n) \in \mathcal{B}_n] = f_\varphi(\mathcal{B}_n) .$$

Often this parameter is N but it could be any other parameter or a vector of parameters. In the *Bayesian approach* we construct another process such that the parameter φ is considered as a random variable Φ with a given *prior* distribution function $G(\varphi)$. More precisely, we assume that

$$\mathbb{P}_\varphi [(X_1, \dots, X_n) \in \mathcal{B}_n \mid \Phi = \varphi] = f_\varphi(\mathcal{B}_n) ,$$

so that

$$\mathbb{P} [(X_1, \dots, X_n) \in \mathcal{B}_n] = \int f_\varphi(\mathcal{B}_n) dG(\varphi) .$$

We have already seen such a case in Section 2.7 as we explain in more detail now. We have shown that given a GOS process $(N(t))_{t \geq 0}$, if N is considered to be a Poisson random variable, then conditional on $\{N = n_0\}$ the random variable $N(t)$ is binomially distributed and the process $(N(t))_{t \geq 0}$ is also an NHPP. Thus, we can interpret (2.41) as a prior distribution on N . All concepts and properties introduced so far can be generalized in a Bayesian sense. For example, the Markov property can be expressed as follows:

$$\mathbb{P}[X_n \in \mathcal{B} \mid X_1, \dots, X_{n-1}, \Phi] = \mathbb{P}[X_n \in \mathcal{B} \mid X_{n-1}, \Phi] .$$

Therefore, we can consider the *Bayesian* models as another class of software reliability models. They can be classified as semi-Markov, Markov or independent

increments in a Bayesian sense which means that they are semi-Markov, Markov or have independent increments conditionally on Φ . Examples of Bayesian models can be found in [Langberg and Singpurwalla \(1985\)](#) and [Littlewood and Verrall \(1973\)](#). The Bayesian approach is crucial for our certification procedure as we will see in Chapter 5.

2.9 Some other models

In Section 2.4.2 we have presented a classification scheme for software reliability growth models based on some properties of stochastic processes. A sample with some of the most well-known models is shown in Table 2.5. Most of them are Markov models and in particular two large classes like GOS and NHPP models are Markov. We finish this chapter by describing the model introduced in [Schick and Wolverson \(1978\)](#). This model is semi-Markov but not Markov and it is neither GOS nor NHPP.

2.9.1 Schick-Wolverson model

One of the first extensions of the Jelinski-Moranda model is due to [Schick and Wolverson \(1978\)](#). The distribution of the times between failures is derived assuming that the hazard rate after the i^{th} failure detection (unlike for the Jelinski-Moranda model for which it is constant) is a linear function of time, i.e.,

$$h_i(x_i) = (N - i + 1)\lambda x_i, \quad (2.53)$$

for all $i = 1, \dots, N$ and $x_i > 0$. This is depicted in Figure 2.7. The reliability

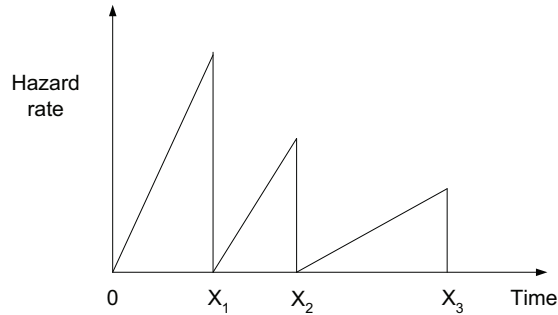


Figure 2.7: Schick-Wolverson model hazard rate. It is a linear function of the time between failure observations.

function of X_i can be computed using (2.9) and it is given by

$$S_i(x_i) = e^{-\frac{(N-i+1)\lambda x_i^2}{2}}, \quad (2.54)$$

which is of the form of a Rayleigh distribution with parameter $((N - i + 1)\lambda)^{-1/2}$. This model is usually characterized by the density function of the times between failures (cf. [Schick and Wolverton \(1978\)](#)) as follows:

$$f_i(x_i) = (N - i + 1)\lambda x_i e^{-\frac{(N-i+1)\lambda x_i^2}{2}}. \quad (2.55)$$

Therefore, since the distribution of the times between failures is not the Exponential, the process $(N(t))_{t \geq 0}$ is not Markov (see Section 2.2.2). Note that the assumptions for this model (cf. [Xie et al. \(2004\)](#)[Chapter 4]) are the same as those for the Jelinski-Moranda with the exception that the hazard rate is a linear function of time over the interval between fault occurrences. Thus, this model is also criticized for considering all faults being of the same type. Like the Jelinski-Moranda model, the distribution of the times between failures depends on N , which is considered to be unknown but *fixed*. Thus, the Schick-Wolverton model is not an NHPP model. Moreover, it is not a GOS model either as we will see now. Suppose that Z_1, \dots, Z_N are N i.i.d. random variables having a common Rayleigh distribution with parameter $\lambda > 0$. Thus,

$$f(z) = \lambda z e^{-\frac{\lambda z^2}{2}},$$

is the density function of Z_i for all $z \geq 0$ and $i = 1, \dots, N$. The distribution of the first order statistic can be computed using Theorem 2.1 and it is given by

$$\begin{aligned} f_{(1)}(z) &= N f(z) (1 - F(z))^{N-1} \\ &= \lambda N z e^{-\frac{\lambda N z^2}{2}}, \end{aligned}$$

which is the same as the distribution of $X_1 = T_1$ as given in (2.55). Thus, the distribution of X_1 in the Schick-Wolverton model corresponds to the first order statistic of a Rayleigh distribution. However, this does not hold for X_2, \dots, X_n . As shown in [Pyke \(1965\)](#)[Section 2.2], the distribution function of $D_i = Z_{(i)} - Z_{(i-1)}$, for all $2 \leq i \leq n$, is given by

$$f_{D_i}(z) = \frac{N!}{(i-2)!(N-i)!} \int_0^\infty (F(x))^{i-2} (1 - F(x+z))^{N-i} f(x) f(x+z) dx. \quad (2.56)$$

In case of the Rayleigh distribution, we can write (2.56) as

$$\begin{aligned} f_{D_i}(z) &= \frac{\lambda^2 N!}{(i-2)!(N-i)!} \int_0^\infty x(x+z) \left(1 - e^{-\frac{\lambda x^2}{2}}\right)^{i-2} \times \\ &\quad \times e^{-\frac{((N-i+1)x^2 + (x+z)^2)\lambda}{2}} dx. \end{aligned} \quad (2.57)$$

It follows that the density function of D_i given in (2.57) is not the same as the density function of X_i given in (2.55), for all $2 \leq i \leq n$. This final step is left to the reader. Take for example $i = 2$ or $i = N$, which simplifies the right-hand side of (2.57), and check that the obtained density is not the same as the one defined by the Schick-Wolverton model. A generalization of the Schick-Wolverton model is studied in [Shanthikumar \(1981\)](#), where the proportionality factor λ is considered to be also a function of time.

CHAPTER 3

STATISTICAL INFERENCE FOR SOFTWARE RELIABILITY GROWTH MODELS

In this chapter we provide a general overview on statistical features used in analysis of software failure data based on reliability growth models. Most of the concepts introduced here are well-known in the software reliability literature. Nevertheless, there are certain issues that are not trivial and require special attention, in particular Maximum Likelihood procedures and convergence issues discussed in Section 3.4. In spite of the rich literature on software reliability (see e.g. monographs like [Lyu \(1996\)](#), [Musa \(2006\)](#), [Musa et al. \(1987\)](#), [Pham \(2006\)](#) and [Xie and Hong \(2001\)](#)) it is not always easy to find correct or sufficiently detailed information on these models. Common problems found in the literature include vague mathematical descriptions, incorrect use of asymptotic results from mathematical statistics, lack of universal agreement on assumptions behind these models and lack of attention for numerical instabilities in parameter estimation algorithms. Most of the techniques presented in this chapter are implemented in a software tool for reliability analyses that will be presented in Chapter 4. Thus, this chapter also provides a statistical background for the features implemented in the tool.

In the line of [Goel \(1985\)](#), we present in the remainder of this chapter a step-by-step procedure to (statistically) analyze software failure data. For us statistical analysis of software failure data should include data description, trend tests, initial model selection, estimation of model parameters, model validation and model interpretation. The main difference with [Goel \(1985\)](#) approach is that we provide more insight on each of the steps and give some examples of application. In particular, the problem of initial model selection is just mentioned as a necessary step in [Goel \(1985\)](#) but no explanation about how this should be done is given. In fact, this problem has not been studied in details in the software reliability literature, being [Kharchenko et al. \(2002\)](#) an exception. We discuss the problem of initial model selection in Section 3.3. Moreover, an important step like the analysis of trend of data (see Section 3.2) is not considered in [Goel \(1985\)](#). The following sections provide a description of these steps.

3.1 Data description

As mentioned in Section 2.4, we assume that the fault detection process of a software system can be modelled as a stochastic counting process $(N(t))_{t \geq 0}$, where $N(t)$ represents the number of failures observed at time $t \geq 0$. For any $n \geq 1$, we can define the failure times T_1, \dots, T_n as in (2.10). According to the Kolmogorov existence theorem introduced in Section 2.2, the process $(N(t))_{t \geq 0}$ can be uniquely characterized by any finite dimensional distribution. Thus, in particular, it may be

specified by the joint distribution of T_1, \dots, T_n . However, failure times are often not observable in practice and, in this case, the process $(N(t))_{t \geq 0}$ may be specified by the joint distribution of $N(\ell_1), \dots, N(\ell_k)$, for any $0 < \ell_1 < \dots < \ell_k$ and $k \geq 1$. Therefore, software failure data may appear in the form of either *point-time* or *interval-time* observations as we will explain now. When failures are reported individually at the *random* times when they occur we speak of point-time observations. In this case, data represents either the failure times $T_1 < T_2 < \dots < T_n$ or the corresponding times between failures $X_1 = T_1, X_2 = T_2 - T_1, \dots, X_n = T_n - T_{n-1}$. Note that in this case n is the total number of observed failures. In practice, the system is often observed up to certain time t such that $T_n < t$ and $N(t) = n$. In this case, we speak of *time truncated* data. Otherwise, when $T_n = t$, we speak of *failure truncated* data. Point-time observations are also called *ungrouped* or *exact* data in the software reliability literature. When failures are reported in intervals of a certain length we speak of interval-time observations. The interval bounds do not correspond to failure times. Therefore, they are not random variables but *fixed* points in \mathbb{R}_+ . Moreover, all the intervals do not have to be necessarily of the same length. In this case, data is collected in pairs consisting of the observation intervals $(\ell_{i-1}, \ell_i]$, for all $i = 1, \dots, k$, where $M_i = N(\ell_i) - N(\ell_{i-1})$ is the *random* number of failures reported in each interval and $M = \sum_{i=1}^k M_i$ is the total number of faults detected at time ℓ_k , with $\ell_0 = 0$ and $N(\ell_0) = 0$. Note that in this case, data is always time truncated. Interval-time observations are also called *grouped* or *interval* data in the software reliability literature. It is highly recommendable to carefully look at the data before starting any kind of statistical analysis. For example, it is possible to gain some understanding about the nature of the process being studied simply by plotting the data as a function of time. Figure 3.1 shows a plot of the failure times against the cumulative number of observed failures. The data set used in Figure 3.1 is in the form of interval-time observations (where the interval length is measured in seconds) and it can be found in Joe (1989). This data set will be used throughout this chapter to illustrate different concepts. Note that a concave plot indicates that software becomes more reliable during testing, due to the fact that faults are repaired, so that more effort is required to find future faults. With this simple step we may detect that failure times follow certain patterns that may reveal some trend associated to a growth or a decrease in reliability.

3.2 Trend analysis

All software reliability models described in Chapter 2 have been developed based on the assumption that the reliability of the system grows as long as faults are found and repaired. Thus, before trying to fit any model to data we should verify whether data indicates reliability growth or not. This can be done by the application of *trend tests*. We distinguish between *graphical* and *statistical* trend tests (see e.g. Ascher and Feingold (1984)). In this section we present some of the most popular (graphical and statistical) trend tests and provide some references for further information.

A simple way to check whether software failure data exhibits a trend is to study

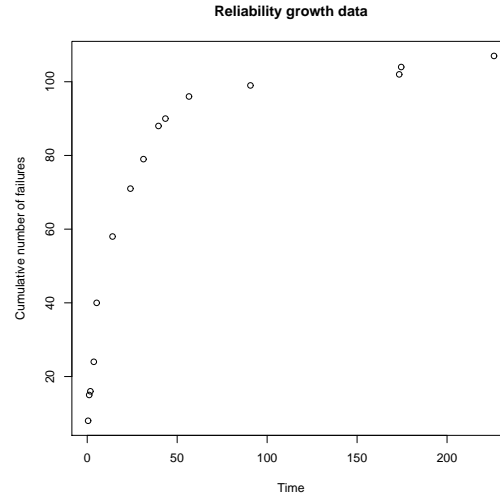


Figure 3.1: Failure times vs. the cumulative number of observed failures. A concave plot indicates reliability growth.

running averages plots like in Figure 3.2. For interval-time observations this plot is produced by computing the running arithmetic averages of the number of failures per interval, i.e., we plot $M_i/(\ell_i - \ell_{i-1})$, for all $i = 1, \dots, n$, where we recall that M_i denotes the number of faults detected in the i^{th} interval, and $\ell_i - \ell_{i-1}$ denotes the length of the i^{th} interval. If the running averages decrease with i , then the number of failures observed per interval also decreases. Therefore, the data shows reliability growth. For point-time observations we compute the running arithmetic averages of the successive failure times, i.e., we plot T_i/i , for all $i = 1, \dots, n$. If the running averages increase with i , then the time between failures also increases. Therefore, the data shows reliability growth. Figure 3.2 shows the running averages plot for the data set in Joe (1989), which is in the form of interval-time observations. Since the running averages decrease, the plot indicates a positive trend.

Another well-known trend plot is the *total time on test* (TTT) plot (see e.g. Klefsjo and Kumar (1992)). Initially developed by Barlow and Campo (1975) for non-repairable systems, TTT plots may be used to decide whether a certain (unknown) distribution exhibits reliability growth. The main idea behind this plot is the following. Suppose that n identical components are tested simultaneously up to a certain time $t > 0$. The random variables T_1, \dots, T_n , with common cumulative distribution function $F(t)$, denote the lifetimes of the n components. Therefore, the order statistics $T_{(1)} \leq T_{(2)} \leq \dots \leq T_{(n)}$ represent the failure times of the components in the order that they occurred (note that this interpretation of the failure times is similar to the one for GOS models described in Section 2.5). Suppose now that i of the components have failed in the interval $(0, t]$. Thus, the total lifetime

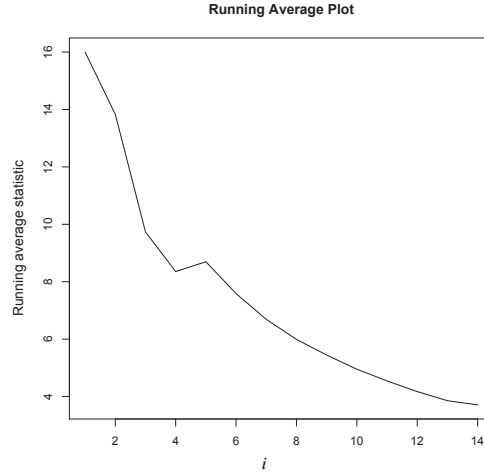


Figure 3.2: Running averages plot for interval-time observations. A decreasing plot indicates reliability growth.

of these i components is given by $\sum_{j=1}^i T_{(j)}$. Since the remaining $n - i$ components have survived the interval $(0, t]$, the total lifetime of these components is given by $(n - i)t$. Therefore, the total time on test at time t , denoted by $\tau(t)$, is defined as the total observed lifetime of the n components at time t , i.e.,

$$\tau(t) = \sum_{j=1}^i T_{(j)} + (n - i)t ,$$

where i is such that $T_{(i)} \leq t \leq T_{(i+1)}$, with the convention that $T_{(0)} = 0$ and $T_{(n+1)} = \infty$. Therefore, the total time on test at the i^{th} failure time is given by

$$\tau(T_{(i)}) = \sum_{j=1}^i T_{(j)} + (n - i)T_{(i)} ,$$

for all $i = 1, \dots, n$. Then, the TTT plot is a plot of the ordered pairs

$$\left(\frac{i}{n}, \frac{\tau(T_{(i)})}{\tau(T_{(n)})} \right) , \quad (3.1)$$

for all $i = 1, \dots, n$. The statistic $\tau(T_{(i)})/\tau(T_{(n)})$ is often called the *scaled TTT statistic*. In Høyland and Rausand (1994)[Section 9.2] and Langberg et al. (1980) it is shown that, for any $0 \leq u \leq 1$, as $n \rightarrow \infty$, it follows that

$$\frac{\tau(T_{(\lceil nu \rceil)})}{\tau(T_{(n)})} \longrightarrow \psi(u) \text{ a.s. ,}$$

where $\lceil \cdot \rceil$ denotes the ceiling function and

$$\psi(u) = \frac{1}{\mu} \int_0^{F^{-1}(u)} (1 - F(t)) dt ,$$

where μ denotes the mean of the distribution $F(t)$. In particular, if the distribution $F(t)$ of the components lifetimes is the Exponential, then $\psi(u) = u$ (cf. Høyland and Rausand (1994)[Example 9.2, p. 362]). Thus, in case of exponential failure times the TTT plot converges to the diagonal of the unit square. A convex plot indicates reliability growth and a concave one indicates reliability decrease. Figure 3.3 shows a TTT plot for the data set in Joe (1989). Since the TTT plot is slightly convex, reliability growth may be expected. For a detailed description and properties of

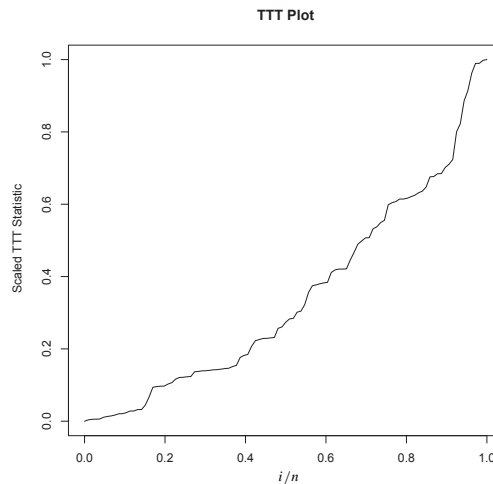


Figure 3.3: Total time on test (TTT) plot for interval-time observations. A convex plot indicates reliability growth.

TTT plots we refer to Høyland and Rausand (1994)[Chapter 9] and Klefsjo and Kumar (1992). The above-described methods for analyzing trend of software failure data, and several other graphical methods, are studied and compared in Kunitz and Pamme (1991).

We now discuss two *statistical* trend tests: the Laplace and the Military Handbook (MIL-HDBK-189) tests (see e.g. Rigdon and Basu (2000)[Section 4.3] for an overview on both tests). The *null hypothesis* of the test is that the fault detection process is an homogeneous Poisson process (HPP) while the alternative depends on the kind of trend (reliability growth or decrease) to be detected. The Laplace test is a conditional statistical test that can be derived as follows. Suppose first that the data is failure truncated, i.e., as explained in Section 3.1, we observe the system up to a certain failure time, say T_n (in contrast to time truncated where we look at an arbitrary fixed time t). Therefore, the failure times of the system are given by

$T_1 < \dots < T_n$. Under the null hypothesis of an HPP, Theorem 2.4 says that, conditional on the event $\{T_n = t_n\}$, it follows that $T_1 < T_2 < \dots < T_{n-1}$ are distributed as $n - 1$ order statistics from a Uniform distribution on the interval $(0, t_n)$. Thus, the random variable $S = \sum_{i=1}^{n-1} T_i$ has mean $(n - 1)t_n/2$ and variance $(n - 1)t_n^2/12$. Therefore, if $\mathcal{N}(0, 1)$ denotes the standard normal distribution, then under the null hypothesis of HPP and by the Central Limit Theorem it follows that

$$U_L = \frac{\sum_{i=1}^{n-1} T_i - (n - 1)t_n/2}{\sqrt{(n - 1)t_n^2/12}} \longrightarrow Z \sim \mathcal{N}(0, 1) .$$

The null hypothesis of an HPP process for the Laplace trend test will be rejected if $U_L < z_{\alpha/2}$ or $U_L > z_{1-\alpha/2}$, where z_γ denotes the γ -percentile of the standard normal distribution. Note that for small values of U_L the null hypothesis is rejected in favor of reliability growth (failure times are small, therefore, faults occur in the beginning of the interval $(0, t_n)$). Suppose now that the data is time truncated, i.e., as explained in Section 3.1, we observe the system up to an arbitrary fixed time t with no fault such that $T_1 < \dots < T_n < t$ and $N(t) = n$. In this case, under the null hypothesis of an HPP and conditional on the event $\{N(t) = n\}$, it follows that $T_1 < T_2 < \dots < T_n$ are distributed as n order statistics from a Uniform distribution on the interval $(0, t)$ (cf. Rigdon and Basu (2000)[Theorem 22]). Thus, the random variable $S = \sum_{i=1}^n T_i$ has mean $nt/2$ and variance $nt^2/12$. Therefore, by the Central Limit Theorem it follows that

$$U_L = \frac{\sum_{i=1}^n T_i - nt/2}{\sqrt{nt^2/12}} \longrightarrow Z \sim \mathcal{N}(0, 1) .$$

The interpretation of the test statistic is the following: for small values of the test statistic the null hypothesis of HPP is rejected in favor of reliability growth. The Laplace test for interval-time observations is studied in Kanoun et al. (1991). In this case, the Laplace test is developed under the restriction that *all test intervals must have the same length*. Thus, the observation intervals are given by $(\ell_{i-1}, \ell_i] = ((i - 1)\ell, i\ell]$, for all $i = 1, \dots, k$, where $M_i = N(i\ell) - N((i - 1)\ell)$ is the random number of failures reported in each interval and $M = \sum_{i=1}^k M_i$ is the total number of faults detected at time $k\ell$. In this case, under the null hypothesis of an HPP and conditional on the event $\{\sum_{i=1}^n M_i = m\}$ it follows that

$$U_L = \frac{\sum_{i=1}^k (i - 1)M_i - m(k - 1)/2}{\sqrt{m(k^2 - 1)/12}} \longrightarrow Z \sim \mathcal{N}(0, 1) .$$

The interpretation of the test is the same as in the point-time case: for small values of the test statistic, the null hypothesis is rejected in favor of reliability growth. For

details on the derivation of the Laplace test statistic in this case, we refer to [Kanoun et al. \(1991\)](#)[Appendix A].

The MIL-HDBK-189 trend test is also a conditional statistical test but based on the power-law process (see Section 2.6.3). The rationale behind this test is the following. The intensity function of the power-law process is given by

$$\lambda(t) = \frac{b}{a} \left(\frac{t}{a} \right)^{b-1},$$

where $a > 0$ and $b > 0$ are the parameters of the model, for all $t \geq 0$. Note that if $b < 1$, then $\lambda(t)$ decreases, meaning that the failures tend to occur less frequently. Therefore, the system shows reliability growth. If $b > 1$, then the system shows reliability decrease. Finally, if $b = 1$, then the process is an HPP. Conditional on the event $\{T_n = t_n\}$, the Maximum Likelihood (ML) estimator of b of a failure truncated power-law process (see e.g. [Rigdon and Basu \(2000\)](#)[p. 118]) is given by

$$\hat{b} = \frac{n}{\sum_{i=1}^{n-1} \log(t_n/T_i)}.$$

Thus, if $\chi^2(n)$ denotes the chi-squared distribution with n degrees of freedom, then under the null hypothesis of $b = 1$ (cf. [Military Handbook](#)[p. 68]) it follows that

$$2n/\hat{b} \sim \chi^2(2(n-1)).$$

Unlike the Laplace test, the distribution of the MIL-HDBK-189 test statistic is exact and not asymptotic. If the alternative hypothesis is two-sided, then the null hypothesis is rejected if

$$\hat{b} > \frac{2n}{\chi_{1-\alpha/2}^2(2(n-1))}$$

or

$$\hat{b} < \frac{2n}{\chi_{\alpha/2}^2(2(n-1))},$$

where $\chi_{\gamma}^2(\nu)$ denotes the γ -percentile of the chi-squared distribution with ν degrees of freedom. For large values of \hat{b} the null hypothesis is rejected in favor of reliability growth. For a time truncated power-law process and conditional on the event $\{N(t) = n\}$, the ML estimator of b (see e.g. [Rigdon and Basu \(2000\)](#)[p. 137]) is given by

$$\hat{b} = \frac{n}{\sum_{i=1}^n \log(t/T_i)}.$$

Also in this case, under the null hypothesis of $b = 1$ (cf. [Military Handbook](#)[p. 68]) it follows that

$$2n/\hat{b} \sim \chi^2(2n).$$

The interpretation of the test statistic is the same as for the failure truncated case: for large values of \hat{b} , the null hypothesis is rejected in favor of reliability growth. Unlike the Laplace trend test, for interval-time observations the MIL-HDBK-189 test does not require equal length of test intervals. Thus, the observation intervals are given by $(\ell_{i-1}, \ell_i]$, for all $i = 1, \dots, k$, where $d_i = \ell_i - \ell_{i-1}$, $M_i = N(\ell_i) - N(\ell_{i-1})$ and $M = \sum_{i=1}^k M_i$. Nevertheless, the condition $Md_i/l_k > 5$ must be satisfied for all $i = 1, \dots, k$ (see [Military Handbook](#)[p. 69]). In this case, under the null hypothesis of an HPP and conditional on the event $\{\sum_{i=1}^n M_i = m\}$ it follows that, as $k \rightarrow \infty$,

$$\sum_{i=1}^k \frac{(M_i - md_i/l_k)^2}{md_i/l_k} \longrightarrow W \sim \chi^2(k-1).$$

Note that the distribution of the test statistic is asymptotic for interval-time observations (cf. [Military Handbook](#)[p. 69]). The interpretation of the test statistic is the following: for large values of the test statistic the null hypothesis of $b = 1$ (HPP) is rejected in favor of reliability growth.

For a comparison between these and some other trend tests (for which the null hypothesis is that the process is HPP against the alternative of reliability growth) we refer to [Bain et al. \(1985\)](#), [Cohen and Sackrowitz \(1993\)](#) and [Wang and Coit \(2005\)](#). Variations on these tests can be found in [Kvaløy and Linqvist \(1998\)](#). Similar tests for interval data seem not to have been studied in the software reliability literature (cf. [Weller and Ryan \(1998\)](#)).

3.3 Model type selection

The main problem that we find when trying to select a suitable model for a specific problem is that there are no general rules to select a model. This is stressed by the abundance of software reliability models since there are over 200 known models according to [Singpurwalla and Wilson \(1994\)](#). Although it is possible to find a large variety of lists of assumptions and data requirements for software reliability models in the literature (see e.g. [Goel \(1985\)](#), [Lyu \(1996\)](#)[Chapter 3], [Musa and Okumoto \(1984\)](#), [Ohba \(1984\)](#), [Pham \(2006\)](#)[Chapter 6] and [Xie et al. \(2004\)](#)[Chapter 4]), this is often far from facilitating model selection. For example, there is no universal agreement in the literature on the list of assumptions for certain well-known models. Moreover, some assumptions or data requirements are not valid. An example of erroneous data requirements can be found in existing tools for software reliability analyses like Casre and Smerfs³ where it is mentioned that certain GOS or NHPP models work only for interval-time observations or only for point-time observations. This is false since likelihood equations exist for all GOS and NHPP models for both types of data as we will see in Section 3.4.

Systematic approaches to use model assumptions and data requirements for initial model selection are hard to find in the software reliability literature. An exception to this is [Kharchenko et al. \(2002\)](#). They propose a layer structure for model assumptions depending on the effect of the assumption on specific models. On the

first layer they consider *assumptions that apply to all models without exception*. We consider this statement too strong in general. For example, some of the assumptions that they consider in the first layer are the following:

- (1) software is tested under operational conditions,
- (2) immediate repair,
- (3) new faults are not introduced during reparation.

For us, only the first one should be considered as *universal*. For the same reason, universal assumptions should not be used in model selection because they do not discriminate between models at all. For the other two assumptions listed above we already provided some discussion in Section 2.3.2. For example, immediate repair can be modelled as an alternating renewal process (cf. Ansell and Phillips (1994)[Section 5.4.4]) and for NHPP-infinite models it is not possible to distinguish between original faults and faults introduced during reparation. Therefore, we find it more convenient to consider a first layer of assumptions that will determine the *type of stochastic process* to be used. For example, assumption (2) above is valid for counting processes, as defined in Section 2.2. However, for us assumption (3) should be in a lower layer since, for example, it is valid for GOS models but not for NHPP-infinite models. In correspondence with the classification scheme described in Section 2.4.2, we propose the following three main layers of assumptions:

- (i) *Type* layer: modelling phenomena to random variables. These assumptions will determine the stochastic process to be used (counting process, alternating renewal process (see e.g. Ansell and Phillips (1994)[Section 5.4.4]), superimposed process (see e.g. Thompson (1988)[Chapter 7]), etc.).
- (ii) *Class* layer: properties of the probability distributions of the random variables chosen in the layer “Type”. These assumptions will determine whether the stochastic process is semi-Markov, Markov, has independent increments, etc.
- (iii) *Functional form* layer: parameters of the probability distributions chosen in the layer “Class”. These assumptions will determine a distribution function, a mean-value function, an intensity function, etc.

To support the choice of suitable models we also adopt a heuristic matrix-based procedure proposed by Refis (www.refis.nl). A simple version of this matrix, with a selection of some of the most well-known models and basic assumptions, can be found in Table 3.1. The selection matrix is a soft tool providing help to practitioners that do not know the assumptions behind existing models. To select models, one first has to select the assumptions which are relevant to the testing project at hand. Each assumption is related to each model with a weight defining the relative importance of the assumption for the model. In case of the selection matrix the weights vary from 1 (lowest relative importance) to 3 (highest relative importance). After selecting all the relevant assumptions every model receives a final score. If all requirements and selected assumptions of a model are satisfied,

Assumptions	Weight	Duane (1964)	Jelinski and Moranda (1972)	Goel and Okumoto (1978)	Schick and Wolverton (1978)	Moranda (1979)	Goel and Okumoto (1979)	Musa (1979)	Shanthikumar (1981)	Musa and Okumoto (1984)	Yamada et al. (1984)	Goel (1985)	Abdel-Ghaly et al. (1986)	Xie (1990)	Xie and Zhao (1993)
(1) $(N(t))_{t \geq 0}$ has ind. incr.	3	x		x						x	x				x
(2) $(N(t))_{t \geq 0}$ not necessarily Markov	3				x				x						
(3) $(T_n)_{n \geq 0}$ has ind. incr.	3		x		x	x	x	x	x			x	x	x	
(4) Imperfect repair	2	x								x		x			
(5) Infinite number of failures	2	x								x					
(6) Fixed number of faults	3		x		x				x			x	x		

Table 3.1: Selection matrix. An “x” indicates whether an assumption is allowed for the corresponding model.

then the score for this model is 100%. In all other cases the score of the model is defined using the relative importance (weight) of the applicable characteristics of the model. Thus, the higher score a model has, the better the model fulfills the assumptions. Consequently, as initial model choice we may use the models with higher scores. Note that the weights in the selection matrix are *subjective* choices and further research is needed to obtain objective criteria. For example, if we choose assumptions (3), (4) and (6) in Table 3.1, then the model given by Goel (1985) gets a score of 100%. For all the other models their final score depends on the number of selected assumptions (and their relative importance) for each model. Further details can be found in Boon et al. (2007), Brandt et al. (2007a) and Brandt et al. (2007b). This approach has been implemented in a new software reliability tool that is described in Chapter 4.

3.4 Model estimation

As mentioned in Chapter 2, we can assume that the failure detection process of a software system can be described as a counting process $(N(t))_{t \geq 0}$. Software reliability growth models arise when different functional forms for $(N(t))_{t \geq 0}$ are considered. These models are characterized by certain *parametric* functions like the mean-value function of the process or the probability distribution of the failure times. Thus, after an initial selection of possible candidate models has been accomplished, we can use observed failure data to estimate the parameters of the models in order to predict the future behaviour of the system. The Maximum Likelihood (ML) procedure is the preferred point estimation procedure in statistics. ML estimators possess optimal asymptotic properties (asymptotically minimal variance unbiased

estimators). However, as we will explain now, asymptotic properties of ML estimators must be studied with *extra care* in the context of software reliability. In general, asymptotic properties of the ML estimators are obtained under certain *regularity conditions*. These conditions require certain differentiability properties of the density function of the random observations and that the *parameter space is open*. They can be found, for example, in Serfling (1980)[Section 4.2] and van der Vaart (2000)[Section 5.6]. The assumption of open parameter space is usually broken here. For example, for GOS models the ML estimator of the initial number of faults in the system is often in the boundary of the parameter space (see e.g. Joe and Reid (1985)). A generalization of standard asymptotic results for NHPP and GOS models is studied in Joe (1989). Asymptotics in Joe (1989) are defined for time fixed and letting the number of faults in GOS models, and the number of expected faults in NHPP models, tend to infinity. Regularity conditions in this case are defined for the density function of the order statistics in GOS models and for a conditional density function (in a similar way as it is done in Theorem 2.4) for NHPP models. Similar results in a general setting for counting processes can be found in van Pul (1993)[Section 3.3]. In particular, results for the Jelinski-Moranda (cf. Jelinski and Moranda (1972)) and Littlewood (cf. Littlewood (1980)) models are worked out in details.

As far as confidence intervals computation is concerned, similar remarks as those mentioned for ML estimation apply here. Confidence intervals can be derived from asymptotic normality of the ML estimators. However, as shown in Joe (1989), convergence is slow and frequently the ML estimators take their values in the boundary of the parameter space. For example, for GOS models the normal approximation does not work very well since the log-likelihood is a skewed function of N and often \hat{N} is in the boundary of the parameter space, i.e., $\hat{N} = n$ or $\hat{N} = \infty$. Moreover, the true value of N must be sufficiently large. As shown in Joe (1989) and van Pul (1993), these problems can be solved by using an alternative procedure to derive confidence intervals based on the *Wilks likelihood ratio* statistic. For a definition and properties of the Wilks statistic we refer to Deshpande and Purohit (2005)[Section 4.2, p. 53]. Although results based on the Wilks statistic are shown to perform better, asymptotic normality can also be used to derive confidence intervals (usually in terms of the *Fisher information* matrix). However, this is often done in an imprecise way, as in Xie and Hong (2001), without looking at regularity conditions.

In general, ML estimators may not be unique or may not exist. When they do exist, their computation often requires *numerical optimization*. Since the parameters of the models are usually of a different order of magnitude, numerical problems like non-convergence or large flat areas around the maximum may appear (see Yin and Trivedi (1999)). Figure 3.4 illustrates this problem with a data set from Currit et al. (1986) consisting of 25 observed times between failures. This figure shows the log-likelihood function for the given data set for the Goel-Okumoto model (cf. Goel and Okumoto (1978)). As mentioned in Section 2.6.1, the Goel-Okumoto model is an NHPP model with two parameters, denoted by a and b , which are of a different order of magnitude: a represents the total expected number of faults to be experienced in infinite time and b represents the rate at which failures are observed.

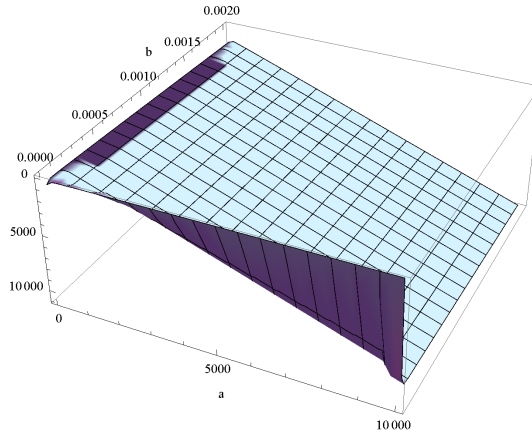


Figure 3.4: Log-likelihood function for a data set from [Currit et al. \(1986\)](#) assuming the Goel-Okumoto model. As b tends to zero, the ML estimator of a goes to infinity.

It is clear from Figure 3.4 that as b tends to zero, the value of a that maximizes the log-likelihood function goes to infinity. For that reason, direct maximization of the likelihood function may cause numerical problems like slow convergence or non-convergence at all. Thus, in order to develop efficient estimation procedures, it is important to pay attention to such convergence issues and apply algorithms that avoid these standard numerical problems. Next we discuss some properties and related problems concerning ML estimation for GOS and NHPP models.

3.4.1 ML estimation for GOS models

As described in Section 2.5, the main assumption for the class of GOS models is that the failure times can be interpreted as the order statistics of a certain random sample of fixed length N (the unknown total number of faults in the system) with common distribution function $F_{(N,\theta)}(t)$, for all $t \geq 0$, where θ denote some other possible parameters of the distribution. In case of point-time observations, suppose that, for any $n \geq 1$, the failure times are given by T_1, \dots, T_n . Thus, n is the number of failures observed at time T_n . Then the likelihood function of T_1, \dots, T_n (see e.g. [Kuo and Yang \(1996\)](#)) is given by

$$L(t_1, \dots, t_n; N, \theta) = \left(\prod_{i=1}^n f_{(N,\theta)}(t_i) \right) \frac{N!}{(N-n)!} (1 - F_{(N,\theta)}(t_n))^{N-n}. \quad (3.2)$$

In case of interval-time observations, suppose that the observation intervals are given by $(\ell_{i-1}, \ell_i]$, for all $i = 1, \dots, k$, where $M_i = N(\ell_i) - N(\ell_{i-1})$ and $M = \sum_{i=1}^k M_i$.

Then the likelihood function of M_1, \dots, M_k (see e.g. [Joe \(1989\)](#)) is given by

$$L(m_1, \dots, m_k; N, \theta) = \left(\prod_{i=1}^k \frac{(F_{(N,\theta)}(\ell_i) - F_{(N,\theta)}(\ell_{i-1}))^{m_i}}{m_i!} \right) \times \quad (3.3)$$

$$\times \frac{N!}{(N-m)!} (1 - F_{(N,\theta)}(\ell_k))^{N-m}.$$

Since likelihood functions exist for both point-time and interval-time observations, ML estimators of N and θ , denoted by \hat{N} and $\hat{\theta}$, respectively, can be computed in principle in both cases. Next we describe a concrete example of ML estimation corresponding to the Jelinski-Moranda model.

Jelinski-Moranda model

As mentioned in Section 2.5.1, the distribution of the order statistics for the Jelinski-Moranda model is the Exponential distribution. If λ denote the parameter of the Exponential distribution, then the likelihood function of T_1, \dots, T_n as defined in (3.2) is given by

$$L(t_1, \dots, t_n; N, \lambda) = \left(\prod_{i=1}^n \lambda e^{-\lambda t_i} \right) \frac{N!}{(N-n)!} (1 - e^{-\lambda t_n})^{N-n}. \quad (3.4)$$

ML estimators of N and λ can be computing by solving

$$\frac{\partial L(t_1, \dots, t_n; N, \lambda)}{\partial N} = 0,$$

and

$$\frac{\partial L(t_1, \dots, t_n; N, \lambda)}{\partial \lambda} = 0.$$

We will refer to these equations as the *ML equations*. In this case, if $x_i = t_i - t_{i-1}$, for all $i = 1, \dots, n$, then the ML estimators of λ and N (see e.g. [Xie et al. \(2004\)](#)[p. 74]) are the solutions of

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^n (\hat{N} - i + 1) x_i}, \quad (3.5)$$

and

$$\sum_{i=1}^n \frac{1}{\hat{N} - i + 1} - \frac{n \sum_{i=1}^n x_i}{\sum_{i=1}^n (\hat{N} - i + 1) x_i} = 0, \quad (3.6)$$

respectively. Whereas $\hat{\lambda}$ is given in an explicit form as a function of \hat{N} , the computation of \hat{N} requires that (3.6) has to be solved numerically. Existence criteria for \hat{N} and $\hat{\lambda}$ can be found in [Finkelstein et al. \(1999\)](#), [Moek \(1984\)](#) and [Osborne](#)

and Severini (2000). Non-existence or degeneracy in this case is related to software deterioration as shown in Littlewood and Verrall (1981). Improving on an earlier attempt in Joe and Reid (1985), a stable algorithm for computing \hat{N} can be found in Finkelstein et al. (1999). An approximate way to compute exact confidence intervals can also be found in Finkelstein et al. (1999). Unfortunately, these results are *only for point-time observations*. As mentioned in the introduction of this section, Joe (1989) and van Pul (1992b) contain correct asymptotics for $N \rightarrow \infty$ and $T_n \rightarrow \infty$, respectively. They show that asymptotic confidence intervals based on the Wilks likelihood ratio statistic are preferred to those obtained from asymptotic normality. Note that the likelihood function for interval-time observations exists and it is given by (3.3). Therefore, similar expressions to (3.5) and (3.6) can be obtained for $\hat{\lambda}$ and \hat{N} when data is of the form of interval-time observations. In this case, the results in Joe (1989) are also valid, however no stable algorithm for computing estimators (nor confidence intervals) is known.

3.4.2 ML estimation for NHPP models

As described in Section 2.6, the main assumption for the class of NHPP models is that the failure times can be interpreted as the event times of a Poisson process $(N(t))_{t \geq 0}$ where $N(t)$ is the number of failures observed at time t . NHPP models are usually described in terms of its mean-value function $\Lambda_\theta(t) = \mathbb{E}_\theta[N(t)]$, for all $t \geq 0$, where θ denote the parameters of the model. Unlike GOS models, the unknown total number of faults in the system N is considered to be a Poisson random variable. In case of point-time observations, the likelihood function of the failure times T_1, \dots, T_n (see e.g. Thompson (1988)[p. 56]) is given by

$$L(t_1, \dots, t_n; \theta) = \left(\prod_{i=1}^n \lambda_\theta(t_i) \right) e^{-\Lambda_\theta(t_n)}. \quad (3.7)$$

In case of interval-time observations, the likelihood function of M_1, \dots, M_k (see e.g. Xie and Hong (2001)[p. 710]) is given by

$$L(m_1, \dots, m_k; \theta) = \left(\prod_{i=1}^k \frac{(\Lambda_\theta(\ell_i) - \Lambda_\theta(\ell_{i-1}))^{m_i}}{m_i!} \right) e^{-\Lambda_\theta(\ell_k)}. \quad (3.8)$$

Since likelihood functions exist for both point-time and interval-time observations, the ML estimator of θ , denoted by $\hat{\theta}$, can be computed in principle in both cases. Next we describe two concrete examples of ML estimation corresponding to the Goel-Okumoto and Duane (power-law) models.

Goel-Okumoto model

As mentioned in Section 2.6.1, the Goel-Okumoto model can be described by the mean value function $\Lambda(t) = a(1 - e^{-bt})$, for all $t \geq 0$, where $a > 0$ and $b > 0$ are the parameters of the model. In this case, the likelihood function of T_1, \dots, T_n as

defined in (3.7) is given by

$$L(t_1, \dots, t_n; a, b) = \left(\prod_{i=1}^n abe^{-bt_i} \right) e^{-a(1-e^{-bt_n})}. \quad (3.9)$$

ML estimators of a and b can be computed by solving the corresponding ML equations. In this case, \hat{a} and \hat{b} (see e.g. Xie et al. (2004)[p. 103]) are the solutions of

$$\hat{a} = \frac{n}{1 - e^{-\hat{b}t_n}}, \quad (3.10)$$

and

$$\frac{n}{\hat{b}} - \frac{nt_n e^{-\hat{b}t_n}}{1 - e^{-\hat{b}t_n}} - \sum_{i=1}^n t_i = 0, \quad (3.11)$$

respectively. Note that \hat{a} is given in an explicit form as a function of \hat{b} . However, the computation of \hat{b} requires that (3.11) has to be solved numerically. Existence conditions for the ML estimators for point-time observations can be found in Hossain and Dahiya (1993) and Knafel and Morgan (1996). The method used in Knafel and Morgan (1996) to obtain ML estimators consists of solving the ML equations in terms of a single equation in one unknown. Usually \hat{a} can be expressed in terms of \hat{b} . Therefore, it is just necessary to solve the ML equation for \hat{b} , where the value of \hat{a} has been substituted in. Existence and uniqueness conditions for the solution to this equation is given in terms of a simple monotonicity criterion. Note also that the likelihood function for interval-time observations exists and it is given by (3.8). Thus, similar expressions to (3.10) and (3.11) can be obtained for \hat{a} and \hat{b} when data is of the form of interval-time observations. In this case, existence criteria for \hat{a} and \hat{b} , and a stable algorithm to compute \hat{b} , can be found in Hossain and Dahiya (1993) and Knafel (1992). Joe (1989) provides a correct theoretical way of obtaining asymptotic confidence intervals for \hat{a} and \hat{b} based on the Wilks likelihood ratio statistic and on asymptotic normality for both point-time and interval-time observations. Confidence intervals based on the Wilks statistic are preferred as explained in the introduction of this section. Unfortunately, no stable algorithms for computing confidence intervals based on the Wilks statistic are known.

Duane (power-law) model

As introduced in Section 2.6.3, the Duane model can be described by the mean value function $\Lambda(t) = (t/a)^b$, for all $t \geq 0$, where $a > 0$ and $b > 0$ are the parameters of the model. In this case, the likelihood function of T_1, \dots, T_n as defined in (3.7) is given by

$$L(t_1, \dots, t_n; a, b) = \left(\prod_{i=1}^n \frac{b}{a} \left(\frac{t_i}{a} \right)^{b-1} \right) e^{-(t_n/a)^b}. \quad (3.12)$$

ML estimators of a and b can be computed by solving the corresponding ML equations. In this case both \hat{a} and \hat{b} have explicit form (cf. Duane (1964)) and they are

given by

$$\hat{a} = \frac{t_n}{n^{1/\hat{b}}}, \quad (3.13)$$

and

$$\hat{b} = \frac{n}{\sum_{i=1}^{n-1} \log(t_n/t_i)}, \quad (3.14)$$

respectively. Duane (1964) and Gaudoin et al. (2006) provide algorithms to compute exact confidence intervals for \hat{b} and asymptotic confidence intervals for \hat{a} , respectively. However, these results are only valid for point-time observations. Note that the likelihood function for interval-time observations exists and it is given by (3.8). Thus, similar expressions to (3.13) and (3.14) can be obtained for \hat{a} and \hat{b} when data is of the form of interval-time observations. In this case, existence criteria for \hat{a} and \hat{b} , and a stable algorithm to compute them, can be found in Knafl (1992). Unfortunately, only algorithms based on asymptotic normality are known for computing confidence intervals.

3.5 Model validation

In Section 3.3 we have presented a heuristic procedure to select software reliability growth models based on the assumptions which are relevant to our testing project. After the initial model selection has been done, we use observed failure data to estimate the parameters of the models in order to predict future behaviour of the system. However, before computing any predicted quantity of interest we should determine to what extent a selected *model fits the data*. This can be done by the application of *goodness-of-fit tests*. We present in this section some of the most popular *graphical* and *statistical* goodness-of-fit tests and provide some references for further information.

A simple way to check initial adequacy of the models chosen beforehand is by plotting the *fitted models* (obtained by substituting the estimated values of the parameters) together with data like in Figure 3.5. The closer the data points lie to the fitted model, the better the model explains the data. Figure 3.5 shows a fitted model based on the data set in Joe (1989) and the specific choice of the Jelinski-Moranda model. As mentioned in Section 2.5.1, the Jelinski-Moranda model is a GOS model with two parameters, denoted by N and λ , where N represents the total number of faults in the system and λ the rate at which failures are observed. The fitted model has been obtained based on the estimates $\hat{N} = 107$ and $\hat{\lambda} = 0.035$. According to Figure 3.5, the Jelinski-Moranda model may seem to be adequate for the chosen data set, however this is a *subjective* appreciation. For objective criteria we need to validate the model via statistical goodness-of-fit tests.

For certain types of models, graphical methods like the *u-plot* (see e.g. Lyu (1996)[Section 4.3.3]) or the TTT plot (see Section 3.2) can be used to assess goodness-of-fit. The u-plot is based on the property known as *probability integral transform*. This property states that if X is a continuous random variable with

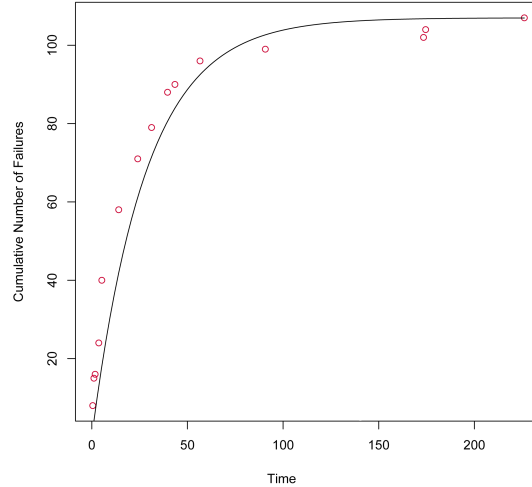


Figure 3.5: Fitted Jelinski-Moranda model (solid line) for interval-time observations (circles).

absolutely continuous cumulative distribution function $F(x)$, then the random variable $Y = F(X)$ is uniformly distributed on $(0, 1)$ (see e.g. [DeGroot \(1986\)](#)). As described in Section 2.5, the failure times of GOS models, denoted by T_1, \dots, T_n , are interpreted as the first n order statistics of N (unknown but fixed) i.i.d. random variables with a common distribution function $F(t)$. Thus, the probability integral transform can be directly applied to GOS models, where the transformed data is $F(T_1), \dots, F(T_n)$. For NHPP models this cannot be applied directly since, as we explained in Section 2.6, N is considered as a Poisson random variable. However, it is also possible to define the u-plot for NHPP models by using Theorem 2.4. Thus, the transformed data is given by $\Lambda(T_1)/\Lambda(t), \Lambda(T_2)/\Lambda(t), \dots, \Lambda(T_n)/\Lambda(t)$, where $T_1 < \dots < T_n < t$. In both cases (GOS and NHPP), when the plot of the transformed data lies close to the diagonal unit square, the selected model may correctly describe the failure process of the system.

The TTT plot introduced in Section 3.2 can also be used to assess goodness-of-fit but *only for the power-law model*, as shown in [Klefsjo and Kumar \(1992\)](#). Suppose that $T_1 < \dots < T_n$ are the first n failure times of a certain process that is described by the power-law model with parameters a and b . If we consider the transformed data

$$w_i = -\log\left(\frac{T_{n-i}}{T_n}\right),$$

for all $i = 1, \dots, n-1$, then w_i are distributed as the first $n-1$ order statistics from an Exponential distribution with parameter b (cf. [Rigdon and Basu \(2000\)](#)[p. 97]). Thus, if we consider the TTT plot in (3.1) with w_i replacing $T_{(i)}$, then we have

a graphical method to assess goodness-of-fit for the power-law process. When the TTT plot lies close to the diagonal unit square, the power-law process may correctly describe the failure process of the system.

We now introduce some statistical tests for goodness-of-fit for both GOS and NHPP models here. In case of GOS models, the failure times are considered as the order statistics from a random sample of size N . Thus, the application of the well-known Kolmogorov goodness-of-fit test (see e.g. Sheskin (2004)[Chapter 7]) is possible. In case of NHPP models, failures times do not admit the order statistics interpretation. Moreover, they are not independent random variables in general. For that reason, most of the goodness-of-fit tests for NHPP models found in the literature are conditional tests based on Theorem 2.4. Some of them are studied in Rigdon and Basu (2000)[Section 4.6]. Moreover, like in the case of u-plots, an additional step to transform the original observations is usually needed. For diverse subclasses of NHPP models, new unconditional goodness-of-fit tests are becoming available. In Zhao and Wang (2005) a goodness-of-fit test for a subclass of NHPP models with intensity function of the form $\lambda(t) = a\tilde{\lambda}(t, b)$ is developed. According to that paper, most of the known NHPP models have an intensity function of this form. For example, this subclass includes the Goel-Okumoto (see Section 2.6.1), Yamada S-shaped (see Section 2.6.2) and Duane (power-law) (see Section 2.6.3) models.

If the result of the application of goodness-of-fit techniques is satisfactory for certain models, then we may assume that the failure detection process of the system can be properly described by such models. We would like to emphasize that a positive goodness-of-fit does not necessarily implies that the model *predicts* well since the concept of goodness-of-fit concerns the *past* behaviour of the system whereas prediction concerns the *future performance* of it. We can compare predictive performance of different models using *prequential likelihoods*. This is a general approach in statistics introduced in Dawid (1984). Its main purpose is to make predictions in a sequential fashion based on previous history. We now introduce some concepts in order to briefly explain this approach. Suppose that $\mathcal{G} = \{F_\theta \mid \theta \in \Theta\}$ is a parametric family of probability distributions for the sequence of random variables X_1, X_2, \dots (for example, a sequence of times between failures of a software system). Any distribution $F \in \mathcal{G}$ determines a *statistical forecasting system* for \mathcal{G} . Thus, for any $n \geq 1$ fixed, the goal of the prequential approach is to specify a *prequential probability distribution* of X_{n+1} given X_1, \dots, X_n . Suppose now that $f_{X_{n+1}|X_1, \dots, X_n}(x)$ and $g_{X_{n+1}|X_1, \dots, X_n}(x)$ are two *prequential density functions* for X_{n+1} . We can compare them for a realization of the random sequence X_1, \dots, X_n , denoted by x_1, \dots, x_n , using the *prequential likelihood ratio* as follows:

$$L_{p,n}(x_1, \dots, x_n) = \prod_{i=1}^n \frac{g_{X_i|X_1, \dots, X_{i-1}}(x_i)}{f_{X_i|X_1, \dots, X_{i-1}}(x_i)}.$$

According to Dawid (1984)[Section 7], we can conclude the following:

- (1) If $\lim_{n \rightarrow \infty} L_{p,n}(x_1, \dots, x_n) = \infty$, then we may assume that the probability distribution defined by g will predict the future behaviour of X_{n+1} better than the probability distribution defined by f .

- (2) If $\lim_{n \rightarrow \infty} L_{p,n}(x_1, \dots, x_n) = 0$, then f is preferred to g , in the sense described in (1).
- (3) If $0 < \lim_{n \rightarrow \infty} L_{p,n}(x_1, \dots, x_n) < \infty$, then we cannot assume that one distribution will predict the future behaviour of X_{n+1} better than the other one. In this case, it is said that f and g are equivalent from a predictive point of view.

The prequential approach has already been used in a software reliability context in [Littlewood and Mayne \(1989\)](#). The results obtained there support the use of prequential techniques to judge and compare the predictive performance of different software reliability growth models.

In case of satisfactory results in both model fitting and prediction, we can move one step forward. Otherwise, additional data collection or better model choice are required.

3.6 Model interpretation

The main reason for developing software reliability models is to use them to support some decisions to be made about the software system, for example to release it or continue testing. Based on the information collected during testing, we can estimate quantities like the number of remaining faults, the future reliability of the system or the time to next failure. All these quantities can be used to support the decision to release a software system after a certain period of testing. For example, in [Currit et al. \(1986\)](#) the decision to release the software is based on a targeted value of the estimated mean time to failure of the system. An example showing how to interpret the results of the analysis of software failure data and how to use them for making some decisions about the software system is presented in [Section 4.3](#) where we revisit the two case studies in software testing introduced in [Brandt et al. \(2007b\)](#).

CHAPTER 4

A NEW STATISTICAL SOFTWARE RELIABILITY TOOL

In this chapter, we report on the status of a *new software reliability tool* to perform statistical analyses of software failure data based on the approach described in chapters 2 and 3. This work was partially presented in [Boon et al. \(2007\)](#) and [Brandt et al. \(2007a\)](#). Existing tools for software reliability analysis like Casre and Smerfs³ do not make full use of state-of-the-art statistical methodology or do not conform to best practices in statistics. Thus, these tools cannot fully support sound software reliability analyses. For that reason, we decided to build a new tool that uses specific algorithms for the models (paying special attention to convergence issues), that is platform independent and that encourages applying best practices from statistics and can easily be extended to incorporate new models. To help the user with problems like data type identification or model selection, this tool has a user-friendly interface that indicates the correctness of the initial data or model user choices. The tool's interface is programmed in Java (platform independent) and the statistical computations are programmed in R (see www.r-project.org). R is an open-source free software maintained by a group of top-level statisticians and is rapidly becoming the standard programming language within the statistical community.

The remainder of this chapter is organized as follows. In Section 4.1 we comment on some general problems found during the implementation of the tool and the proposed solutions. We describe the functionalities implemented in the tool in Section 4.2, paying special attention to the statistical features. Finally, as an example of a real application of the tool, we revisit the two case studies presented in [Brandt et al. \(2007b\)](#) in Section 4.3.

4.1 General remarks about the implementation

The tool is built upon the step-by-step approach presented in Chapter 3 and therefore, it meets the requirements described there. Despite the large number of models known in the literature, the tool is in an early stage of development, in the sense that it has only three models implemented so far: Jelinski-Moranda (see Section 2.5.1), Goel-Okumoto (see Section 2.6.1) and Duane (see Section 2.6.3).

One of the main problems we found during the implementation phase is that, as mentioned in Chapter 3, for many models (also for the most well-known ones) the procedures to analyze interval-time data are not worked out in detail or simply do not exist. A provisional solution to this problem consists of approximating the interval-time data by point-time data distributing the corresponding number of faults at random on each interval obtaining in that way new data points that are

considered as point-time data. This approach has been implemented in the tool to study those cases where no algorithms are known.

The software reliability literature is also lacking results regarding confidence intervals for parameters of the models and goodness-of-fit tests. As discussed in Section 3.4, asymptotic confidence intervals for parameters are usually computed based on the asymptotic normality of ML estimators (in terms of Fisher information) and based on the Wilks statistic. Joe (1989) and van Pul (1992b) show that asymptotic confidence intervals based on the Wilks statistic are preferred to those obtained from asymptotic normality. However, no stable algorithm to derive confidence intervals based on the Wilks statistic has been developed yet. Since the literature does not provide us a better alternative, algorithms using Fisher information have been implemented for those cases where no algorithms are known. Similar remarks can be made for goodness-of-fit tests. For the Goel-Okumoto and Duane models we have implemented the goodness-of-fit test for general NHPP models developed in Zhao and Wang (2005) (see Section 3.5). Conditional on the last observation, the Kolmogorov test can also be used for NHPP models. For the Jelinski-Moranda model (as for GOS models in general) the Kolmogorov test can be used as we discussed in Section 3.5. Unfortunately, these tests are developed for point-time data only. Thus, for interval-time data we use the approximation to the point-time case described before.

The two graphical and the two statistical trend tests described in Section 3.2 have been implemented in the tool. Running averages and TTT plots are implemented for both point-time and interval-time data (although the interval-time version of the TTT plot is again an approximation to the point-time case). The Laplace trend test is implemented for both point-time and interval-time data. However, for interval-time data it is required equal length of intervals (cf. Kanoun et al. (1991)) otherwise it should not be applied. To overcome this restriction the MIL-HDBK-189 trend test has also been implemented.

4.2 Main functionalities

In this section we describe the main functionalities of the tool. Initially, the tool GUI displays four menu items as we can see in Figure 4.1. In the beginning only the *Data* and *Help* menus are enabled. To have access to the working environment we have to select the option *Import* from the *Data* menu in order to have a data set to work with. An open dialog like in Figure 4.2 pops-up. We look for the data file containing the desired data set and then we click on *Open*. After the data set has been loaded in the tool, the menus *Graphics* and *Analysis* are enabled. We can now select the type of data we are working with. We will explain this option in detail in Section 4.2.3 as part of the *Data Type* menu. Moreover, we distinguish two more windows that will remain visible all the time: one displaying data points and another one producing graphical outputs. The data window is shown in Figure 4.3. Three different tabs can be identified at the bottom of the window. The first one, called *Data*, contains the imported data, and the third one, called *Model Analysis*

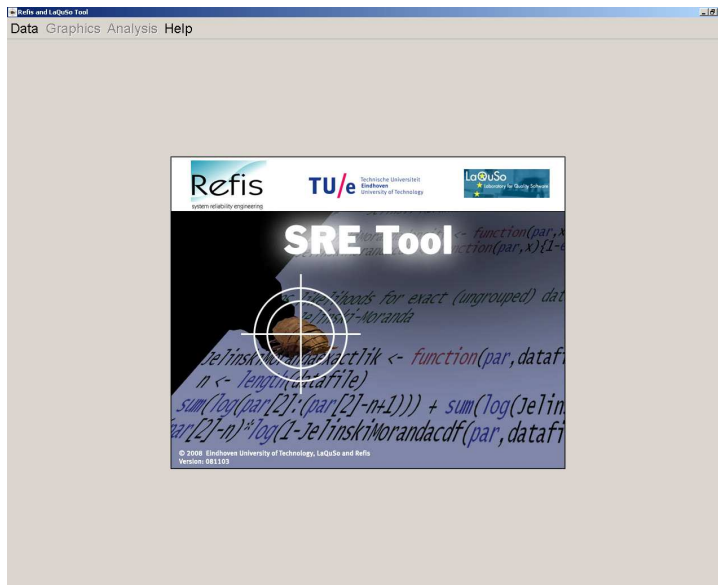


Figure 4.1: Software reliability tool GUI. In the beginning only the *Data* and *Help* menus are enabled.

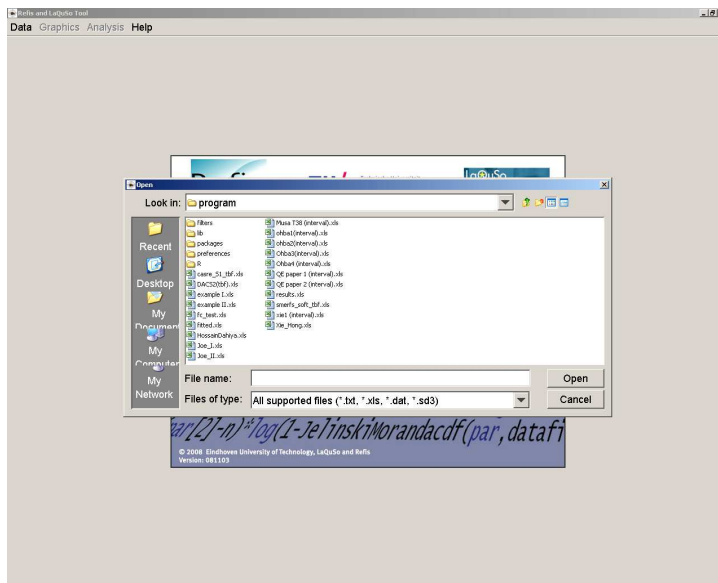
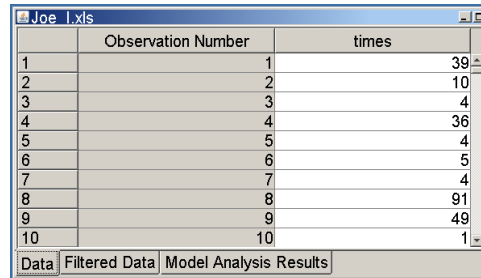


Figure 4.2: Browsing the data file. A data set is needed to access the working environment of the tool.



	Observation Number	times
1	1	39
2	2	10
3	3	4
4	4	36
5	5	4
6	6	5
7	7	4
8	8	91
9	9	49
10	10	1

Figure 4.3: The data window displays the original data, filtered data and the data produced after model analysis.

Results, will show the data originated after model analysis (fitted values, estimated intensity, ...). The tool gives the user the possibility of filtering the initial data set (as an option item of the *Data* menu as we will see in Section 4.2.1). In case we use this option, the resulting filtered data is displayed in the *Filtered Data* tab of the data window. On the other hand, the *Graphical Output* window will display the last plot produced while the tool is being used. An example of this can be seen in Figure 4.4 where a data set in the form of point-time data taken from Joe (1989) has been plotted against time. Typical options like *Copy to Clipboard*, *Export* (supporting

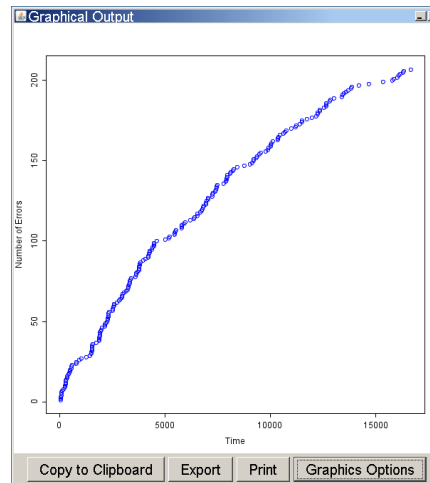


Figure 4.4: *Graphical Output* window. Cumulative failure times vs. cumulative number of detected faults for point-time data.

encapsulating postscript (eps), jpg and png formats) or *Print* are included in this window. We can also change some characteristics of the plot (like the title, the colour and the type of line) using the button *Graphics Options*. In the remainder

of this section we explain the tool's menu options in detail, paying special attention to the *Analysis* menu.

4.2.1 Data menu

Import Imports data files in txt, xls, dat (Casre files) and sd3 (Smerfs³ files) formats. After loading the data set, the working environment is displayed.

Export Exports data files in txt and xls formats.

Filter Data This window (see Figure 4.5) allows us to refine the data set and keep a subset of observations. This option is of special interest since we may work

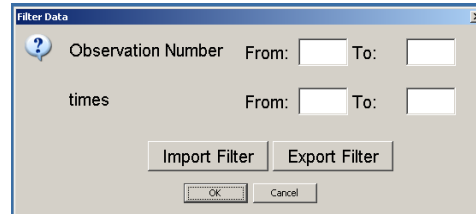


Figure 4.5: *Filter Data* window (allows data set refinement).

with the subset of the original data set that showed reliability growth. The filtered data set can also be exported and imported (in the formats already mentioned).

Remove Filter Removes the data filters created beforehand.

Transform Variable With this option we can create new variables applying basic operations to the variables of the current data set. The available operations can be observed in Figure 4.6.

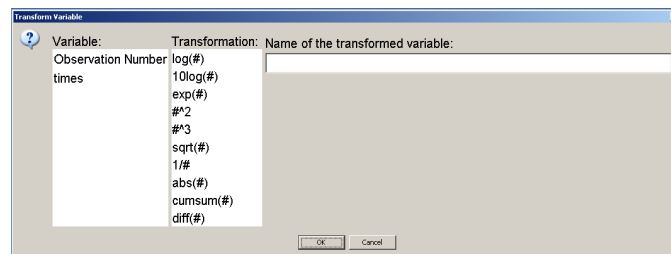


Figure 4.6: *Transform Variable* window with available operations.

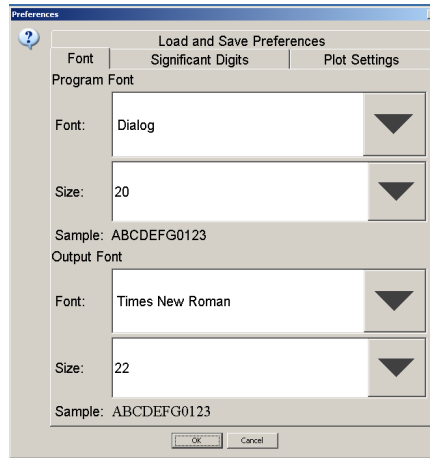


Figure 4.7: *Preferences* window (allows changing user preferences).

Preferences User preferences can be set using this option. Font type and size can be chosen for both program GUI and output (see Figure 4.7). The number of significant digits for the calculations can be selected from 1 up to 8 (by default set to 3). We can also decide the default colours and types of lines to be used in the plots. These options can be saved (or loaded) to (from) a txt file.

Exit Closes the GUI and terminates the program execution.

4.2.2 Graphics menu

Create Scatter Plot As a general recommendation it is advisable to look at data before starting any kind of statistical analysis (see Section 3.1). With this menu option we can generate plots that will be shown in the *Graphical Output* window. After clicking on this item, a window named *Select Plot Variables* pops-up (see Figure 4.8). We can now select the variables we want to plot and assign them to the X or Y axes or change some display options of the plot (like colour or type of line) using the *Plot Style* button. The *Draw Graph* button will finally produce the graph. On the lower part of the window we can set the main title and the labels of the axes of the plot.

Copy Current Plot to Clipboard Copies the plot that is displayed on the *Graphical Output* window to the clipboard. Thus, the plot is cached and can be transferred between documents or applications, via copy and paste operations.

Export Current Plot Selecting this option a save dialog appears to export the current plot to a graphic file. The formats supported are eps, jpg and png.

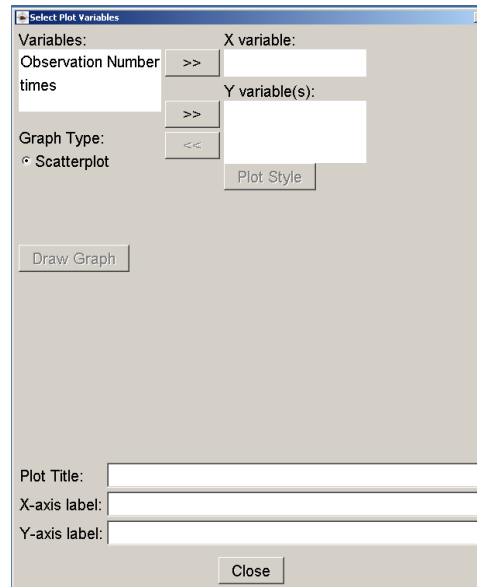


Figure 4.8: *Select Plot Variables* window (allows generating scatter plots).

Print Current Plot With this option a print dialog pops-up allowing us to print the current plot.

4.2.3 Analysis menu

Model Selection Wizard Due to the abundance of software reliability models, the selection of appropriate ones is often a difficult task. To help the user to succeed with this, the tool has a special window called *Model Selection Wizard* (see Figure 4.9). This wizard incorporates the matrix-based procedure to support the initial choice of suitable models introduced in Section 3.3. In Figure 4.9 we can observe a list of common software reliability assumptions taken from the literature and used in different software reliability models followed by a column showing some of these models and another one called *Score*. Each assumption is related to each model with a weight defining the relative importance of the assumption for the model. In this case, the weights vary from 1 (lowest relative importance) to 3 (highest relative importance). After selecting all the assumptions every model receives a final score and the models are sorted by relevance. If all requirements and selected assumptions of a model are satisfied, then the score for this model is 100%. In all other cases the score of the model is defined using the relative importance (weight) of the applicable characteristics of the model. Thus, the higher score a model has, the better the model will fit the assumptions. Consequently, as initial model choice we may use the models with the higher scores. As mentioned in Section 3.3, the weights used here are *subjective* choices. Further research is needed to develop objective criteria.

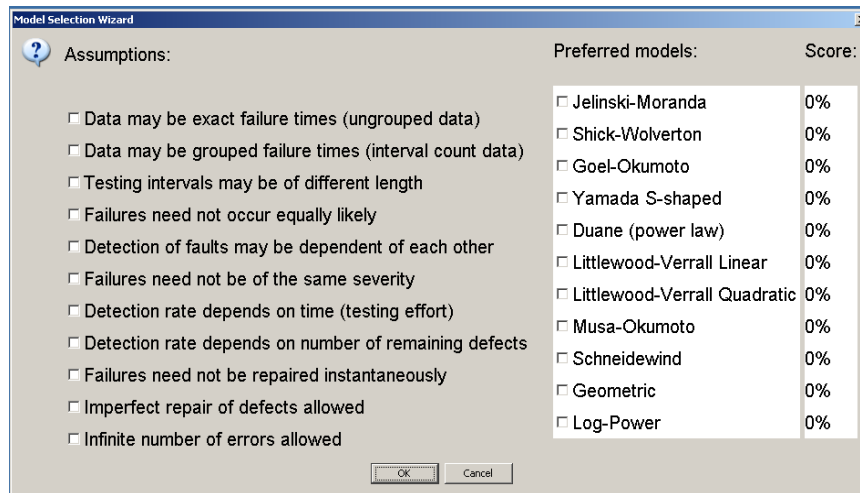
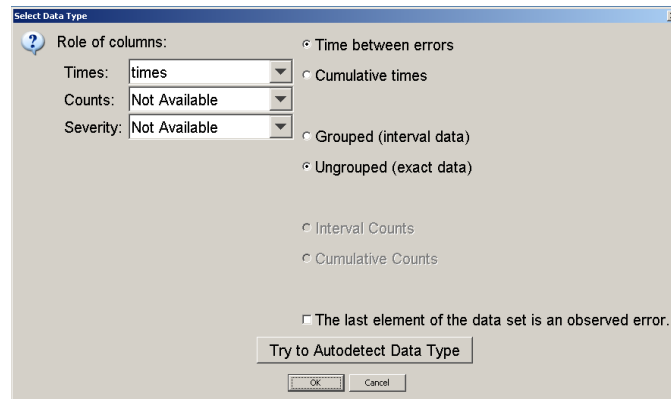
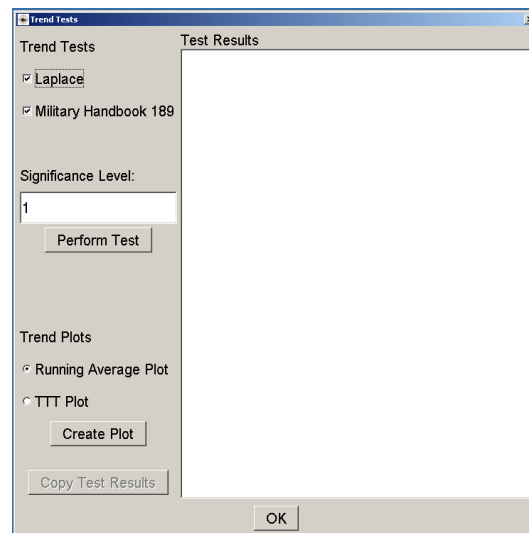


Figure 4.9: *Model Selection Wizard* window containing some of the most popular software reliability assumptions and models.

Data Type As mentioned in Chapter 3, the first step in software reliability analyses is to identify the kind of data we have. To help the user with this problem the tool possesses a window called *Select Data Type* (see Figure 4.10) in which we have the option to select the type of data of our data set. We assume that software failure data appears in the form of either point-time or interval-time observations (see Section 3.1). Note that when we have point-time observations (*Ungrouped (exact data)* option), the observed failures are reported individually either by the failure times (*Cumulative times* option) or by the corresponding times between failures (*Time between errors* option). In this case, we also have the possibility of choosing when data is failure truncated (by selecting the option *The last element of the data set is an observed error.*). When the data set is of the form of interval-time observations (*Grouped (interval data)* option), the observed failures are reported in intervals either by the exact number of failures observed in each interval (*Interval Counts* option) or by the cumulative number of failures observed after each interval (*Cumulative Counts* option). If the user does not have any information about the type of data that is using, then the option *Try to Autodetect Data Type* results of special interest. With this option the program estimates the type of data of the selected data file. Checking the number of columns (interval-time or point-time) or the growth of the data (time between failures or cumulative times) the tool may find out what kind of data we are using.

Trend Tests Besides the graphical methods to study the trend of data discussed in Section 3.2 (running averages and TTT plot), the tool has also the Laplace and the MIL-HDBK-189 trend tests already implemented. The *Trend Tests* window is

Figure 4.10: *Select Data Type* window.Figure 4.11: *Trend Tests* window.

shown in Figure 4.11. We can now select the tests we would like to use and fix the significance level at which the tests will be performed. After clicking on *Perform Test*, the results are shown on the right-hand side of the window. Moreover, a graphical interpretation of the test like in Figure 4.12 will also be displayed on the *Graphical Output* window. The graphical interpretation consists of a plot of the corresponding density function of the test statistic (normal or chi-square), the critical region of the test (shaded area) and the value of the test statistic (denoted by L for the Laplace test and by M for the MIL-HDBK-189 test). Figure 4.12 shows the result of the application of the Laplace (top) and the MIL-HDBK-189 (bottom)

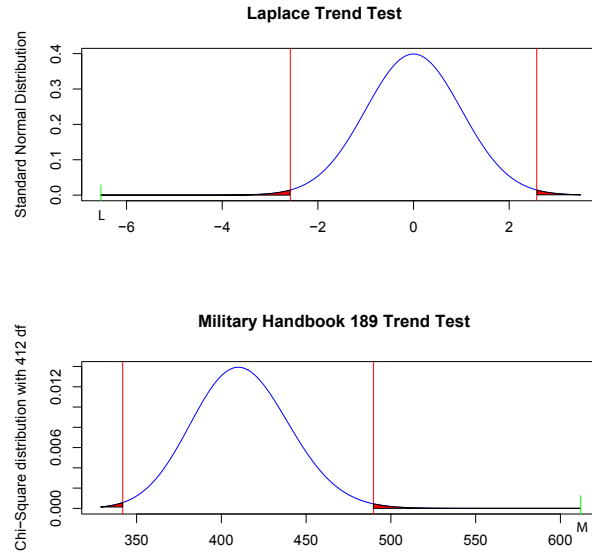


Figure 4.12: Laplace (top) and MIL-HDBK-189 (bottom) trend tests graphical interpretation. The Laplace statistic (L) is small and the MIL-HDBK-189 statistic (M) is large. Reliability growth can be assumed.

trend tests for the data set in [Joe \(1989\)](#). Note that both tests can be applied here since the data set in [Joe \(1989\)](#) is in the form of point-time data (see [Section 3.2](#)). Since the Laplace statistic is small and the MIL-HDBK-189 statistic is large, the hypothesis of an HPP process is rejected in favour of reliability growth. General remarks mentioned in [Section 3.2](#) should be taken into account here. For example, if our data set consists of interval-time data but the intervals do not have the same length, then the Laplace test should not be selected. When the result of the trend tests are positive, i.e., the data shows reliability growth, the next step is to perform analysis of the data applying software reliability growth models.

Analyse Models The *Model Analysis* window is shown in [Figure 4.13](#). The models initially selected, with the help of the *Model Selection Wizard* (see [Figure 4.9](#)), can be seen now in this window. Our main purpose here is to estimate the parameters of the models, based on the observed failure data, in order to predict future behaviour of the system. For that reason, we must select at least one of the models that appear in this window. For the chosen models, we can compute both Maximum Likelihood (ML) and Least Squares (LS) estimators of the parameters (but not at the same time). Confidence intervals for the parameter estimates at any specified confidence level (by default set to 95%) are also computed here. Note that this does

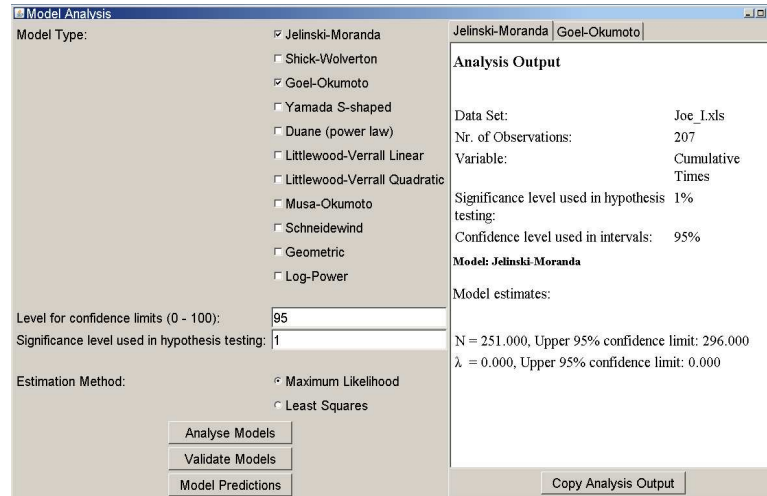


Figure 4.13: *Model Analysis* window. Analysis for the Jelinski-Moranda model.

not imply that the selected models are appropriate to make predictions. Before computing any predicted quantity of interest we should determine to what extent the selected models fit data. This can be done by *Validate Models* as we will see later in this section. For us *model validation* has to do with *model fitting* and can be accomplished by the application of *goodness-of-fit tests*, as explained in Section 3.5. Goodness-of-fit tests are *statistical* tests. Therefore, the significance level to perform hypothesis testing (by default set to 1%) must be specified.

To end with this section we give some implementation details about the estimation procedures. LS estimation is usually not problematic but numerical problems often arise in ML estimation (see Section 3.4). Although both ML and LS procedures are defined for both interval-time and point-time data, most of the algorithms known in the literature are for point-time data only. For the available models of the tool we have introduced the following implementations:

- Goel-Okumoto: The ML algorithm for point-time data is based on the results in Hossain and Dahiya (1993) and Knafl and Morgan (1996) and for interval-time data is based on Hossain and Dahiya (1993) and Knafl (1992). There is no stable algorithm for confidence intervals known in the literature. Thus, approximate results using Fisher information have been implemented.
- Duane: For point-time data the ML upper confidence limit for the shape parameter b is due to Crow (1974) and for the scale parameter a is due to Gaudoin et al. (2006). The ML algorithm for interval-time data uses the results from Hossain and Dahiya (1993) and Knafl (1992). There is no stable algorithm for confidence intervals known in the literature when the data is of the form of interval-time observations. Thus, approximate results using Fisher information have been implemented.

- Jelinski-Moranda: The ML algorithm and the confidence intervals for point-time data are taken from [Finkelstein et al. \(1999\)](#). For interval-time data no stable algorithm is known. Therefore, we have implemented the approximation by point-time data method explained in Section 4.1.

To compute parameter estimates (and confidence intervals) the *Analyse Models* button should be clicked. Figure 4.13 shows the result of the analysis for the data set in [Joe \(1989\)](#) (point-time data). On the right-hand side of the window we observe the estimates of the parameters of the Jelinski-Moranda model with their corresponding upper confidence limits. The data window in Figure 4.14 shows the fitted values for the Jelinski-Moranda and Goel-Okumoto models. Also the *Graphical Output*

Observ...	times	Cumula...	Jelinski...	Jelinski...	Goel-O...	Goel-O...	
1	1	39	39	1.004	0.026	2.283	0.040
2	2	10	49	1.261	0.026	2.668	0.037
3	3	4	53	1.363	0.026	2.815	0.036
4	4	36	89	2.285	0.026	4.010	0.031
5	5	4	93	2.387	0.026	4.132	0.030
6	6	5	98	2.515	0.026	4.282	0.030
7	7	4	102	2.617	0.026	4.401	0.029
8	8	91	193	4.929	0.025	6.801	0.024
9	9	49	242	6.165	0.025	7.937	0.022
10	10	1	243	6.190	0.025	7.959	0.022

Figure 4.14: Fitted values data for the Jelinski-Moranda and Goel-Okumoto models.

window displays the observed values and the fitted models (by default represented by circles and a solid curve, respectively) like in Figure 3.5. Model validation and predictions are available only after parameter estimation has been performed. This can be done by clicking the buttons *Validate Models* and *Model Predictions* in the analysis window (see Figure 4.13) or with the corresponding option buttons of the *Analysis* menu.

Validate Models The *Model Validation* window is shown in Figure 4.15. We must select the models we wish to validate and the corresponding goodness-of-fit test to be performed. For GOS models only the Kolmogorov test is implemented. For NHPP models a conditional version of the Kolmogorov test and the test presented in [Zhao and Wang \(2005\)](#) are implemented. We refer to Section 3.5 for details on these tests. Figure 4.15 shows the result of the Kolmogorov and Zhao-Wang tests for the data set in [Joe \(1989\)](#) (point-time data) for the Goel-Okumoto model. The results for all the other selected model are displayed in different tabs of the window.

Model Predictions Figure 4.16 shows the *Model Predictions* window. We must select models in order to compute the failure intensity at any specified future time, the expected time to next failure, the probability of no failure in a given interval or the estimated number of remaining faults. Figure 4.16 shows all those quantities for the data set in [Joe \(1989\)](#) (point-time data) and the Jelinski-Moranda model.

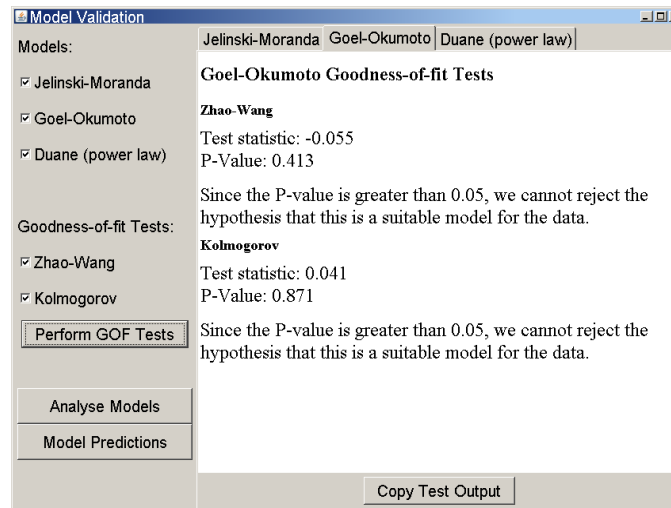


Figure 4.15: *Model Validation* window. Kolmogorov and Zhao-Wang tests for the Goel-Okumoto model.

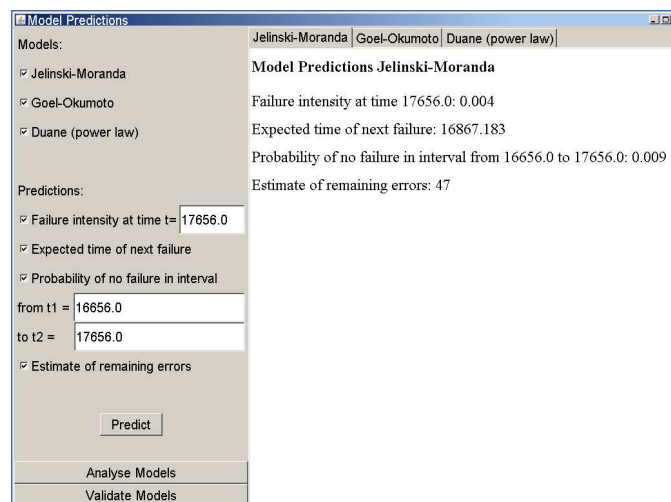


Figure 4.16: *Model Predictions* window. Several predictions for the Jelinski-Moranda model.

Plot Fitted Models This option is enabled only after the analysis has been carried out. It allows us plotting the observed values and the fitted models that we selected for the analysis producing a plot like in Figure 3.5.

4.2.4 Help menu

Tool Help Help files of the tool.

SRE Help Glossary of terms and background on chosen algorithms.

About... Information about the developers, version of the tool, etc.

4.3 Two examples of applying reliability growth models in software development

The goal of this section is to illustrate how the tool can be used to analyse real software failure data. For that reason, we revisit the two case studies in software testing presented in [Brandt et al. \(2007b\)](#). The first case study is about administrative software of an insurance company and the second one is on safety of software control of a closable dam as part of a sea flood protection system. For more details on the problems presented here we refer to [Brandt et al. \(2007b\)](#). In order to statistically analyze these data sets, we propose to follow the steps described in Chapter 3, i.e., data description, trend tests, model type selection, estimation of model parameters, model validation and model interpretation.

4.3.1 Administrative software at an insurance company

The data set of this problem, reproduced in Table 4.1, is of the form of interval-time observations. It consists of a number of failures reported in intervals (of different length) of tested hours per week. The total number of faults detected by time

Week	Failures reported	Test hours
1	8	18
2	45	28
3	27	45
4	30	71
5	37	84
6	49	84
7	53	97
8	17	74
9	12	43
10	2	12
Total	280	556

Table 4.1: Failures counts and testing hours.

$\ell_{10} = 556$ is given by $m = 280$ and the length of the different observation intervals is given in the column *Test hours* of Table 4.1. In order to study whether the

data set presents any kind of trend, we first look at the running averages, i.e., we plot m_i/d_i , for all $i = 1, \dots, 10$, where m_i denotes the number of faults detected in the i^{th} interval, and d_i denotes the length of the i^{th} interval, given by columns *Failures reported* and *Test hours* in Table 4.1, respectively. Figure 4.17 shows that the running averages decrease after the first week, thus a growth in reliability is indicated. Since the test intervals have different lengths, the Laplace trend test

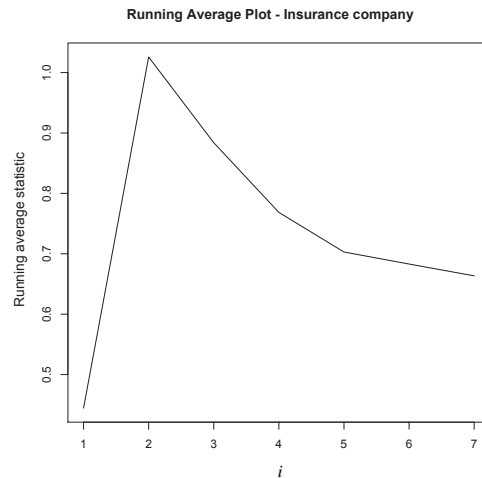


Figure 4.17: Running averages (insurance company data). A decreasing plot indicates reliability growth.

for interval-time data cannot be applied in this case. Nevertheless, for the MIL-HDBK-189 trend test (cf. [Military Handbook](#)) this is not a restriction and it can be applied. However, as explained in Section 3.2, the condition $md_i/l_{10} > 5$ must be satisfied for all $i = 1, \dots, 10$. Since this condition holds for this data set, the test has been performed for a 5% significance level. The (graphical) result can be seen in Figure 4.18. The test statistic (denoted by M) is large (77.72) and the p -value almost zero. Thus, the null hypothesis of a homogeneous process is rejected in favour of reliability growth. In [Brandt et al. \(2007b\)](#) it is shown that suitable reliability models for our problem are the Goel-Okumoto and the Yamada S-shaped models. Unfortunately, as we mention in Section 4.1, the Yamada S-shaped model is not yet implemented in the tool. To exploit the maximum number of functionalities of the tool we have decided to perform data analysis considering the three models implemented so far in the tool (Goel-Okumoto, Duane (power-law) and Jelinski-Moranda). Note that this can also be seen as if we adopt the point of view of a user that has no previous knowledge about the adequacy of selecting specific models for the problem at hand. The remaining steps of the analysis are estimation of the parameters, validation and interpretation of the selected models. For both parameter estimation and model validation we do the following. We consider a subset of the observations where the last 20% of the observations have been discarded. In this

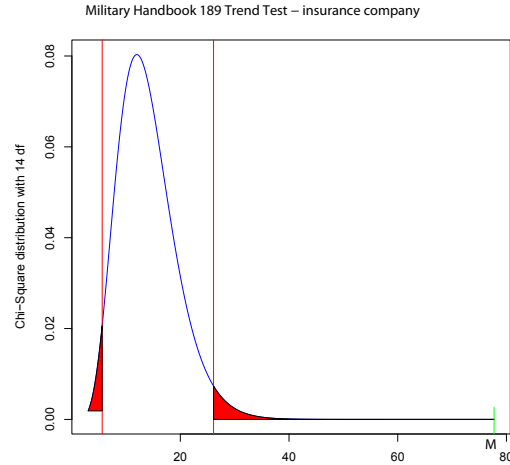


Figure 4.18: MIL-HDBK-189 trend test (insurance company data). The test statistic (M) is large. The null hypothesis of an HPP is rejected in favour of reliability growth.

case, we keep the observations in Table 4.1 corresponding to the first 8 weeks. With this *filtered* data set we do parameter estimation and model validation. In particular, we are interested in checking how the fitted expected number of failures for weeks 9 and 10 are compared to the real observed values for those weeks. Finally, for model interpretation we consider the original data set consisting of 10 weeks since this corresponds to the whole observation interval and makes no sense to make predictions before this point. The filtered data set fulfills conditions for the existence of ML estimators for the three models as explained in [Knafl and Morgan \(1996\)](#). The corresponding ML estimates can be seen in Table 4.2. We have also plotted the fit-

Model	Parameter Estimates
Goel-Okumoto	$\hat{a} = 424.313$ $\hat{b} = 0.002$
Duane	$\hat{a} = 1.950$ $\hat{b} = 0.791$
Jelinski-Moranda	$\hat{N} = 439.000$ $\hat{\lambda} = 0.001$

Table 4.2: ML estimates for the Goel-Okumoto, Duane and Jelinski-Moranda models (insurance company data).

ted models in Figure 4.19. The expected number of failures in weeks 9 and 10 given by the fitted models (together with the corresponding observed failures) are shown

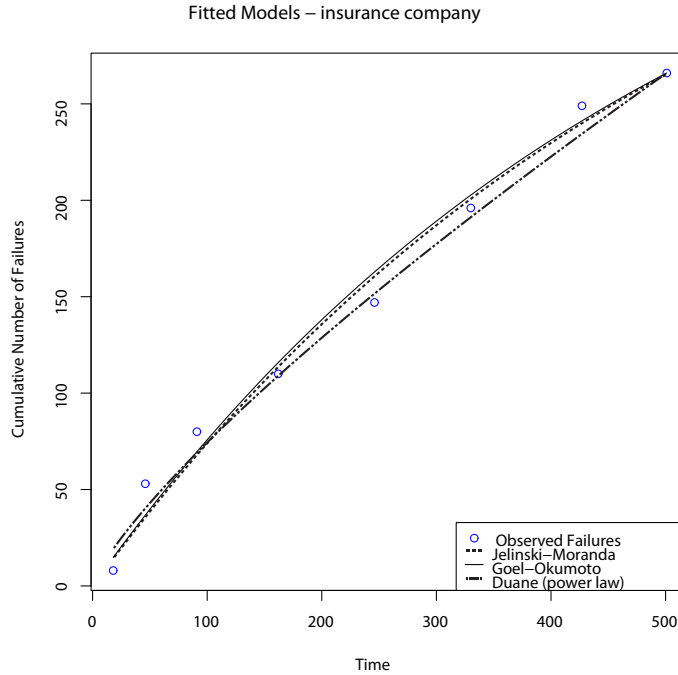


Figure 4.19: Fitted models (insurance company data).

in Table 4.3. In this case, we observe how the values fitted by the Goel-Okumoto

Week	Defects reported	Goel-Okumoto	Duane	Jelinski-Moranda
9	278	278.83	283.93	274.10
10	280	282.22	288.88	277.63

Table 4.3: Cumulative number of failures observed and predicted in weeks 9 and 10.

model are those which are closer to the observed values, especially in week 9 which is almost a perfect fit. For week 10 the difference between the expected number of failures fitted by the Goel-Okumoto and Jelinski-Moranda models and the observed value is almost the same (in absolute value). However, the Goel-Okumoto model fits in a pessimistic way (more expected failures than observed in reality) and the Jelinski-Moranda in an optimistic way. The expected number of failures fitted by the Duane model is the one that differs the most from the observed number of failures. However, we cannot conclude that this model does not properly fit the data set. To come to such a conclusion goodness-of-fit tests must be applied. Goodness-of-fit tests are hard to use here because of the limited number of observations, as

it is often the case in software reliability data sets. However, we have performed the unconditional goodness-of-fit test for NHPP models proposed in [Zhao and Wang \(2005\)](#) (see also Section 3.5) for both the Goel-Okumoto and Duane models. The result of this test supports the idea that both models seems to be suitable for the data (p -value > 0.01). The Kolmogorov test has been computed for the Jelinski-Moranda model. The conclusion in this case is that the Jelinski-Moranda model does not seem to fit the data very well (p -value < 0.01). This is not completely obvious if one looks at Figure 4.19 since they all seem to have a similar behaviour. Therefore, we may assume that only the Goel-Okumoto and Duane models are valid to describe the failure detection process of the system under study. Finally, we would like to say something statistically about the future behaviour of the system. This can be accomplished by considering the whole data set in Table 4.1 and calculating parameter estimates again. With these estimates we compute certain quantities of interest like the failure intensity at a specified time in the future, the expected time to the next failure, the probability of no failure in a given interval or the estimated number of remaining faults. All these quantities and parameter estimates for both the Goel-Okumoto and Duane models are shown in Table 4.4. We observe that the

	Goel-Okumoto	Duane
Parameter estimates	$\hat{a} = 438.5508$ $\hat{b} = 0.0018$	$\hat{a} = 2.2051$ $\hat{b} = 0.7663$
Failure intensity $t = 586$	0.2746	0.3812
Expected time next fault	559.45	558.59
Prob. no fault in (556, 586]	0.0002	0.0000
Expected remaining faults	159	—

Table 4.4: ML estimates for the Goel-Okumoto and Duane models and some predictions (insurance company data).

expected time to next failure is almost the same for both models. However, the failure intensity at $t = 586$ is larger for the Duane model. For that reason we should expect a higher number of estimated failures by the Duane model. This can also be observed in Figure 4.20 where the estimated reliability function is plotted for both models. We can see that the reliability function for the Goel-Okumoto model is higher than for the Duane model. Therefore, we can conclude that the Goel-Okumoto model is more optimistic than the Duane model. Moreover, according to Table 4.3 the Goel-Okumoto predicts better than the Duane model. Therefore, in this case we may conclude that the Goel-Okumoto is the one that better describes the behaviour of the failure observation process of the software system. However, as mentioned in Section 3.5, prequential procedures should be used to compare the predictive performance of different models. Unfortunately, this is not yet implemented in the tool.

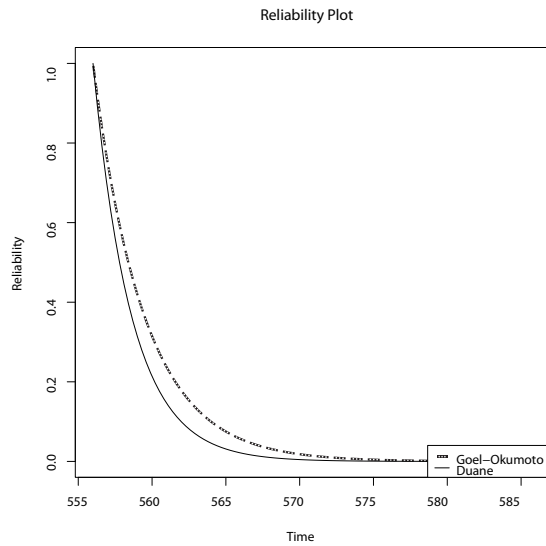


Figure 4.20: Reliability plot for the Goel-Okumoto and Duane models (insurance company data).

4.3.2 A closable dam operating system

The data set of this problem is of the form of interval-time observations consisting of a number of observed failures reported per week. A reduced version of it is shown in Table 4.5. The total number of faults detected by time $\ell_{14} = 14$ is given by $m = 38$.

Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Defects	8	2	4	4	1	4	5	4	1	1	2	0	1	1

Table 4.5: Failures counts per test week.

Note that the length of the different observation intervals is the same for all the intervals and is equal to 1 week. In order to study whether the data set shows any kind of trend, we first look at the running averages (see Section 3.2). Figure 4.21 shows the running averages plot of reported failures per week. The running averages decrease indicating thus a growth in reliability. In this case, the test intervals have the same length, thus the Laplace trend test for interval-time data can be applied (see Section 3.2). Note that in principle the MIL-HDBK-189 trend test could also be used, however, the condition $md_i/\ell_{14} > 5$, for all $i = 1, \dots, 14$, does not hold for this data set. Thus, only the Laplace trend test has been performed for a 5% significance level. The (graphical) result can be seen in Figure 4.22. The test statistic (denoted by L) is small (-3.26) and the p -value is 0.001. Thus, the null

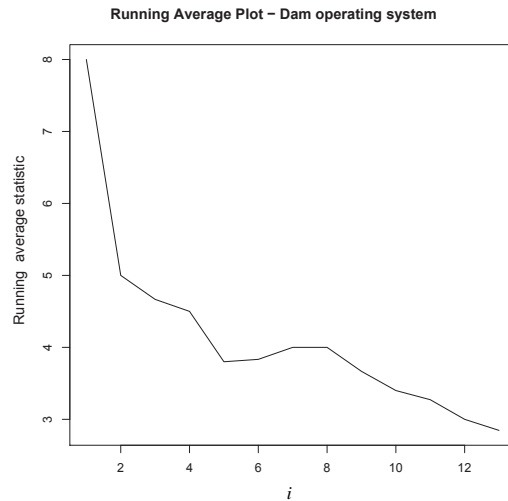


Figure 4.21: Running averages (dam operating system data).

hypothesis of a homogeneous process is rejected in favour of reliability growth. In

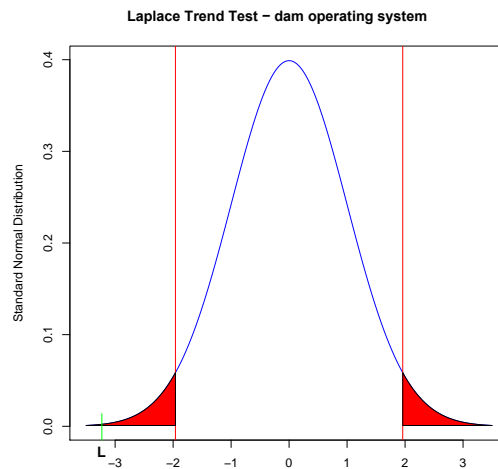


Figure 4.22: Laplace trend test (dam operating system data). The test statistic (L) is small. The null hypothesis of an HPP is rejected in favour of reliability growth.

[Brandt et al. \(2007b\)](#) it is shown that suitable reliability models for this problem are the Goel-Okumoto, the Yamada S-shaped and the Generalized Poisson models. Like in the case of the insurance company problem (see Section 4.3.1), we have decided

to perform data analysis considering the three models implemented so far in the tool (Goel-Okumoto, Duane (power-law) and Jelinski-Moranda). For the remaining steps of the analysis (estimation of the parameters, validation and interpretation of the selected models) we do like in the case of the insurance company. For both parameter estimation and model validation we consider a subset of the observations consisting of the first 11 weeks. With this *filtered* data set we do parameter estimation and model validation. Again, we are interested in checking how the fitted expected number of failures for weeks 12 to 14 are compared to the real observed values for those weeks. Finally, for model interpretation we consider the original data set consisting of 14 weeks. The filtered data set fulfills the conditions for the existence of ML estimates for the three models as explained in [Knafl and Morgan \(1996\)](#). The corresponding ML estimates can be seen in Table 4.6. We have also plotted the fitted

Model	Parameter Estimates
Goel-Okumoto	$\hat{a} = 55.693$ $\hat{b} = 0.094$
Duane	$\hat{a} = 7.323$ $\hat{b} = 0.664$
Jelinski-Moranda	$\hat{N} = 47.000$ $\hat{\lambda} = 0.128$

Table 4.6: ML estimates for the Goel-Okumoto, Duane and Jelinski-Moranda models (dam operating system data).

models in Figure 4.23. The expected number of failures in weeks 12, 13 and 14 given by the fitted models (together with the corresponding observed failures) are shown in Table 4.7. In this case, we observe how the values fitted by the Duane model

Week	Defects reported	Goel-Okumoto	Duane	Jelinski-Moranda
12	36	37.77	38.14	36.88
13	37	39.39	40.22	38.09
14	38	40.86	42.25	39.16

Table 4.7: Cumulative number of failures observed and predicted in weeks 12, 13 and 14.

are those which are further from the observed ones. The Jelinski-Moranda model fits values closer to the observed ones but like in case of the Goel-Okumoto model it is also pessimistic. We have performed the unconditional goodness-of-fit test for NHPP models proposed in [Zhao and Wang \(2005\)](#) for both the Goel-Okumoto and Duane models. The result of this test supports the idea that only the Duane model seem to be suitable for the data (p -value > 0.01). The Kolmogorov test has been computed for the Jelinski-Moranda model. The conclusion in this case is that we reject the hypothesis that the failure process is described by the Jelinski-Moranda

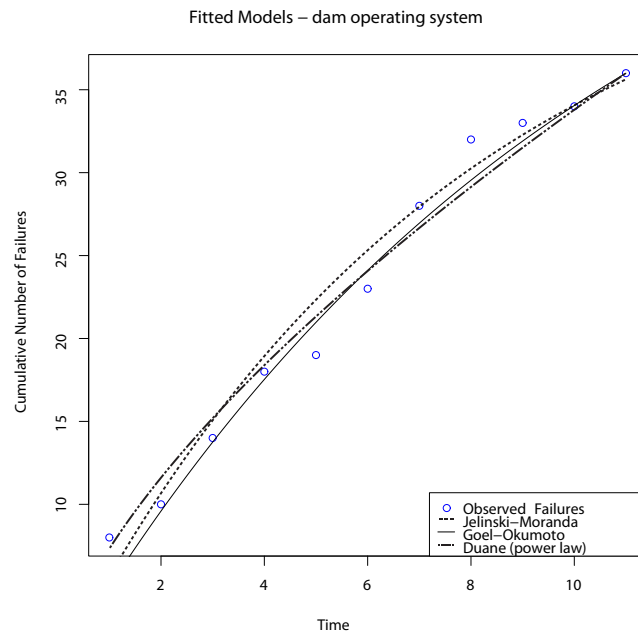


Figure 4.23: Fitted models (dam operating system data).

model. Therefore, we may assume that only the Duane model is valid to describe the failure detection process of the system under study. Finally, we consider the whole data set in Table 4.5 and calculate parameter estimates again in order to compute the failure intensity at a specified time in the future, the expected time to the next failure, the probability of no failure in a given interval or the estimated number of remaining faults. All these quantities and parameter estimates for the Duane model are shown in Table 4.8. We observe that the next failure is expected after the

	Duane model
Parameter estimates	$\hat{a} = 8.0409$ $\hat{b} = 0.5881$
Failure intensity $t = 20$	1.3781
Expected time next fault	14.6322
Prob. no fault in $(14, 20]$	0.0001
Expected remaining faults	–

Table 4.8: ML estimates for the Duane model and some predictions (dam operating system data).

middle of week 14. The failure intensity at $t = 20$ is large and the probability of no failure in the 6 coming weeks is small. In Figure 4.24 we have plotted the estimated reliability function for the Duane model. We can see that for example, after one

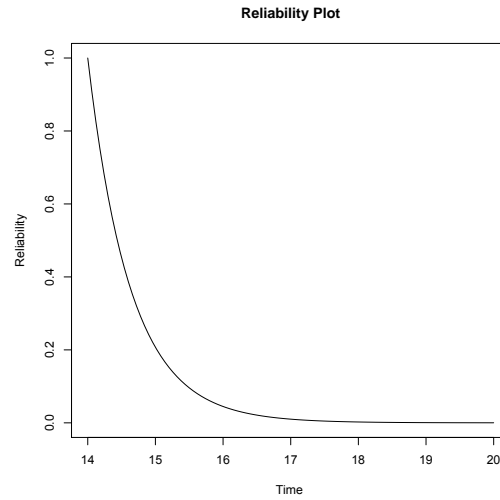


Figure 4.24: Reliability plot for the Duane model (dam operating system data).

week, the reliability function has decreased at the level of 0.2 which supports the result of expecting the next failure within one week with high probability.

CHAPTER 5

STATISTICAL APPROACH TO SOFTWARE RELIABILITY CERTIFICATION

Statistical procedures to support software release decisions are usually based on a loss function that in general considers the trade-off between the cost of extra testing and the cost of undetected faults. Such procedures are thus developed from the software producer point of view. Generally, two main decision policies are considered: policies based on observed failures (see e.g. Dalal and Mallows (1988), Morali and Soyer (2003b), Özekici and Catkan (1993), Özekici and Soyer (2001) and van Dorp et al. (1997)) and policies based on time, i.e., faults found in some time interval (see e.g. McDaid and Wilson (2001), Singpurwalla (1991) and Singpurwalla and Wilson (1994)). The first approach decides to stop or to continue testing after each failure observation while the second one stops always after testing for some fault-free interval that is optimal for a certain criterion.

Unlike the previous approaches, we do not base our decisions on a loss function but on a *certification criterion*. Certification is developed from the perspective of software users. They expect software producers to certify that the software produced performs according to certain reliability requirements. Statistical approaches to software reliability certification can be found in Currit et al. (1986) and more recently in Di Bucchianico et al. (2008). We will discuss the procedures developed there in more detail in the next section. Besides that, in this chapter we present a sequential software release procedure that certifies with high confidence that the next software failure is not observed in a certain time interval. Our procedure is developed assuming that the fault detection process of software systems can be modelled as a stochastic process with *independent times between failures*. The decision to stop or continue testing is based on the fault-free interval since the last failure observation. Such time intervals depend on the test history and, by choosing them appropriately, we can certify that the software is released in an optimal (local) time and the global risk in the procedure can be controlled. Thus, our procedure can be seen as a special case of sequential testing policy, where at each stage of testing we update our statistics with the information obtained from the test history and we decide whether to stop or to continue testing. The main difference with other policies is that we allow the software to be tested for the time interval that is optimal for our certification criterion before stopping, which occurs only in case that no faults are found during such an interval. Otherwise, we continue testing repeating this procedure in a dynamic fashion.

The remainder of this chapter is organized as follows. Previous approaches to software reliability certification are discussed in Section 5.1. In Section 5.2, we establish a general framework for Bayesian inference. Finally, in Section 5.3 we describe a certification procedure for software reliability growth models having independent times between failures. The main result of that section shows that if

the residual lifetime probability for the next failure observation is large enough, then the global risk in the procedure can be controlled.

5.1 Previous work on software reliability certification

Software producers must certify with high confidence that the software they produce is performing according to user's expectations. Statistical certification procedures provide software users a formal indication about the quality of software systems after their release. Although there exists a vast literature on software reliability it turns out that the development of statistical certification procedures for software systems has not received enough attention. Exceptions to this are [Currit et al. \(1986\)](#) and [Di Bucchianico et al. \(2008\)](#). The first approach certifies that the mean time to next failure is larger than a certain value, while the second approach certifies that the software system is fault-free. In the next subsections we briefly describe these two certification procedures and discuss the limitations we found in them.

5.1.1 Certification procedure based on expected time to next failure

A sequential certification procedure based on the predicted mean time to next failure of consecutive incremented versions of software systems is developed in [Currit et al. \(1986\)](#). The model used for the analysis is equivalent to the geometric Moranda model introduced in [Moranda \(1979\)](#). This model is usually characterized in terms of the probability distribution of the times between failures. These are assumed to be independent and exponentially distributed with parameter λc^{i-1} , for all $i \geq 1$, where $\lambda > 0$ and $0 < c \leq 1$. In particular, the density function of the i^{th} inter-failure time is given by

$$f_{X_i}(x) = \lambda c^{i-1} e^{-\lambda c^{i-1} x} .$$

Thus, as mentioned in Section 2.4.2, the process $(T_n)_{n \geq 0}$ has independent increments (see Table 2.5 for a full classification of this model). Note that when $c = 1$, the times between failures are all i.i.d. with parameter λ . Thus, the counting process $(N(t))_{t \geq 0}$ is an HPP. Note also that this model considers that successive hazard rates, given by λc^{i-1} , decrease geometrically by a factor c . Thus, we can interpret c as the effect of a fault removal. The release procedure developed in [Currit et al. \(1986\)](#) can be briefly described as follows. When a new version of the system under development has been tested, an estimate of the expected time to next failure is computed. If this estimate exceeds a prefixed value, then the software can be released. Therefore, the procedure is heuristic and does not provide statistical confidence. The concept of expected time to next failure admits a similar discussion as the one for the failure rate presented in Section 2.1 since it can be defined in different ways (cf. [Thompson \(1981\)](#)). Thus, it is very important to specify *which* expected time to next failure is being considered. In this case, the expected time to next failure is defined as the expected value of the random variable X_i . Note that, if n failures have been observed at times $0 < t_1 < t_2 < \dots < t_n$, then the conditional residual lifetime of T_{n+1} is the same as the conditional survival probability of X_{n+1} ,

i.e.,

$$\begin{aligned}
R_{T_{n+1}|T_n=t_n}(t_n, x) &= \mathbb{P}[T_{n+1} > t_n + x \mid T_{n+1} > t_n, T_n = t_n] \\
&= \mathbb{P}[X_{n+1} > x \mid T_n = t_n] \\
&= S_{X_{n+1}|T_n=t_n}(x) .
\end{aligned} \tag{5.1}$$

Therefore, the expected time to next failure can be expressed as

$$\mathbb{E}[X_{n+1} \mid T_n = t_n] = \int_0^\infty S_{X_{n+1}|T_n=t_n}(x) dx . \tag{5.2}$$

For exponential times between failures, the expected time to next failure can be easily computed due to the memoryless property of the Exponential distribution. Moreover, the expected time to next failure is the reciprocal of the hazard rate of X_{n+1} . For example, for the geometric Moranda model one has

$$\mathbb{E}[X_{n+1} \mid T_n = t_n] = \mathbb{E}[X_{n+1}] = \frac{1}{\lambda c^n} ,$$

and for the Jelinski-Moranda model (see Section 2.5.1) it follows that

$$\mathbb{E}[X_{n+1} \mid T_n = t_n] = \mathbb{E}[X_{n+1}] = \frac{1}{(N-n)\lambda} .$$

However, this metric is not suited for all software reliability models in general as we will see now. As explained in Section 2.6, for NHPP models the number of failures to be observed between two consecutive failure times is a random variable having a Poisson distribution. In this case, it follows that

$$\begin{aligned}
S_{X_{n+1}|T_n=t_n}(x) &= \mathbb{P}[X_{n+1} > x \mid T_n = t_n] \\
&= \mathbb{P}[N(t_n + x) - N(t_n) = 0 \mid T_n = t_n] \\
&= e^{-(\Lambda(t_n+x) - \Lambda(t_n))} .
\end{aligned}$$

Thus, application of (5.2) yields

$$\mathbb{E}[X_{n+1} \mid T_n = t_n] = \int_0^\infty e^{-(\Lambda(t_n+x) - \Lambda(t_n))} dx .$$

Note that in this case the expected time to next failure depends on the last failure observation $T_n = t_n$. Assuming the Goel-Okumoto model (see Section 3.4.2) for the failure process we have that

$$\mathbb{E}[X_{n+1} \mid T_n = t_n] = \int_0^\infty e^{(-ae^{-bt_n}(1-e^{-bx}))} dx . \tag{5.3}$$

Since,

$$\lim_{x \rightarrow \infty} e^{(-ae^{-bt_n}(1-e^{-bx}))} = e^{(-ae^{-bt_n})} > 0 ,$$

the integral in (5.3) does not converge. Therefore, for the Goel-Okumoto model the expected time to next failure is infinite. Thus, this approach is not applicable in general. Another important limitation is that it does not take into account the risk of making a wrong decision. The procedure developed by Currit et al. (1986) is known in the literature as the *Cleanroom Software Engineering* process. Among the papers inspired by this work we mention Cobb and Mills (1990), Linger (1993), Mills et al. (1987), Poore et al. (1993) and Selby et al. (1987). However, none of them overcome the limitations indicated above.

5.1.2 Certification procedure based on fault-free system

A different concept of statistical certification is introduced in Di Bucchianico et al. (2008). The goal of this procedure is to certify with high confidence that software systems are fault-free. The model used there will be referred later in this chapter as the *Run* model and it can be described as follows. It is assumed that the initial number of faults in the system, denoted by N , is unknown but *fixed* and finite (as in GOS models). A *test run* is defined as a finite walk through the state space of the system and it is such that at most one fault can be found in one test run. The test strategy assumes also that faults are detected independently of each other and that each undetected fault has a probability θ to be on a test run. Thus, when there are m remaining faults in the system, the probability of not finding any error in one test run is equal to $(1 - \theta)^m$. When a fault is found, it is repaired before starting testing again without introducing new faults in the system. Therefore, perfect and immediate repair is assumed. The random variable X_i , for all $i = 1, \dots, N$, is defined as the number of test runs needed to find the i^{th} software fault after repairing the $(i - 1)^{\text{st}}$ fault. Clearly, it follows a geometric distribution with parameter $p_i = 1 - (1 - \theta)^{N - i + 1}$. In practice, testing until finding the next fault will not be performed since there is no certainty about the existence of a next fault. For that reason, it is assumed that there exist integer bounds $k_i > 0$, for all $1 \leq i \leq N + 1$ such that $Y_i = \min(X_i, k_i)$. Therefore, when $Y_i = k_i$ it means that the i^{th} fault has not been found after k_i test runs after repairing the $(i - 1)^{\text{st}}$ fault. Note that the definition of Y_i corresponds to the usual definition of the times between failures in a discrete-time context but truncated at time k_i . Note also that, in fact, *only the last observation is truncated*, i.e., the observations are the values of the random variables $X_1 = x_1, X_2 = x_2, \dots, X_{i-1} = x_{i-1}$ and $Y_i = k_i$. The following statistical tests are sequentially performed: if $Y_{i+1} = k_{i+1}$, then the conclusion is that there are no faults left in the system. On the other hand, if $Y_{i+1} < k_{i+1}$, then testing continues repeating the same procedure. Thus, the parameters for the procedure are the thresholds k_i that must be chosen in such a way that the conclusion of fault-free system is given with the desired confidence. This procedure can be put into a sequential statistical hypothesis testing framework using the following hypotheses:

$$H_{0,i} : N > i \text{ vs. } H_{1,i} : N = i .$$

The test statistic is Y_i and the critical region is given by $\mathcal{C}_i = \{y \mid y \geq k_i\}$. Note that the procedure always stops in finite time since it stops when no faults are found

in k_i tests. Thus, the probability of having a type II error (not rejecting $H_{0,i}$ when $H_{1,i}$ is true) is equal to 0. The global risk in the procedure represents the probability to stop testing when there are still faults left in the system. Thus, the global risk can be defined as the type I error probability and can be controlled by properly choosing the thresholds k_i . In this way, the main limitation in Currit et al. (1986) (the procedure was developed with no statistical confidence) is overcome.

5.2 Bayesian approach

In this section we first establish a general framework for Bayesian inference providing also the notation and some important concepts to be used later in our procedure. Note that we have already briefly introduced the Bayesian approach in Section 2.8. Suppose that $Y^{(n)} = (Y_1, \dots, Y_n)$ denotes a random vector whose probability distribution (discrete or continuous) depends on some parameter φ that is unknown. Thus, we may write

$$\mathbb{P}_\varphi [(Y_1, \dots, Y_n) \in \mathcal{B}_n] = f_\varphi (\mathcal{B}_n) .$$

In Bayesian inference the parameter φ is considered as a random variable Φ with a given distribution function $F_\Phi(\varphi)$. More precisely, we assume that

$$\mathbb{P}_\varphi [(Y_1, \dots, Y_n) \in \mathcal{B}_n \mid \Phi = \varphi] = f_\varphi (\mathcal{B}_n) ,$$

so that

$$\mathbb{P} [(Y_1, \dots, Y_n) \in \mathcal{B}_n] = \int f_\varphi (\mathcal{B}_n) dF_\Phi (\varphi) .$$

The probability distribution defined by $F_\Phi(\varphi)$ is called the *prior* distribution of Φ . Without loss of generality we may assume that $Y^{(n)}$ and Φ are both continuous random vectors. Therefore, Φ can also be characterized by a density function $f_\Phi(\varphi)$. The dependence of $Y^{(n)}$ on Φ can be expressed in terms of the conditional density function of $Y^{(n)}$ given $\Phi = \varphi$, denoted by $f_{Y^{(n)}|\Phi=\varphi}(y^{(n)} \mid \varphi)$, which it is often called the *likelihood function*. Application of Bayes theorem yields

$$f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi \mid y^{(n)}) = \frac{f_\Phi(\varphi) f_{Y^{(n)}|\Phi=\varphi}(y^{(n)} \mid \varphi)}{\int f_\Phi(\varphi) f_{Y^{(n)}|\Phi=\varphi}(y^{(n)} \mid \varphi) d\varphi} .$$

In this case, the probability distribution defined by $f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi \mid y^{(n)})$ is called the *posterior* distribution of Φ . Suppose now that the prior distribution of Φ belongs to a certain class of probability distributions \mathcal{G} (Exponential, Normal, Binomial, etc.). When the posterior distribution is also in \mathcal{G} , the prior distribution is said to be *conjugate*. There are many examples of conjugate families and some of them can be found in Lee (1989)[Section 2.10].

Our main purpose is to develop a sequential certification procedure based on the fault-free interval after the last observed failure. For that reason, the following

concepts are needed. The conditional reliability function of Y_{n+1} given $Y^{(n)} = y^{(n)}$ is defined by

$$S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z | y^{(n)}) = \mathbb{P} \left[Y_{n+1} \geq z | Y^{(n)} = y^{(n)} \right], \quad (5.4)$$

for all $n \geq 0$ and $z \geq 0$. The conditional reliability function of Y_{n+1} given $\Phi = \varphi$ is defined by

$$\tilde{S}_{Y_{n+1}|\Phi=\varphi}(z | \varphi) = \mathbb{P} [Y_{n+1} \geq z | \Phi = \varphi], \quad (5.5)$$

for all $n \geq 0$ and $z \geq 0$. In a similar way we define the conditional residual lifetime of Y_{n+1} given $Y^{(n)} = y^{(n)}$, for $\Delta \geq 0$, as follows:

$$R_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z, \Delta | y^{(n)}) = \mathbb{P} \left[Y_{n+1} \geq z + \Delta | Y_{n+1} \geq z, Y^{(n)} = y^{(n)} \right]. \quad (5.6)$$

We also need to characterize when certain random variables are *conditionally independent*.

Definition 5.1 (Conditional Independence). *Let X , Y and Z be three continuous random variables with density functions $f_X(x)$, $f_Y(y)$ and $f_Z(z)$, respectively. Assuming that the joint and the conditional density functions exist, we say that X and Y are conditionally independent given Z if and only if*

$$f_{X,Y|Z=z}(x, y | z) = f_{X|Z=z}(x | z) f_{Y|Z=z}(y | z).$$

It is easy to see (cf. [Lauritzen \(1996\)](#)[p. 29]) that the above definition of conditional independence is equivalent to the following one

$$f_{X|Y=y,Z=z}(x | y, z) = f_{X|Z=z}(x | z). \quad (5.7)$$

This characterization of conditional independence is used in the proof of the following lemma.

Lemma 5.1. *Let Y_1, \dots, Y_{n+1} be continuous positive random variables whose distributions depend on the non-negative random parameter Φ , for all $n \geq 1$. If Y_1, \dots, Y_{n+1} are independent given $\Phi = \varphi$, then*

$$S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z | y^{(n)}) = \int_0^\infty \tilde{S}_{Y_{n+1}|\Phi=\varphi}(z | \varphi) f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi, \quad (5.8)$$

for all $z \geq 0$, $\varphi \geq 0$ and $y^{(n)} \in \mathbb{R}_+^n$.

Proof. We assume that all the density functions used in the proof exist. Suppose that $f_{Y_{n+1}|Y^{(n)}=y^{(n)}}(u | y^{(n)})$ is the density function of Y_{n+1} given $Y^{(n)} = y^{(n)}$, for all $u \geq 0$. Then, we may write

$$\begin{aligned} S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z | y^{(n)}) &= \mathbb{P} \left[Y_{n+1} \geq z | Y^{(n)} = y^{(n)} \right] \\ &= \int_z^\infty f_{Y_{n+1}|Y^{(n)}=y^{(n)}}(u | y^{(n)}) du. \end{aligned} \quad (5.9)$$

If $f_{Y_{n+1}, Y^{(n)}}(u, y^{(n)})$ is the joint density function of Y_{n+1} and $Y^{(n)}$, and $f_{Y^{(n)}}(y^{(n)})$ is the marginal density function of the random vector $Y^{(n)}$, then we may write

$$f_{Y_{n+1}|Y^{(n)}=y^{(n)}}(u | y^{(n)}) = \frac{f_{Y_{n+1}, Y^{(n)}}(u, y^{(n)})}{f_{Y^{(n)}}(y^{(n)})}, \quad (5.10)$$

for all $u \geq 0$ and $y^{(n)} \in \mathbb{R}_+^n$ such that $f_{Y^{(n)}}(y^{(n)}) > 0$. Let $f_{Y_{n+1}, Y^{(n)}, \Phi}(u, y^{(n)}, \varphi)$ be the joint density function of Y_{n+1} , $Y^{(n)}$ and Φ , for all $u \geq 0$, $y^{(n)} \in \mathbb{R}_+^n$ and $\varphi \geq 0$. Then, application of the law of total probability yields

$$f_{Y_{n+1}, Y^{(n)}}(u, y^{(n)}) = \int_0^\infty f_{Y_{n+1}, Y^{(n)}, \Phi}(u, y^{(n)}, \varphi) d\varphi. \quad (5.11)$$

Suppose now that the density function of Y_{n+1} given $Y^{(n)} = y^{(n)}$ and Φ is given by $f_{Y_{n+1}|Y^{(n)}=y^{(n)}, \Phi=\varphi}(u | y^{(n)}, \varphi)$, for all $u \geq 0$. Since conditional independence of Y_{n+1} and $Y^{(n)}$ given $\Phi = \varphi$ is assumed, we can apply (5.7) and write

$$f_{Y_{n+1}|Y^{(n)}=y^{(n)}, \Phi=\varphi}(u | y^{(n)}, \varphi) = f_{Y_{n+1}|\Phi=\varphi}(u | \varphi). \quad (5.12)$$

Thus, if $f_{Y^{(n)}, \Phi}(y^{(n)}, \varphi)$ is the joint density function of $Y^{(n)}$ and Φ , for all $y^{(n)} \in \mathbb{R}_+^n$ and $\varphi \geq 0$, then we can write the joint density function of Y_{n+1} , $Y^{(n)}$ and Φ as

$$\begin{aligned} f_{Y_{n+1}, Y^{(n)}, \Phi}(u, y^{(n)}, \varphi) &= f_{Y_{n+1}|Y^{(n)}=y^{(n)}, \Phi=\varphi}(u | y^{(n)}, \varphi) f_{Y^{(n)}, \Phi}(y^{(n)}, \varphi) \\ &= f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) f_{Y^{(n)}, \Phi}(y^{(n)}, \varphi), \end{aligned} \quad (5.13)$$

where we recall that the last equality in (5.13) comes from the conditional independence property. Therefore, we can write (5.11) as

$$f_{Y_{n+1}, Y^{(n)}}(u, y^{(n)}) = \int_0^\infty f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) f_{Y^{(n)}, \Phi}(y^{(n)}, \varphi) d\varphi.$$

Substitution into (5.10) yields

$$f_{Y_{n+1}|Y^{(n)}=y^{(n)}}(u | y^{(n)}) = \int_0^\infty \frac{f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) f_{Y^{(n)}, \Phi}(y^{(n)}, \varphi) d\varphi}{f_{Y^{(n)}}(y^{(n)})}, \quad (5.14)$$

for all $y^{(n)} \in \mathbb{R}_+^n$ such that $f_{Y^{(n)}}(y^{(n)}) > 0$. We can now write

$$\frac{f_{Y^{(n)}, \Phi}(y^{(n)}, \varphi)}{f_{Y^{(n)}}(y^{(n)})} = f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}).$$

Substitution into (5.14) yields

$$f_{Y_{n+1}|Y^{(n)}=y^{(n)}}(u | y^{(n)}) = \int_0^\infty f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi. \quad (5.15)$$

Substitution into (5.9) and application of Fubini's theorem (cf. Pitt (1985)[p. 32]) yields

$$\begin{aligned} S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z | y^{(n)}) &= \int_z^\infty \int_0^\infty f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi du \\ &= \int_0^\infty \int_z^\infty f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) du f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi . \end{aligned} \quad (5.16)$$

Note that $f_{Y_{n+1}|\Phi=\varphi}(u | \varphi)$ is the density function of Y_{n+1} given $\Phi = \varphi$. Therefore, we can write

$$\begin{aligned} \tilde{S}_{Y_{n+1}|\Phi=\varphi}(z | \varphi) &= \mathbb{P}[Y_{n+1} \geq z | \Phi = \varphi] \\ &= \int_z^\infty f_{Y_{n+1}|\Phi=\varphi}(u | \varphi) du . \end{aligned}$$

Thus, substitution into (5.16) yields

$$S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z | y^{(n)}) = \int_0^\infty \tilde{S}_{Y_{n+1}|\Phi=\varphi}(z | \varphi) f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi . \quad (5.17)$$

□

As a consequence of the previous lemma, the conditional residual lifetime of Y_{n+1} given $Y^{(n)} = y^{(n)}$, as defined in (5.6), can be characterized as follows.

Corollary 5.1. *In the conditions of Lemma 5.1, for a fixed $\Delta \geq 0$, it follows that*

$$\begin{aligned} R_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z, \Delta | y^{(n)}) &= \frac{S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z + \Delta | y^{(n)})}{S_{Y_{n+1}|Y^{(n)}=y^{(n)}}(z | y^{(n)})} \\ &= \frac{\int_0^\infty \tilde{S}_{Y_{n+1}|\Phi=\varphi}(z + \Delta | \varphi) f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi}{\int_0^\infty \tilde{S}_{Y_{n+1}|\Phi=\varphi}(z | \varphi) f_{\Phi|Y^{(n)}=y^{(n)}}(\varphi | y^{(n)}) d\varphi} , \end{aligned} \quad (5.18)$$

for all $z \geq 0$, $\varphi \geq 0$ and $y^{(n)} \in \mathbb{R}_+^n$ such that the denominator in (5.18) is strictly positive.

Next we describe our release procedure assuming that the stochastic process used to model the fault detection process of a software system has independent times between failures.

5.3 Bayesian release procedure for software reliability growth models with independent times between failures

Let us assume that the fault detection process of a software system can be modelled as a counting process $(N(t))_{t \geq 0}$ depending on a non-negative random parameter Φ . We also assume that the stochastic process defined by the failure times, denoted by $(T_n)_{n \geq 0}$, has independent increments, i.e., the times between failures X_1, X_2, \dots , are independent random variables given $\Phi = \varphi$. Our goal is to certify with high confidence that the next software fault is not occurring within a certain time interval. Thus, we can define the conditional reliability function of X_{n+1} given $\Phi = \varphi$ as follows:

$$\tilde{S}_{X_{n+1}|\Phi=\varphi}(z | \varphi) = \mathbb{P}[X_{n+1} \geq z | \Phi = \varphi] , \quad (5.19)$$

for all $z \geq 0$. Therefore, the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ can be expressed by

$$R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) = \mathbb{P}\left[X_{n+1} \geq z + \Delta \mid X_{n+1} \geq z, X^{(n)} = x^{(n)}\right] . \quad (5.20)$$

If we fix a *reliability level*, denoted by $1 - \delta$, for some $\delta \in (0, 1)$, then our release procedure consists of finding $z_n \geq 0$, which depends on the test history $x^{(n)}$, such that

$$R_{X_{n+1}|X^{(n)}=x^{(n)}}(z_n, \Delta | x^{(n)}) \geq 1 - \delta . \quad (5.21)$$

In all decision-based procedures it is of special interest to study the *risk* taken due to a wrong decision. In this case, it is defined as follows. Our procedure certifies that if no failure has been observed in the interval $(t_n, t_n + z_n]$, for a certain $n \geq 0$, then, with probability at least $1 - \delta$, no failure is observed in the interval $(t_n + z_n, t_n + z_n + \Delta]$, for some $\Delta \geq 0$. In this case, we stop testing at time $t_n + z_n$. Therefore, the risk at that time can be defined as the probability of finding the next fault before time $t_n + z_n + \Delta$. Thus, the *risk after n observed failures* can be defined as

$$G_n(\Delta, z^{(n)}) = \mathbb{P}\left[z_n \leq X_{n+1} \leq z_n + \Delta, \bigcap_{i=1}^n \{X_i \leq z_{i-1}\}\right] . \quad (5.22)$$

Taking the sum over all $n \geq 0$, we define the *global risk* taken in the procedure as

$$G(\Delta) = \sum_{n \geq 0} G_n(\Delta, z^{(n)}) . \quad (5.23)$$

The next theorem shows that if the residual lifetime of X_{n+1} is larger than or equal to $1 - \delta$, then we can keep the global risk under control, i.e., the global risk always smaller than or equal to δ .

Theorem 5.2 (Global risk bound). *For any $\Delta \geq 0$, $\delta \in (0, 1)$ and $n \geq 0$ such that $R_{X_{n+1}|X^{(n)}=x^{(n)}}(z_n, \Delta | x^{(n)}) \geq 1 - \delta$, it follows that $G(\Delta) \leq \delta$.*

Proof. The global risk is given by (5.23). Therefore,

$$G(\Delta) = \sum_{n \geq 0} \mathbb{P} \left[z_n \leq X_{n+1} \leq z_n + \Delta, \bigcap_{i=1}^n \{X_i \leq z_{i-1}\} \right]. \quad (5.24)$$

Application of the Law of Total Probability to (5.24) yields

$$\begin{aligned} G(\Delta) &= \sum_{n \geq 0} \mathbb{E} \left[\mathbb{P} \left[z_n \leq X_{n+1} \leq z_n + \Delta, \bigcap_{i=1}^n \{X_i \leq z_{i-1}\} \mid X^{(n)} \right] \right] \\ &= \sum_{n \geq 0} \mathbb{E} \left[\mathbb{E} \left[I_{\{z_n \leq X_{n+1} \leq z_n + \Delta\}} \cdot I_{\{\bigcap_{i=1}^n \{X_i \leq z_{i-1}\}\}} \mid X^{(n)} \right] \right]. \end{aligned} \quad (5.25)$$

Since only $I_{\{\bigcap_{i=1}^n \{X_i \leq z_{i-1}\}\}}$ belongs to the σ -algebra generated by $X^{(n)}$, we may write (5.25) as

$$\begin{aligned} G(\Delta) &= \sum_{n \geq 0} \mathbb{E} \left[I_{\{\bigcap_{i=1}^n \{X_i \leq z_{i-1}\}\}} \mathbb{E} \left[I_{\{z_n \leq X_{n+1} \leq z_n + \Delta\}} \mid X^{(n)} \right] \right] \\ &= \sum_{n \geq 0} \mathbb{E} \left[I_{\{\bigcap_{i=1}^n \{X_i \leq z_{i-1}\}\}} \mathbb{P} \left[z_n \leq X_{n+1} \leq z_n + \Delta \mid X^{(n)} \right] \right]. \end{aligned} \quad (5.26)$$

Note that $\mathbb{P} [z_n \leq X_{n+1} \leq z_n + \Delta \mid X^{(n)}]$ is a function on the σ -algebra generated by $X^{(n)}$. Thus, it is a function on the sample space Ω . By the properties of conditional expectations (cf. [Grimmett and Stirzaker \(1988\)](#)[Section 3.7, p.67]), the above probability can be expressed as a function of $X^{(n)}$. Therefore, we can write

$$\begin{aligned} \mathbb{P} [z_n \leq X_{n+1} \leq z_n + \Delta \mid X^{(n)}] &= \mathbb{P} [z_n \leq X_{n+1} \leq z_n + \Delta \mid X^{(n)} = x^{(n)}] \\ &= \mathbb{P} [X_{n+1} \leq z_n + \Delta \mid X^{(n)} = x^{(n)}, X_{n+1} \geq z_n] \times \\ &\quad \times \mathbb{P} [X_{n+1} \geq z_n \mid X^{(n)} = x^{(n)}]. \end{aligned} \quad (5.27)$$

Note now that

$$\begin{aligned} \mathbb{P} [X_{n+1} \leq z_n + \Delta \mid X^{(n)} = x^{(n)}, X_{n+1} \geq z_n] &= 1 - R_{X_{n+1} \mid X^{(n)} = x^{(n)}}(z_n, \Delta \mid x^{(n)}) \\ &\leq \delta, \end{aligned} \quad (5.28)$$

and since $\mathbb{P} [X_{n+1} \geq z_n \mid X^{(n)} = x^{(n)}] \leq 1$, for all $z_n \geq 0$, we get that

$$\begin{aligned} G(\Delta) &\leq \delta \sum_{n \geq 0} \mathbb{E} [I_{\{\bigcap_{i=1}^n \{X_i \leq z_{i-1}\}\}}] = \delta \sum_{n \geq 0} \mathbb{P} \left[\bigcap_{i=1}^n \{X_i \leq z_{i-1}\} \right] \\ &= \delta \mathbb{P} \left[\bigcup_{n \geq 0} \left(\bigcap_{i=1}^n \{X_i \leq z_{i-1}\} \right) \right], \end{aligned} \quad (5.29)$$

where the last equality comes from the independent times between failures property. Hence, since the probability of any event is always smaller than or equal to 1, it follows that $G(\Delta) \leq \delta$. \square

5.3.1 Jelinski-Moranda and Goel-Okumoto models

We now consider the cases where the fault detection process of a software system can be described by the Jelinski-Moranda (see Section 2.5.1) and the Goel-Okumoto (see Section 2.6.1) models. We treat the Goel-Okumoto model as a Bayesian case of the Jelinski-Moranda model, i.e., we assume N to be Poisson distributed in the Jelinski-Moranda model to obtain the Goel-Okumoto model (see Section 2.7 for details). The following considerations are also useful for Chapter 6 where we study the performance of our procedure via simulation.

Case 1: N and λ deterministic

Let us first consider the case where both N and λ are known and fixed. Due to the memoryless property of the Exponential distribution, the conditional residual lifetime of X_{n+1} , for all $n = 0, 1, \dots, N$, can be written as

$$R_{X_{n+1}}(z, \Delta) = e^{-(N-n)\lambda\Delta} . \tag{5.30}$$

for all $z \geq 0$ and $\Delta \geq 0$. Since (5.30) is constant for every n , there is no critical value z_n to be computed. Note that only aspect of the test history taken into account here is n , i.e., the number of faults discovered so far. Thus, if we fix a reliability level $1 - \delta$, for some $\delta \in (0, 1)$, and $\Delta \geq 0$, then our procedure stops as soon as (5.30) is larger than or equal to $1 - \delta$. In this case, the number of failures to be observed before we stop testing is given by

$$s = \max \{0, \lceil N + (\log(1 - \delta)/\lambda\Delta) \rceil \} , \tag{5.31}$$

where $\lceil \cdot \rceil$ denotes the ceiling function. Therefore, for any $n \in \mathbb{Z}_+$ such that $n < s$, it follows that (5.30) is smaller than $1 - \delta$. Since $\log(1 - \delta)$ is negative, s can be equal to 0 for small values of Δ . In fact, we have that $s = 0$ if and only if

$$\Delta \leq -\log(1 - \delta)/\lambda N = \Delta_{\min} . \tag{5.32}$$

Note also that in order to have $e^{-(N-n)\lambda\Delta} \geq 1 - \delta$ we must choose Δ in such a way that

$$\Delta \leq -\log(1 - \delta)/(N - n)\lambda , \tag{5.33}$$

for all $n = 0, 1, \dots, N$. In particular, since (5.33) is increasing as a function of n , except for the case where $n = N$ (all faults are discovered) for which $R_{X_{n+1}}(z, \Delta) = 1$ for any $\Delta \geq 0$, the situation with the highest possible Δ corresponds to the case where $n = N - 1$ (one fault left). Therefore, if we define

$$\Delta_{\max} = -\log(1 - \delta)/\lambda , \tag{5.34}$$

then for any $\Delta > \Delta_{\max}$, it follows that (5.30) is always smaller than $1 - \delta$, for all $n \leq N - 1$. Thus, the main conclusion we can extract from here is that the election of Δ must be done carefully, since for values of Δ larger than Δ_{\max} , certification is only possible when all faults have been discovered.

Case 2: N known and fixed, λ Gamma distributed

Suppose now that only N is known and fixed. In this case, N may represent an upper bound for the total number of faults in the system. We assume that λ is a random variable that has a prior Gamma distribution with parameters $k > 0$ and $w > 0$. We have chosen this prior since it is a *conjugate prior* for exponential likelihoods, i.e., the posterior distribution of λ is also a Gamma distribution with parameters $n + k$ and $(\frac{1}{w} + \sum_{i=1}^n (N - i + 1)x_i)^{-1}$ (see e.g. Achcar et al. (1997)). In this case, the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ (cf. Singpurwalla (1991)) has the following expression

$$R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) = \left(\frac{\sum_{i=1}^n (N - i + 1)x_i + \frac{1}{w} + (N - n)z}{\sum_{i=1}^n (N - i + 1)x_i + \frac{1}{w} + (N - n)(z + \Delta)} \right)^{n+k}. \quad (5.35)$$

Note that, unlike Case 1, the residual lifetime depends now also on the test history $x^{(n)}$ and not only on the number of discovered faults. Moreover, we saw in Case 1 that there exists a maximum value of Δ , denoted by Δ_{\max} , such that if we ask for a Δ larger than Δ_{\max} , then certification is only possible when all failures have been observed. This is not the case now since (5.35) depends on z . The conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ has important monotonicity properties as shown in the next lemma.

Lemma 5.2. *The conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$, denoted by $R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)})$, given in (5.35) is monotone increasing with respect to z and monotone decreasing with respect to N .*

Proof. Note that it suffices to study the quantity which is raised to the power $n + k$ in (5.35). Its derivative with respect to z is given by

$$\frac{(N - n)^2 \Delta}{\left(\sum_{i=1}^n (N - i + 1)x_i + \frac{1}{w} + (N - n)(z + \Delta) \right)^2} \quad (5.36)$$

and since all the factors in (5.36) are positive, the residual lifetime is increasing with respect to z . In order to study the monotonicity with respect to N , we treat N as a continuous variable and check whether the derivative with respect to N is larger than 0. In this case, it is convenient to write

$$\begin{aligned} \sum_{i=1}^n (N - i + 1)x_i &= N \sum_{i=1}^n x_i - \sum_{i=1}^n (i - 1)x_i \\ &= N t_n - \sum_{i=1}^n (i - 1)x_i. \end{aligned} \quad (5.37)$$

Substitution into (5.35) yields

$$\begin{aligned} R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) &= \\ &= \left(\frac{(t_n + z)N - \sum_{i=1}^n (i-1)x_i + \frac{1}{w} - nz}{(t_n + z + \Delta)N - \sum_{i=1}^n (i-1)x_i + \frac{1}{w} - n(z + \Delta)} \right)^{n+k}. \end{aligned}$$

Note that also in this case it suffices to study the quantity which is raised to the power $n + k$. The reader can check that the sign of the derivative with respect to N depends only on the sign of the numerator and it is given by

$$w\Delta \left(w \sum_{i=1}^n (i-1)x_i - nwt_n - 1 \right). \quad (5.38)$$

Using (5.37) we can write (5.38) as

$$-w\Delta \left(w \sum_{i=1}^n (n-i+1)x_i + 1 \right), \quad (5.39)$$

which is always smaller than or equal to zero. □

Therefore, since for any $n \geq 0$ the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ is monotone increasing with respect to z , there exists a unique *optimal release time* for our procedure given by

$$z_n = \min \left\{ z \geq 0 : R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) \geq 1 - \delta \right\}. \quad (5.40)$$

Note that we decide to stop testing after the n^{th} failure observation only if the next failure is not observed before z_n . Thus, the number of failures to be observed before we stop testing can be defined as

$$s = \min \{ n \geq 0 : z_n < X_{n+1} \}. \quad (5.41)$$

Although the existence of z_n is proved in Lemma 5.2, it may occur that to wait for it would not be feasible in practice. For example, we may fix a value of Δ big enough to make z_n unrealistically large. This phenomenon will be explained later in Section 6.1.2. On the other hand, decreasing monotonicity with respect to N confirms what one may expect: a system with more faults is less reliable. In this case, to consider a worst case scenario (smallest residual lifetime probability) means to take $N = \infty$. However, for such a case the Jelinski-Moranda model is not defined. This supports the assumption of considering N as an upper bound for the initial number of faults.

Case 3: N Poisson distributed, λ known and fixed (Goel-Okumoto model)

Note that this case has already been described in Section 2.7 as we remind now. Suppose that only λ is known and fixed. A common approach is to assume that N follows a Poisson distribution with finite mean $\theta \geq 0$. Therefore, for all $n_0 = 0, 1, 2, \dots$, we have that

$$\mathbb{P}[N = n_0] = \frac{\theta^{n_0} e^{-\theta}}{n_0!} . \quad (5.42)$$

As mentioned in Section 2.5.1, the Jelinski-Moranda model is a GOS model where the failure times $T_1 < T_2 < \dots < T_n$ are the first n order statistics from a random sample Z_1, \dots, Z_N with common cumulative distribution function $F(t) = 1 - e^{-\lambda t}$, for all $t \geq 0$. As given in (2.26), for GOS models the random variable $N(t)$ conditional on $\{N = n_0\}$ is binomially distributed with parameters n_0 and $F(t)$. As shown in (2.44), the unconditional probability distribution of $N(t)$ follows a Poisson distribution with mean $\theta F(t)$. Thus, we can write

$$\mathbb{P}[N(t) = n] = \frac{(\theta F(t))^n}{n!} e^{-\theta F(t)} . \quad (5.43)$$

As a consequence, the process $(N(t))_{t \geq 0}$ is an NHPP with the following mean-value function

$$\Lambda(t) = \mathbb{E}[N(t)] = \theta(1 - e^{-\lambda t}) , \quad (5.44)$$

for all $t \geq 0$. Note that (5.44) is precisely the mean-value function of the Goel-Okumoto model with parameters θ and λ , as described in Section 2.6.1. Moreover, as given by (2.52), we can compute the posterior probability distribution of N given $T_1 = t_1, \dots, T_n = t_n$, as follows:

$$\mathbb{P}[N = n_0 \mid T_1 = t_1, \dots, T_n = t_n] = \frac{\theta^{n_0 - n}}{(n_0 - n)!} e^{-(\theta e^{-\lambda t_n} + (n_0 - n)\lambda t_n)} . \quad (5.45)$$

In this case, the conditional residual lifetime of X_{n+1} , given $X^{(n)} = x^{(n)}$, is given by

$$R_{X_{n+1} \mid X^{(n)} = x^{(n)}}(z, \Delta \mid x^{(n)}) = e^{-\theta(e^{-(t_n+z)\lambda} - e^{-(t_n+z+\Delta)\lambda})} . \quad (5.46)$$

As shown in Lemma 5.2, the above function is monotone increasing with respect to z . Thus, for some $\delta \in (0, 1)$ fixed, the optimal release time for our procedure is given by

$$z_n = \min \left\{ z \geq 0 : e^{-\theta(e^{-(t_n+z)\lambda} - e^{-(t_n+z+\Delta)\lambda})} \geq 1 - \delta \right\} .$$

Note that when z_n is strictly positive, this is given by

$$z_n = \frac{-1}{\lambda} \log \left(-\frac{\log(1 - \delta)}{\theta(1 - e^{-\lambda \Delta})} \right) - t_n , \quad (5.47)$$

which is increasing and concave as a function of Δ (the first derivative with respect to Δ is positive and the second derivative is negative, for all $\Delta \geq 0$). Moreover, it follows that

$$\lim_{\Delta \rightarrow 0} z_n = 0 , \quad (5.48)$$

and

$$\lim_{\Delta \rightarrow \infty} z_n = \frac{-1}{\lambda} \log \left(-\frac{\log(1 - \delta)}{\theta} \right) - t_n. \quad (5.49)$$

Thus, if we plot z_n as a function of Δ , then we obtain a curve that has also exponential shape with an upper bound given by (5.49), as can be seen in Figure 5.1. This means that for small values of Δ , and provided that $z_n > 0$, the quotient z_n/Δ

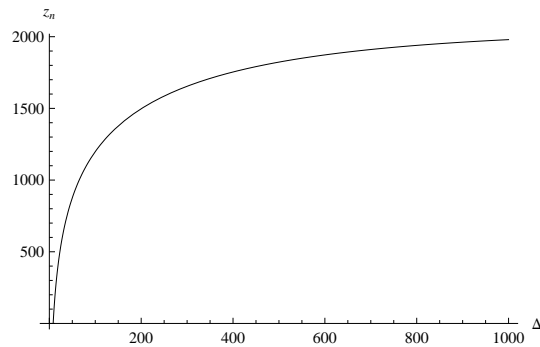


Figure 5.1: Optimal release time z_n as a function of Δ when $\theta = 424.31$, $\lambda = 0.002$ and $t_n = 556$ (results from data set in Section 4.3.1).

is large. Thus, to certify that Δ is a fault-free period, the system has to be observed without fault for a period of time that may be much larger than Δ (as in the case described in Figure 5.1). However, it follows that $z_n/\Delta \rightarrow 0$ when $\Delta \rightarrow \infty$. This means that if the system has been observed for a long enough period with no fault, then the system will eventually survive up to infinity with no failure with probability at least $1 - \delta$. Although z_n exists, it may occur that the value needed to certify Δ is too large to be considered as a realistic value. This will depend on the scale of the data of the problem at hand (which is determined by the parameter θ). We will study this in details in Section 6.1.3. Note finally that, for a fixed $\Delta > 0$, it follows that z_n is a decreasing function of n . This means that, as long as more faults are discovered, $z_n/\Delta \rightarrow 0$ more rapidly. Thus, less effort is required to certify Δ as fault-free interval.

Case 4: N Poisson and λ Gamma distributed (full Bayesian approach)

We finish the study of our certification procedure for the Jelinski-Moranda model by considering that both N and λ are random variables. According to the notation introduced in Section 2.3.1, we denote by n_0 and λ_0 the realization of the random variables N and λ , respectively. We assume that N has a prior Poisson distribution with parameter $\theta \geq 0$ and λ has a prior Gamma distribution with parameters $k > 0$ and $w > 0$. These are typical choices for prior distributions in this case (although not unique). However, we are not interested in the Bayesian analysis itself. For that reason, we do not study the performance of the procedure for different choices of

prior distributions. For details on this we refer to [Achcar \(1997\)](#) and [Kuo and Yang \(1996\)](#). Note that assuming prior independence, the joint posterior distribution of N and λ , denoted by $f_{N,\lambda}(n_0, \lambda_0)$, is given by

$$f_{N,\lambda}(n_0, \lambda_0) = \frac{\frac{\theta^{n_0}}{(n_0 - n)!} \lambda_0^{n+k-1} e^{-\left(\frac{1}{w} + \sum_{i=1}^n (n_0 - i + 1)x_i\right) \lambda_0}}{\int_0^\infty \sum_{n_0=n+1}^\infty \frac{\theta^{n_0}}{(n_0 - n)!} \lambda_0^{n+k-1} e^{-\left(\frac{1}{w} + \sum_{i=1}^n (n_0 - i + 1)x_i\right) \lambda_0} d\lambda_0} . \quad (5.50)$$

In this case, the residual lifetime probability of X_{n+1} does not have a simple expression like in (5.35) where N was assumed to be known. However, it can be computed as follows. First note that the conditional reliability function of X_{n+1} , given $N = n_0$ and $\lambda = \lambda_0$, is given by

$$\tilde{S}_{X_{n+1}|N=n_0, \lambda=\lambda_0}(z | n_0, \lambda_0) = e^{-(n_0 - n)\lambda_0 z} . \quad (5.51)$$

Thus, if we define

$$A_n(z, \Delta) = \sum_{i=1}^n x_i + z + \Delta ,$$

and

$$B_n(z, \Delta) = \frac{1}{w} - \sum_{i=1}^n (i - 1)x_i - (z + \Delta)n ,$$

then the conditional reliability function of X_{n+1} given $X^{(n)} = x^{(n)}$, as given by Lemma 5.1, can be written as

$$\begin{aligned} S_{X_{n+1}|X^{(n)}=x^{(n)}}(z | x^{(n)}) &= \\ &= \int_0^\infty \lambda_0^{n+k-1} \sum_{n_0=n+1}^\infty \frac{\theta^{n_0}}{(n_0 - n)!} e^{-(A_n(z,0)n_0 + B_n(z,0))\lambda_0} d\lambda_0 . \end{aligned} \quad (5.52)$$

First note that

$$\begin{aligned} \sum_{n_0=n+1}^\infty \frac{\theta^{n_0}}{(n_0 - n)!} e^{-(A_n(z,0)n_0 + B_n(z,0))\lambda_0} &= \\ &= \theta^n e^{-(A_n(z,0)n + B_n(z,0))\lambda_0} \left(e^{\theta e^{-\lambda_0 A_n(z,0)}} - 1 \right) . \end{aligned} \quad (5.53)$$

Moreover, it follows that

$$\begin{aligned} A_n(z, 0)n + B_n(z, 0) &= \frac{1}{w} + \sum_{i=1}^n (n - i + 1)x_i , \\ &= A_n(z, \Delta)n + B_n(z, \Delta) . \end{aligned} \quad (5.54)$$

Since (5.54) does not depend on z , we will denote it by C_n . Substitution into (5.52) yields

$$S_{X_{n+1}|X^{(n)}=x^{(n)}}(z | x^{(n)}) = \theta^n \int_0^\infty \lambda_0^{n+k-1} e^{-\lambda_0 C_n} \left(e^{\theta e^{-\lambda_0 A_n(z,0)}} - 1 \right) d\lambda_0. \quad (5.55)$$

Thus, we can use Lemma 5.1 to write the residual lifetime probability of X_{n+1} given $X^{(n)} = x^{(n)}$ as follows:

$$R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) = \frac{\int_0^\infty \lambda_0^{n+k-1} e^{-\lambda_0 C_n} \left(e^{\theta e^{-\lambda_0 A_n(z,\Delta)}} - 1 \right) d\lambda_0}{\int_0^\infty \lambda_0^{n+k-1} e^{-\lambda_0 C_n} \left(e^{\theta e^{-\lambda_0 A_n(z,0)}} - 1 \right) d\lambda_0}. \quad (5.56)$$

In order to define a local optimal release time we need to prove that the residual lifetime probability is a monotone increasing function of z . The next lemma shows the desired result.

Lemma 5.3. *The conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$, denoted by $R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)})$, given in (5.56) is monotone increasing with respect to z .*

Proof. Let us first define the following functions

$$g(\lambda_0) = \lambda_0^{n+k-1} e^{-\lambda_0 C_n}, \quad (5.57)$$

$$f(\lambda_0, z) = \left(e^{\theta e^{-\lambda_0 A_n(z,\Delta)}} - 1 \right), \quad (5.58)$$

and

$$h(\lambda_0, z) = \left(e^{\theta e^{-\lambda_0 A_n(z,0)}} - 1 \right). \quad (5.59)$$

In this case, we can write the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ as

$$R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) = \frac{\int_0^\infty g(\lambda_0) f(\lambda_0, z) d\lambda_0}{\int_0^\infty g(\lambda_0) h(\lambda_0, z) d\lambda_0}. \quad (5.60)$$

Since (5.60) is a quotient, we only need to study the sign of the numerator of its derivative with respect to z to determine its monotonicity. Thus, the condition we need to check is

$$\begin{aligned} & \int_0^\infty g(\lambda_0) f'(\lambda_0, z) d\lambda_0 \int_0^\infty g(\lambda_0) h(\lambda_0, z) d\lambda_0 \\ & - \int_0^\infty g(\lambda_0) h'(\lambda_0, z) d\lambda_0 \int_0^\infty g(\lambda_0) f(\lambda_0, z) d\lambda_0 \geq 0. \end{aligned} \quad (5.61)$$

First note that (5.61) is equivalent to

$$\frac{\int_0^\infty g(\lambda_0) f'(\lambda_0, z) d\lambda_0}{\int_0^\infty g(\lambda_0) h'(\lambda_0, z) d\lambda_0} \geq \frac{\int_0^\infty g(\lambda_0) f(\lambda_0, z) d\lambda_0}{\int_0^\infty g(\lambda_0) h(\lambda_0, z) d\lambda_0}. \quad (5.62)$$

Note also that the right-hand side of (5.62) is precisely the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ as given in (5.60). Therefore, it takes values between 0 and 1. Thus, if we prove that

$$\frac{\int_0^\infty g(\lambda_0) f'(\lambda_0, z) d\lambda_0}{\int_0^\infty g(\lambda_0) h'(\lambda_0, z) d\lambda_0} \geq 1, \quad (5.63)$$

then (5.62) is true. In this case, if $f'(\lambda_0, z) \geq h'(\lambda_0, z)$, for all $\lambda_0 > 0$, then (5.63) holds. If we compute $f'(\lambda_0, z)$ and $h'(\lambda_0, z)$, then we get that (5.63) is true if and only if

$$\theta \left(e^{-\lambda_0 A_n(z, 0)} - e^{-\lambda_0 A_n(z, \Delta)} \right) + \lambda_0 \Delta \geq 0, \quad (5.64)$$

for all $\lambda_0 > 0$. Since $A_n(z, \Delta)$ is increasing in Δ , and θ , λ_0 and Δ are all non-negative, it follows that (5.64) is true. Hence, the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ is monotone increasing with respect to z . \square

Therefore, there exists a unique $z_n \geq 0$ that is an optimal release time for our procedure. Similar remarks about the scale of z_n as the ones made in the previous cases are also valid here. Therefore, we refer to them for details.

5.3.2 Run model

In this section we extend the work in [Di Bucchianico et al. \(2008\)](#) with some results regarding the certification procedure previously introduced in this chapter. As done with the Jelinski-Moranda model in the previous section, we now discuss some issues that need to be considered when the fault detection process of a software system can be described by the Run model, as we called it in Section 5.1.2. For this model it is assumed that the initial number of faults in the system, denoted by N , is unknown. As defined in Section 5.1.2, the random variable X_i , for all $i = 1, \dots, N$, represents the number of test runs needed to find the i^{th} software fault after repairing the $(i-1)^{\text{st}}$ fault and follows a geometric distribution with success parameter

$$p_i = 1 - (1 - \theta)^{N-i+1}. \quad (5.65)$$

Thus, we have a model with two parameters, namely N and θ , where the probability distribution of the times between failures is given by

$$\begin{aligned} \mathbb{P}[X_i = x_i] &= (1 - p_i)^{x_i-1} p_i \\ &= (1 - \theta)^{(x_i-1)(N-i+1)} (1 - (1 - \theta)^{N-i+1}), \end{aligned} \quad (5.66)$$

and

$$\begin{aligned}\mathbb{P}[X_i > x_i] &= (1 - p_i)^{x_i} \\ &= (1 - \theta)^{(x_i - 1)(N - i + 1)},\end{aligned}\tag{5.67}$$

for all $x_i \geq 1$. Note that, unlike in Section 5.1.2, we are not interested in truncated testing but in the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$, for some $\Delta \geq 0$. This is given by

$$\begin{aligned}R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) &= \mathbb{P}[X_{n+1} > z + \Delta | X_{n+1} > z, X^{(n)} = x^{(n)}] \\ &= \frac{\mathbb{P}[X_{n+1} > z + \Delta | X^{(n)} = x^{(n)}]}{\mathbb{P}[X_{n+1} > z | X^{(n)} = x^{(n)}]}.\end{aligned}\tag{5.68}$$

Depending on the nature of the parameters of the model (deterministic or random) we consider the following cases.

Case 1: N and θ deterministic

Let us first assume that both N and θ are known and fixed. Due to the memoryless property of the geometric distribution, the conditional residual lifetime of X_{n+1} , for all $n = 0, \dots, N$ and $\Delta \geq 0$, can be written as follows:

$$R_{X_{n+1}}(z, \Delta) = (1 - \theta)^{(N - n)\Delta},\tag{5.69}$$

for all $z \geq 0$. Note that the above probability is the same as the one obtained for the Jelinski-Moranda model in (5.31) if we take the parameter λ in the Jelinski-Moranda model to be equal to $-\log(1 - \theta)$. Thus, the same considerations made in Section 5.3.1 apply here.

Case 2: N Poisson distributed, θ known and fixed

Suppose now that only θ is known and fixed and N follows a Poisson distribution with finite mean $\lambda > 0$. Therefore, the prior probability mass function of N is given by

$$\mathbb{P}[N = n_0] = \frac{\lambda^{n_0} e^{-\lambda}}{n_0!},\tag{5.70}$$

for all $n_0 = 0, 1, 2, \dots$. In order to compute the conditional residual lifetime of X_{n+1} , given $X^{(n)} = x^{(n)}$, as defined in (5.68), we need to calculate the conditional reliability function of X_{n+1} as follows:

$$\begin{aligned}&\mathbb{P}[X_{n+1} > z | X^{(n)} = x^{(n)}] \\ &= \sum_{n_0=n+1}^{\infty} \mathbb{P}[X_{n+1} > z | N = n_0] \mathbb{P}[N = n_0 | X^{(n)} = x^{(n)}].\end{aligned}\tag{5.71}$$

Note that, conditional on $\{N = n_0\}$, the random variable X_{n+1} is geometrically distributed. Thus, we can use (5.67) to compute $\mathbb{P}[X_{n+1} > z \mid N = n_0]$. Therefore, we only need to compute the posterior distribution of N in order to calculate the conditional reliability function of X_{n+1} in (5.71). Since the random variables X_i are independent, given $N = n_0$, for all $i = 1, \dots, N$, we can use (5.66) to compute the likelihood function of X_1, \dots, X_n , as follows:

$$\begin{aligned} L(x_1, \dots, x_n; n_0, \theta) &= \prod_{i=1}^n \mathbb{P}[X_i = x_i \mid N = n_0] \\ &= \prod_{i=1}^n (1 - \theta)^{(x_i-1)(n_0-i+1)} (1 - (1 - \theta)^{n_0-i+1}) \\ &= (1 - \theta)^{\sum_{i=1}^n (x_i-1)(n_0-i+1)} \prod_{i=1}^n (1 - (1 - \theta)^{n_0-i+1}) . \end{aligned} \quad (5.72)$$

Products of the form of the one on the right-hand side of (5.72) can be expressed in terms of *q-Pochhammer symbols* (see Koepf (1998)[p. 25] for details). In general, the *q-Pochhammer symbol*, denoted by $[a; q]_k$, for any integer $k \geq 1$, is defined as

$$[a; q]_k = \prod_{i=1}^{k-1} (1 - aq^i) . \quad (5.73)$$

In this case, it follows that

$$\prod_{i=1}^n (1 - (1 - \theta)^{n_0-i+1}) = \frac{[(1 - \theta)^{n_0+1}; (1 - \theta)^{-1}]_{n+1}}{1 - (1 - \theta)^{n_0+1}} . \quad (5.74)$$

Moreover, if we define

$$D_n(n_0) = \sum_{i=1}^n (x_i - 1)(n_0 - i + 1) , \quad (5.75)$$

then we can write (5.72) in a more compact way as follows:

$$L(x_1, \dots, x_n; n_0, \theta) = \frac{(1 - \theta)^{D_n(n_0)}}{1 - (1 - \theta)^{n_0+1}} [(1 - \theta)^{n_0+1}; (1 - \theta)^{-1}]_{n+1} . \quad (5.76)$$

Therefore, we can use (5.70), (5.76) and Bayes theorem to compute the posterior probability mass function of N . Note that the denominator in Bayes formula does not depend on n_0 nor on θ . Thus, for sake of brevity, in the remainder of this section we will always denote it by C . In this case, we can write the posterior probability mass function of N as follows:

$$\mathbb{P}[N = n_0 \mid X^{(n)} = x^{(n)}] = \frac{1}{C} \frac{\lambda^{n_0}}{n_0!} \frac{(1 - \theta)^{D_n(n_0)}}{1 - (1 - \theta)^{n_0+1}} [(1 - \theta)^{n_0+1}; (1 - \theta)^{-1}]_{n+1} . \quad (5.77)$$

Note that, if $m = \sum_{i=1}^n x_i$ is the number of observed failures, then we can use (5.77) to compute $\mathbb{P}[N > m \mid X^{(n)} = x^{(n)}]$, i.e., the probability of having remaining faults in the system. Moreover, using (5.67) and (5.77), we can write the conditional reliability function of X_{n+1} given $X^{(n)} = x^{(n)}$, as defined in (5.71), as follows:

$$\mathbb{P}[X_{n+1} > z \mid X^{(n)} = x^{(n)}] = \frac{1}{C} \sum_{n_0=n+1}^{\infty} \frac{\lambda^{n_0} (1-\theta)^{D_n(n_0)+(n_0-n)z}}{n_0! 1 - (1-\theta)^{n_0+1}} \times \quad (5.78)$$

$$\times [(1-\theta)^{n_0+1}; (1-\theta)^{-1}]_{n+1} .$$

Finally, the conditional residual lifetime of X_{n+1} , given $X^{(n)} = x^{(n)}$, as defined in (5.68), can be computed using (5.78). If we prove that this is monotone increasing with respect to z , then there exists a local optimal release time for our procedure. The proof in this case follows the same steps as the proof of Lemma 5.3. For that reason we skip it here. The reader can check that the derivative of the conditional residual lifetime of X_{n+1} with respect to z is positive if and only if

$$-(n_0 - n) \log(1 - \theta) \left((1 - \theta)^{(n_0 - n)z} - (1 - \theta)^{(n_0 - n)(z + \Delta)} \right) \geq 0 . \quad (5.79)$$

Since $\theta \in (0, 1)$, the function $(1 - \theta)^y$ is decreasing in y , for all $y \geq 0$. Thus, it follows that $((1 - \theta)^{(n_0 - n)z} - (1 - \theta)^{(n_0 - n)(z + \Delta)}) \geq 0$. Moreover, since $n_0 \geq n$ and $\log(1 - \theta) \leq 0$, it follows that (5.79) always holds.

Case 3: N known and fixed, θ Beta distributed

We assume now that only N is known and fixed. Like in the Jelinski-Moranda case, we can think of N as an upper bound for the total number of faults in the system. We also assume that θ is a random variable that has a prior Beta distribution with parameters $\alpha > 0$ and $\beta > 0$. The choice of this prior distribution is well-known in Bayesian statistics. Note that it is suitable for θ since the Beta distribution has support on $(0, 1)$. Moreover, its flexibility (two-parameter distribution) can describe many different types of situations. Therefore, we can write the prior density function of θ as follows:

$$f_{\theta}(\theta_0) = \frac{(1 - \theta_0)^{\beta-1} \theta_0^{\alpha-1}}{B(\alpha, \beta)} , \quad (5.80)$$

where $B(\alpha, \beta) = \int_0^1 (1 - \theta_0)^{\beta-1} \theta_0^{\alpha-1} d\theta_0$, denotes the Euler Beta function. The posterior distribution of θ can be computed by using the likelihood function of X_1, \dots, X_n given in (5.76), but replacing θ by θ_0 and n_0 by N . Thus, if C denotes the denominator in Bayes formula, then the posterior density function of θ is given by

$$f_{\theta \mid X^{(n)}=x^{(n)}}(\theta_0 \mid x^{(n)}) = \frac{1}{C} \frac{\theta_0^{\alpha-1} (1 - \theta_0)^{\beta-1 + D_n(N)}}{1 - (1 - \theta_0)^{N+1}} [(1 - \theta_0)^{N+1}; (1 - \theta_0)^{-1}]_{n+1} . \quad (5.81)$$

Thus, using (5.67), (5.75) and (5.81), we can write the conditional reliability function of X_{n+1} given $X^{(n)} = x^{(n)}$, as follows:

$$\begin{aligned} \mathbb{P} \left[X_{n+1} > z \mid X^{(n)} = x^{(n)} \right] &= \frac{1}{C} \int_0^1 \frac{\theta_0^{\alpha-1} (1-\theta_0)^{\beta-1+D_n(N)+(N-n)z}}{1 - (1-\theta_0)^{N+1}} \times \\ &\quad \times [(1-\theta_0)^{N+1}; (1-\theta_0)^{-1}]_{n+1} d\theta_0 . \end{aligned} \quad (5.82)$$

Finally, the conditional residual lifetime of X_{n+1} , given $X^{(n)} = x^{(n)}$, as defined in (5.68), can be computed using (5.82). As done with previous cases, if we prove that the residual lifetime probability of X_{n+1} is monotone increasing with respect to z , then there exists a local optimal release time for our procedure. Monotonicity in this case can be proved in a similar way as it was done in Case 2. In fact, the condition to be checked for the sign of the derivative is the same given in (5.79), but replacing θ by θ_0 and n_0 by N . This is not surprising since in both cases the residual lifetime probability of X_{n+1} depends on z only through the survival probability of a geometric random variable as given in (5.67).

Case 4: N Poisson and θ Beta distributed (full Bayesian approach)

We finish the study of our certification procedure for the Run model by considering that both N and θ are random variables. We assume that N has a prior Poisson distribution with parameter $\lambda \geq 0$ and θ has a prior Beta distribution with parameters $\alpha > 0$ and $\beta > 0$. Note that assuming prior independence, the joint posterior distribution of N and θ is given by

$$f_{N,\theta}(n_0, \theta_0) = \frac{1}{C} \frac{\lambda^{n_0} \theta_0^{\alpha-1} (1-\theta_0)^{\beta-1+D_n(n_0)}}{n_0! 1 - (1-\theta_0)^{n_0+1}} [(1-\theta_0)^{n_0+1}; (1-\theta_0)^{-1}]_{n+1} , \quad (5.83)$$

where C denotes, as above, the denominator in Bayes formula. Note that we can use (5.67), (5.75) and (5.83) to write the conditional reliability function of X_{n+1} given $X^{(n)} = x^{(n)}$, as follows:

$$\begin{aligned} \mathbb{P} \left[X_{n+1} > z \mid X^{(n)} = x^{(n)} \right] &= \frac{1}{C} \sum_{n_0=n+1}^{\infty} \int_0^1 \frac{\lambda^{n_0} \theta_0^{\alpha-1} (1-\theta_0)^{\beta-1+D_n(n_0)+(n_0-n)z}}{n_0! 1 - (1-\theta_0)^{n_0+1}} \times \\ &\quad \times [(1-\theta_0)^{n_0+1}; (1-\theta_0)^{-1}]_{n+1} d\theta_0 . \end{aligned} \quad (5.84)$$

Thus, the conditional residual lifetime of X_{n+1} , given $X^{(n)} = x^{(n)}$, as defined in (5.68), can be computed using (5.84). In a similar way as it was done in the previous cases, we can prove that the residual lifetime probability of X_{n+1} is monotone increasing with respect to z . Therefore, also in this case there exists a local optimal release time for our procedure. The proof of monotonicity here is left to the reader since it follows the same steps as in the previous cases.

CHAPTER 6

PERFORMANCE OF THE CERTIFICATION PROCEDURE

In Section 5.3 we have presented a sequential software release procedure that certifies (with some statistical confidence) that if we decide to stop testing, then the next software fault is not occurring within a certain time interval. This decision is based on the fault-free interval since the last failure observation and the test history. In this chapter, we study the performance of our procedure via simulation. We consider the same models used in Section 5.3, namely the Jelinski-Moranda and the Run model. Different cases are studied depending on the nature (random or deterministic) of the parameters of the models. Thus, as explained in Section 5.3.1, we also consider the Goel-Okumoto model as a special case of the Jelinski-Moranda model, where the total number of faults in the system is considered to be a Poisson random variable.

6.1 Jelinski-Moranda model

The Jelinski-Moranda model is usually characterized by the density function of the times between failures given in (2.33). Following the steps of Section 5.3.1, we first consider the case where both parameters of the model are assumed to be known and fixed. Next we consider the cases where only one parameter is assumed to be a random variable and finally we consider the case where both parameters are treated as random variables.

6.1.1 Case 1: N and λ deterministic

In Section 5.3.1 (Case 1), we defined the number of failures to be observed before we stop testing as

$$s = \max \{0, \lceil N + (\log(1 - \delta)/\lambda\Delta) \rceil \} , \quad (6.1)$$

where $\lceil \cdot \rceil$ denotes the ceiling function, N and λ are the parameters of the Jelinski-Moranda model, $1 - \delta$ is the reliability level of the procedure and Δ is the time to be certified as fault-free. We have computed s/N for different values of Δ , when $\lambda = 0.0005$ and $1 - \delta = 0.90$. The results are shown in Table 6.1. As one may expect, for a fixed value of Δ , the number of faults that has to be discovered to certify, with probability at least 0.90, that the next software fault will not occurred before Δ , increases with both N and Δ . We also defined in Section 5.3.1 (Case 1) the quantities

$$\Delta_{\min} = -\log(1 - \delta)/\lambda N$$

and

$$\Delta_{\max} = -\log(1 - \delta)/\lambda .$$

	$\Delta = 20$	$\Delta = 25$	$\Delta = 50$	$\Delta = 100$	$\Delta = 150$
$N = 10$	0.00	0.20	0.60	0.80	0.90
$N = 20$	0.50	0.60	0.80	0.90	0.95
$N = 50$	0.80	0.84	0.92	0.96	0.98
$N = 100$	0.90	0.92	0.96	0.98	0.99

Table 6.1: Mean percentage of failures to be observed to reach a reliability level larger than or equal to 0.90 for different values of N and Δ when $\lambda = 0.0005$.

As shown there, for any $\Delta < \Delta_{\min}$ it follows that $s = 0$, and for any $\Delta > \Delta_{\max}$ certification is only possible when all faults are discovered, i.e., when $s = N$. Table 6.2 shows several values of Δ_{\max} and Δ_{\min} , for different values of λ , when $N = 20$ and $1 - \delta = 0.90$. We can observe that the larger the value of λ , the smaller the value of

	$\lambda = 0.00025$	$\lambda = 0.0005$	$\lambda = 0.001$	$\lambda = 0.002$	$\lambda = 0.004$
Δ_{\max}	421.44	210.72	105.36	52.68	26.34
Δ_{\min}	21.07	10.53	5.26	2.63	1.31

Table 6.2: Some values of Δ_{\max} and Δ_{\min} when $N = 20$ and $1 - \delta = 0.90$ for different values of λ .

both Δ_{\max} and Δ_{\min} . This is not surprising since for the Jelinski-Moranda model, the hazard rate of the next failure occurrence is proportional to λ . Thus, if this is large, then faults occur more frequently. This also means that the scale of failure observations, and therefore of Δ_{\max} and Δ_{\min} , depends only on λ , independently of the number of faults in the system. However, the more faults we have, the more difficult it is to certify a certain value of Δ (as it is shown in Table 6.1) since in this case the times between failures are smaller.

6.1.2 Case 2: N known and fixed, λ Gamma distributed

We now consider the case where only N is fixed and λ is a random variable that has a prior Gamma distribution with parameters $k > 0$ and $w > 0$. This case was studied in Section 5.3.1 (Case 2). Conditional on $\{X^{(n)} = x^{(n)}\}$, the residual lifetime of X_{n+1} is given by

$$R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) = \left(\frac{\sum_{i=1}^n (N - i + 1)x_i + \frac{1}{w} + (N - n)z}{\sum_{i=1}^n (N - i + 1)x_i + \frac{1}{w} + (N - n)(z + \Delta)} \right)^{n+k}, \quad (6.2)$$

for any $\Delta \geq 0$. We saw in Section 6.1.1 that, for λ and N fixed, there exists a maximum value of Δ , denoted by Δ_{\max} , such that if we ask for a Δ larger than Δ_{\max} , then certification is only possible when all failures have been observed. This is not the case now since (6.2) depends on z . Thus, in theory, any finite value of Δ can be certified by letting z go to infinity. We proved in Lemma 5.2 that (6.2)

is monotone increasing with respect to z . Therefore, there exists a unique optimal release time $z_n \geq 0$ for our procedure given by

$$z_n = \min \left\{ z \geq 0 : R_{X_{n+1}|X^{(n)}=x^{(n)}}(z, \Delta | x^{(n)}) \geq 1 - \delta \right\} . \quad (6.3)$$

Moreover, the number of failures to be observed before we stop testing is given by

$$s = \min \{ n \geq 0 : z_n < X_{n+1} \} . \quad (6.4)$$

Suppose now that we simulate N failure times from a Jelinski-Moranda model. If we perform $r \geq 1$ simulations, then we compute the following metrics. We know that, for each simulation, the optimal release time for our procedure is given by (6.3). We denote this by $z_{j,n}$, for all $j = 1, \dots, r$ and $n = 0, 1, \dots, N$. Thus, the *mean release time after the n^{th} failure observation* is given by

$$\bar{z}_n = \frac{1}{r} \sum_{j=1}^r z_{j,n} \quad (6.5)$$

and the *overall mean release time* is

$$\bar{z} = \frac{1}{N} \sum_{n=0}^N \bar{z}_n . \quad (6.6)$$

We also know that, for each simulation, the number of failures to be observed before we stop testing is given by (6.4). We denote this by s_j , for all $j = 1, \dots, r$. Note that the *true* release time for each simulation is z_{s_j} . Thus, the *mean number of failures to be observed before we stop testing* is given by

$$\bar{s} = \frac{1}{r} \sum_{j=1}^r s_j \quad (6.7)$$

and the *mean true release time* is

$$\bar{z}_s = \frac{1}{r} \sum_{j=1}^r z_{s_j} . \quad (6.8)$$

Finally, note that, for each simulation, the total time on test (*TTT*) is given by $TTT_j = T_{s_j} + z_{s_j}$. Thus, the *mean total time on test* is

$$\overline{TTT} = \frac{1}{r} \sum_{j=1}^r TTT_j . \quad (6.9)$$

Table 6.3 illustrates how these metrics are computed. Besides the above defined metrics, we are also interested in the number of times that our procedure yields a *correct* result. For us a correct result means that, if our procedure stops after observing the n^{th} failure, for some $n > 0$, then $X_{n+1} \geq z_n + \Delta$. This (averaged over

Simulation	Release time				Failures stop	True release time	Total time test
1	$z_{1,0}$	$z_{1,1}$	\dots	$z_{1,N}$	s_1	z_{s_1}	TTT_1
2	$z_{2,0}$	$z_{2,1}$	\dots	$z_{2,N}$	s_2	z_{s_2}	TTT_2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
r	$z_{r,0}$	$z_{r,1}$	\dots	$z_{r,N}$	s_r	z_{s_r}	TTT_r
Mean	\bar{z}_0	\bar{z}_1	\dots	\bar{z}_N	\bar{s}	\bar{z}_s	\overline{TTT}

Table 6.3: Main metrics: release time, number of failures to stop, true release time and total time on test.

Notation	Definition
\bar{z}_n	mean release time after the n^{th} failure observation
\bar{z}	overall mean release time
\bar{s}	mean number of failures to stop testing
\bar{z}_s	mean true release time
\overline{TTT}	mean total time on test
$\tilde{\delta}$	observed reliability

Table 6.4: Notation used in this chapter.

the number of simulations r) is called the *observed reliability* and it is denoted by $\tilde{\delta}$. This notation will be used not only in this section but also in the remainder of this chapter. For that reason, it is summarized in Table 6.4 so that the reader may refer back to this table should confusion about notation arise. The default settings for the simulations carried out in this section can be seen in Table 6.5. For the results

Number of failures:	$N = 20$
Reliability level:	$1 - \delta = 0.90$
Prior distribution of λ :	Gamma(5,0.0015)
Number of simulations:	$r = 100$

Table 6.5: Default settings for simulations.

below we will specify only the values that differ from the default ones. First note that the choice of the prior distribution determines the scale of the times between failures in the problem (small values of λ produce large times between failures). In

this case, the sample mean of the times between failures that have been simulated is equal to $\bar{x} = 22.57$. However, the choice of the prior distribution does not have great effect in the posterior distribution as long as the posterior distribution is updated with new failure observations. This can be seen in Figure 6.1 where we have plotted the prior distribution (dashed) and the evolution of the posterior distribution of λ after a few updates. Note how the posterior distribution becomes sharper near

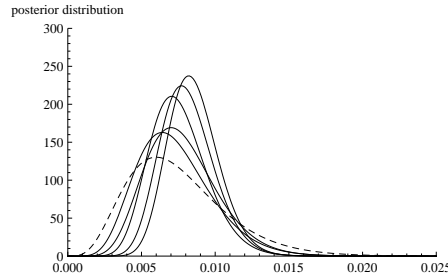


Figure 6.1: Prior (dashed) and posterior distribution of λ .

0.0075 (the prior mean of λ) as long as new data is observed and a new posterior distribution is calculated. Table 6.6 shows some results for several values of Δ . We

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\Delta = 1$	0.20	9.77	0.92	3.60	36.68
$\Delta = 3$	0.74	2.92	0.93	130.36	188.39
$\Delta = 5$	0.83	1.40	0.92	304.15	287.50
$\Delta = 10$	0.93	0.79	0.94	887.70	391.76

Table 6.6: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of Δ .

can conclude from Table 6.6 that choosing a large Δ is *expensive*. Increasing Δ produces an increase of the mean percentage of failures to be observed before we stop testing (\bar{s}/N) and on the mean total time of test (\overline{TTT}). This behaviour is not surprising as we already explained in the deterministic case. For small values of Δ we have that \bar{z}_s/Δ is large. Thus, to certify Δ as fault-free, the system has to be observed without fault for a period of time that is much larger than Δ . Therefore, a relationship between \bar{z}_s and Δ like in Figure 5.1 is expected. Note also that for all the values of Δ , the procedure reaches the desired reliability ($\tilde{\delta}$). The increase of the value of \bar{z} with Δ has an important interpretation as we will see now. Although it is possible to provide a theoretical solution for our problem, i.e., there always exists a value of z_n given by (6.3), this is may be useless in practice since the value of z_n is often extremely large with respect to the scale of the data to consider it as realistic. This can be observed in Figure 6.2 where we have plotted the sample mean

of the simulated times between failures, denoted by \bar{x}_n , and the sample mean of the release times given by our stopping rule, denoted by \bar{z}_n , when $\Delta = 10$. Note that

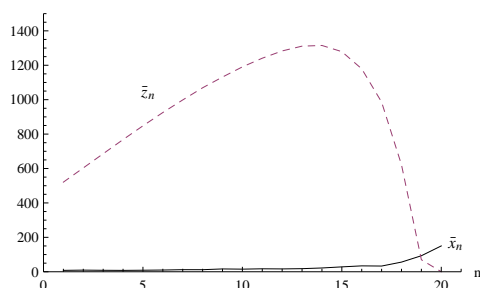


Figure 6.2: Mean release times (dashed) and mean times between failures (solid) as a function of n , when $\Delta = 10$.

the difference of magnitude between the release times and the times between failures is very large, making thus almost any value of z_n unfeasible in practice. Only when a sufficiently large number of faults has been discovered (at later stages of testing) the value of \bar{z}_n approaches the values of the times between failures. It is only at this stage when we can use the release times in practice, i.e., when the curves in Figure 6.2 intersect, testing can be stopped. Note that this corresponds to the value of \bar{s}/N in Table 6.6 for $\Delta = 10$. Figure 6.2 also shows the typical behaviour of z_n . In early stages of testing, where only a few faults have been discovered and repaired, these are increasing and after a certain number of faults have been removed from the system they start to decrease to eventually become 0. Note that if $z_n = 0$, for a certain $n > 0$, then our procedure decides to stop testing after the $(n - 1)^{\text{st}}$ fault detection. The behaviour of z_n described above is not completely intuitive since one may expect that as long as the system is repaired and improved the time needed to certify a fixed fault-free interval becomes smaller. Figure 6.3 can explain this. On its left-hand side we observe the residual lifetime probability as a function of z for $n = 1$, $n = 5$, $n = 10$ and $n = 15$. The reliability level has been fixed to $1 - \delta = 0.90$ (horizontal dashed line). Thus, for each n , the optimal release time for the procedure corresponds to the intersection of the residual lifetime probability and the reliability level. In this case we can see that $z_1 < z_5 < z_{10} < z_{15}$. On the right-hand side we observe the residual lifetime probability for $n = 15$, $n = 16$, $n = 17$, and $n = 18$. In this case, we see that z_n decreases for such values of n .

A system containing a large number of faults requires more effort to certify that the next fault will not occurred before Δ . This will be reflected in an increase of both the number of failures to be observed before we stop testing and the total time of test. Note that the optimal release time should also increase in this case, since the more faults the system has the longer we should wait without observing a failure to decide that the next one is not occurring before Δ . This can be observed in Table 6.7, where the usual metrics (see Table 6.4) for different values of N , when

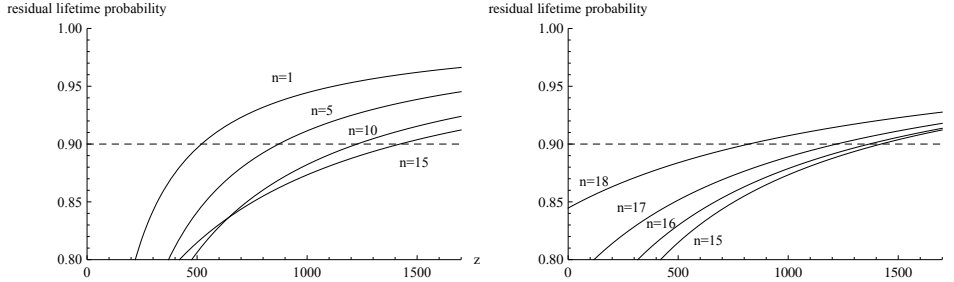


Figure 6.3: Left: residual lifetime probability for $n = 1, n = 5, n = 10$ and $n = 15$: z_n increases with respect to n . Right: residual lifetime probability for $n = 15, n = 16, n = 17$, and $n = 18$: z_n decreases with respect to n .

$\Delta = 10$, are shown. We can observe that, in fact, \bar{s}, \bar{z} and \overline{TTT} increase with N .

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$N = 5$	0.74	0.99	0.90	179.23	179.43
$N = 10$	0.66	2.73	0.96	430.07	287.81
$N = 15$	0.91	2.15	0.94	664.75	357.43
$N = 20$	0.93	0.79	0.94	887.70	391.76

Table 6.7: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of N , when $\Delta = 10$.

Note also that for all the values of N , the procedure reaches the desired reliability ($\tilde{\delta}$). With respect to \bar{z}_s/Δ no real trend can be observed. Therefore, no conclusion can be drawn in this case.

Finally, we study the effect of changing the reliability level of our procedure. In Table 6.8 we show the mean number of failures to be observed before we stop testing (\bar{s}), the observed reliability ($\tilde{\delta}$) and the mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 5$. For lower reliability levels release can be done earlier since less failure observations are required to stop testing. However, when the reliability level becomes higher, the procedure tends to be *exhaustive*. For example, as shown in Table 6.8, for a reliability level of $1 - \delta = 0.99$, the 99% of the failures must be observed in order to stop testing.

6.1.3 Case 3: N Poisson distributed, λ known and fixed (Goel-Okumoto model)

We now consider the case where only λ is fixed and N is a random variable that has a prior Poisson distribution with parameter θ . This case was studied in Section 5.3.1 (Case 3) and, as shown there, it corresponds to the Goel-Okumoto model. In this

	\bar{s}/N	$\tilde{\delta}$	\overline{TTT}
$1 - \delta = 0.99$	0.99	1.00	580.93
$1 - \delta = 0.95$	0.94	0.98	368.52
$1 - \delta = 0.90$	0.93	0.94	391.76

Table 6.8: Mean percentage of failures to be observed before we stop testing (\bar{s}/N), observed reliability ($\tilde{\delta}$) and mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 5$.

case, the optimal release time for our procedure is given by

$$z_n = \min \left\{ z \geq 0 : e^{-\theta(e^{-(t_n+z)\lambda} - e^{-(t_n+z+\Delta)\lambda})} \geq 1 - \delta \right\} .$$

When z_n is strictly positive, it has the following form:

$$z_n = \frac{-1}{\lambda} \log \left(-\frac{\log(1 - \delta)}{\theta(1 - e^{-\lambda\Delta})} \right) - t_n . \quad (6.10)$$

Note that our procedure stops as soon as there exists $n \geq 1$ such that $t_n + z_n < t_{n+1}$. Since the failure times are always increasing ($t_1 < t_2 < \dots < t_n$), it is clear that, unlike the previous case, z_n decreases with n . Thus, as long as the system is repaired, the time needed to certify a fixed fault-free interval becomes smaller. Note also that (6.10) provides an *optimal total time on test* for our procedure, which is given by

$$t_n + z_n = -\frac{1}{\lambda} \log \left(-\frac{\log(1 - \delta)}{\theta(1 - e^{-\lambda\Delta})} \right) , \quad (6.11)$$

for all $n \geq 1$. The default settings for the simulations carried out in this section can be seen in Table 6.9. For the results below we will specify only the values that differ

Parameters Goel-Okumoto model:	$\theta = 20, \lambda = 0.0075$
Reliability level:	$1 - \delta = 0.90$
Prior distribution of N :	Poisson(20)
Number of simulations:	$r = 100$

Table 6.9: Default settings for simulations.

from the default ones. First note that the choice of θ and λ determines the scale of the times between failures in the problem. In this case, the sample mean of the times between failures that have been simulated is equal to $\bar{x} = 14.36$.

Like in the previous case, we compute the usual metrics (see Table 6.4) for different values of Δ . The results are shown in Table 6.10. The conclusions here are

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\Delta = 1$	0.28	8.37	0.90	7.43	46.62
$\Delta = 3$	0.74	9.00	0.91	87.45	192.08
$\Delta = 5$	0.85	8.03	0.92	146.76	257.97
$\Delta = 10$	0.93	5.51	0.96	226.93	336.90

Table 6.10: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of Δ .

similar to those obtained from Table 6.6. In fact, we can observe that increasing Δ produces an increase in \bar{s}/N , \bar{z} and \overline{TTT} . Note also that for all the values of Δ , the procedure reaches the desired reliability ($\tilde{\delta}$). With respect to \bar{z}_s/Δ no real trend can be observed and no conclusion can be drawn in this case. The difference in scale between \bar{z}_n and \bar{x}_n (the sample mean of the simulated times between failures) for $\Delta = 3$ is shown in Figure 6.4. We can also observe that \bar{z}_n is decreasing with n , as

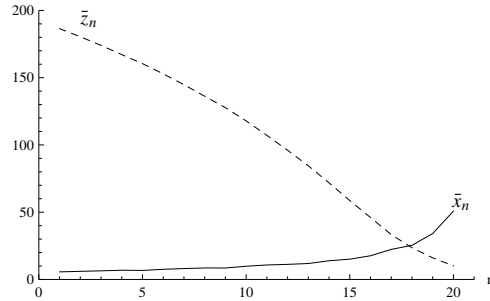


Figure 6.4: Mean release times (dashed) and mean times between failures (solid), when $\Delta = 3$.

we already deduced from (6.10). Finally, note that the value of z_s in Table 6.10 is increasing with Δ . This is the expected behaviour since the larger the Δ we wish to certify, the longer we should wait to do that.

A system containing more faults requires more effort to certify that the next fault will not occurred before Δ . This can be seen in Table 6.11 where the usual metrics (see Table 6.4) are shown for different values of θ , when $\Delta = 10$. We can observe that, \bar{s} , \bar{z} and \overline{TTT} increase with θ while no trend is observed for \bar{z}_s/Δ . Note also that for all the values of θ considered here, the procedure reaches the desired reliability level ($\tilde{\delta}$).

The effect of changing the reliability level of our procedure is the same observed in the previous case (see Table 6.8): for lower reliability levels less failure observations are required to stop testing and when the reliability level tends to 1, the procedure

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\theta = 5$	0.73	4.57	0.94	68.41	151.97
$\theta = 10$	0.83	5.94	0.92	142.88	244.56
$\theta = 15$	0.90	5.94	0.91	192.25	293.38
$\theta = 20$	0.93	5.51	0.96	226.93	336.90

Table 6.11: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of N , when $\Delta = 10$.

tends to be exhaustive. This can be seen in Table 6.12 where we show \bar{s}/N , $\tilde{\delta}$ and \overline{TTT} for different reliability levels, when $\Delta = 5$. Note that also here the procedure

	\bar{s}/N	$\tilde{\delta}$	\overline{TTT}
$1 - \delta = 0.99$	0.99	1.00	458.89
$1 - \delta = 0.95$	0.96	0.96	397.20
$1 - \delta = 0.90$	0.85	0.92	257.97

Table 6.12: Mean percentage of failures to be observed before we stop testing (\bar{s}/N), observed reliability ($\tilde{\delta}$) and mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 5$.

reaches the desired reliability ($\tilde{\delta}$) in all cases.

Since we consider now that N follows a Poisson distribution, we can also compute the posterior distribution of N given failure times $T_1 = t_1, \dots, T_m = t_m$. As shown in (5.45), this is given by

$$\mathbb{P}[N = n_0 \mid T_1 = t_1, \dots, T_m = t_m] = \frac{\theta^{n_0 - m}}{(n_0 - m)!} e^{-(\theta e^{-\lambda t_m} + (n_0 - m)\lambda t_m)}. \quad (6.12)$$

We can now determine the probability of having at most $k \geq 0$ remaining faults in the system given that we have already observed m . Note that such a probability is simply

$$\mathbb{P}[m \leq N \leq m + k \mid T_1 = t_1, \dots, T_m = t_m] = \sum_{j=m}^{m+k} \mathbb{P}[N = j \mid T_1 = t_1, \dots, T_m = t_m]. \quad (6.13)$$

We have computed the probability of having at most $k \geq 0$ remaining faults for different values of k and m assuming that N follows a Poisson distribution with mean $\theta = 20$. The results are shown in Table 6.13. It is remarkable that even in the case where $m = 20$, i.e., the expected number of faults has been observed, the probability of having at most 1 fault left is very small. Thus, if we define a stopping rule based on the probability of having at most k remaining faults when we decide

	$m = 5$	$m = 10$	$m = 15$	$m = 20$
$k = 1$	0.0000004	0.00006	0.011	0.064
$k = 5$	0.0004	0.017	0.369	0.715
$k = 10$	0.039	0.319	0.933	0.994

Table 6.13: Probability of having at most k remaining faults given that m have already been observed when N follows a Poisson distribution with mean $\theta = 20$.

to stop testing, the choice of k is critical since it may be possible that for small values of k the usual 0.90 or 0.95 reliability level is never reached. A stopping rule based on this approach is considered in Chapter 8.

6.1.4 Case 4: N Poisson and λ Gamma distributed (full Bayesian approach)

We finish the study of our certification procedure for the Jelinski-Moranda model by considering that both N and λ are random variables. We assume that N has a prior Poisson distribution with parameter $\theta \geq 0$ and λ has a prior Gamma distribution with parameters $k > 0$ and $w > 0$. As shown in Lemma 5.3, the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ is monotone increasing with respect to z . Therefore, there exists a unique optimal release time $z_n \geq 0$ for our procedure. The default settings for the simulations carried out in this section can be seen in Table 6.14. For the results below we will specify only the values that differ from the

Prior distribution of λ :	Gamma(5,0.0015)
Prior distribution of N :	Poisson(20)
Reliability level:	$1 - \delta = 0.90$
Number of simulations:	$r = 100$

Table 6.14: Default settings for simulations.

default ones.

Like in previous sections, we compute the usual metrics (see Table 6.4) for different values of Δ . The results are shown in Table 6.15. Most of the conclusions here are similar to those obtained in previous sections. We can observe that choosing a large Δ increases \bar{s}/N and \overline{TTT} . The increase of the value of \bar{z} with Δ (and the difference in scale with respect to the failure times) has the same interpretation as in previous cases. This can be observed in Figure 6.5, where we have plotted \bar{z}_n and \bar{x}_n (the sample mean of the simulated times between failures), when $\Delta = 10$. Note that, as in Section 6.1.2, in early stages of testing the values of z_n increase and after a certain number of faults have been removed from the system they start to decrease to eventually become 0. The explanation to this is the same as the one

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\Delta = 1$	0.05	0.00	0.91	0.37	9.40
$\Delta = 3$	0.69	8.65	0.90	62.48	175.30
$\Delta = 5$	0.82	8.42	0.92	137.32	266.76
$\Delta = 10$	0.91	13.82	0.95	351.09	415.07

Table 6.15: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of Δ .

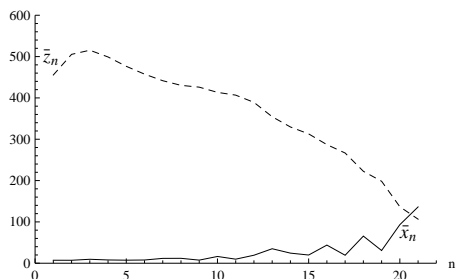


Figure 6.5: Mean release times (dashed) and mean times between failures (solid), when $\Delta = 10$.

given in Section 6.1.2.

The system containing more faults requires more effort to certify that the next fault will not occurred before Δ . In particular, this is reflected in an increase of both \bar{s}/N and \overline{TTT} . The usual metrics (see Table 6.4) for different values of θ , when $\Delta = 10$, are shown in Table 6.16. We can observe that, in fact, all \bar{s} , \overline{TTT}

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\theta = 5$	0.72	7.89	0.92	102.8	195.40
$\theta = 10$	0.88	10.35	0.96	221.6	303.34
$\theta = 15$	0.91	11.11	0.95	308.7	369.26
$\theta = 20$	0.91	13.82	0.95	351.0	415.07

Table 6.16: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of θ , when $\Delta = 10$.

and \bar{z} increase with θ . Note that, unlike in previous cases, \bar{z}_s/Δ (for which no trend had been observed in previous cases) increases with θ . Finally, note that for all the values of θ considered here, the procedure reaches the desired reliability ($\tilde{\delta}$).

The effect of changing the reliability level of our procedure is the same as above. In Table 6.17 we show \bar{s}/N , $\tilde{\delta}$ and \overline{TTT} for different reliability levels, when $\Delta = 10$. As in previous cases, we have that \bar{s} and \overline{TTT} are smaller for small values of $1 - \delta$.

	\bar{s}/N	$\tilde{\delta}$	\overline{TTT}
$1 - \delta = 0.99$	1.00	1.00	555.63
$1 - \delta = 0.95$	0.98	1.00	553.32
$1 - \delta = 0.90$	0.91	0.95	415.07

Table 6.17: Mean percentage of failures to be observed before we stop testing (\bar{s}/N), observed reliability ($\tilde{\delta}$) and mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 10$.

For all the reliability levels considered, the observed reliability ($\tilde{\delta}$) is larger than the reliability level, however, in this example to reach $1 - \delta = 0.99$ requires that all failures must be observed.

As done in Section 6.1.3, we can compute the posterior distribution of N , given $T_1 = t_1, \dots, T_m = t_m$, and use it to calculate the probability of having k remaining faults. This can be observed in Table 6.18 for different values of k and m assuming that N follows a Poisson distribution with mean $\theta = 20$ and λ follows a Gamma distribution with mean 0.0075. Also in this case, the probability of having at most

	$m = 5$	$m = 10$	$m = 15$	$m = 20$
$k = 1$	0.00002	0.002	0.006	0.080
$k = 5$	0.006	0.098	0.201	0.606
$k = 10$	0.158	0.553	0.734	0.958

Table 6.18: Probability of having at most k remaining faults given that m have already been observed when N follows a Poisson distribution with mean $\theta = 20$ and λ follows a Gamma distribution with mean 0.0075.

1 fault left is very small even if the expected number of faults has been observed ($m = 20$). Thus, if we consider a stopping rule based on the probability of having at most k remaining faults, then we must choose k carefully since it may happen that the usual 0.90 or 0.95 reliability level is never reached.

6.2 Run model

The Run model is usually characterized by the probability mass function of the random variable X_i defined as the number of test runs needed to find the i^{th} software fault after repairing the $(i - 1)^{\text{st}}$ fault. As described in Section 5.1.2, it follows a geometric distribution with success parameter

$$p_i = 1 - (1 - \theta)^{N - i + 1}, \quad (6.14)$$

for all $i = 1, \dots, N$, where $N > 0$ and $\theta \in (0, 1)$ are the parameters of the model. Since the geometric distribution can be seen as the discrete case of the Exponential distribution, we may expect that, in general, the Run model behaves in a similar way as the Jelinski-Moranda model. Following the steps of Section 5.3.2, we start with the case where both parameters are assumed to be known and fixed.

6.2.1 Case 1: N and θ deterministic

In Section 5.3.2, we saw that the conditional residual lifetime of X_{n+1} , given by (5.69), is the same as the one obtained for the Jelinski-Moranda model in (5.31) when we take the parameter λ in the Jelinski-Moranda model to be equal to $-\log(1 - \theta)$. Thus, the results given in Table 6.1 and in Table 6.2 are also valid here when $\theta = 1 - e^{-0.0005}$ and $1 - \delta = 0.90$.

6.2.2 Case 2: N Poisson distributed, θ known and fixed

We now assume that θ is fixed and N is a random variable that has a prior Poisson distribution with mean $\lambda \geq 0$. As shown in Section 5.3.2, the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$, for $\Delta \geq 0$, is monotone increasing with respect to z . Therefore, there exists a unique optimal release time $z_n \geq 0$ for our procedure. The default settings for the simulations carried out in this section can be seen in Table 6.19. For the results below we will specify only the values that differ from

Parameters Run model:	$\theta = 0.001, \lambda = 20$
Reliability level:	$1 - \delta = 0.90$
Prior distribution of N :	Poisson(20)
Number of simulations:	$r = 100$

Table 6.19: Default settings for simulations.

the default ones. First note that the choice of θ and λ determines the scale of the times between failures in the problem. In this case, the sample mean of the times between failures that have been simulated is equal to $\bar{x} = 186.07$.

We first compute the usual metrics (see Table 6.4) for different values of Δ . The results are shown in Table 6.20. The conclusions here are similar to those obtained from Table 6.10 in Section 6.1.3. We can observe that increasing Δ produces an increase in both \bar{s}/N and \overline{TTT} . Moreover, for all the values of Δ considered above, the procedure reaches the desired reliability ($\tilde{\delta}$). The increase of the value of \bar{z} with Δ has the same interpretation as in the continuous case: although z_n exists, for all $n \geq 0$, it is often useless in practice since its value is too large with respect to the scale of the data.

The effect of increasing the number of faults in the system is illustrated in Table 6.21 where the usual metrics (see Table 6.4) are shown for different values of λ , when $\Delta = 25$. We can observe that all \bar{s} , \bar{z} and \overline{TTT} increase with θ (thus, more

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\Delta = 25$	0.78	5.13	0.92	771.76	1117.80
$\Delta = 50$	0.91	7.85	0.92	1521.51	2345.99
$\Delta = 100$	0.99	8.80	1.00	4241.87	3321.40

Table 6.20: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of Δ .

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\lambda = 5$	0.25	4.31	0.91	11.56	348.99
$\lambda = 10$	0.57	7.93	0.90	279.12	861.36
$\lambda = 15$	0.73	8.26	0.91	567.02	1270.14
$\lambda = 20$	0.78	5.13	0.92	771.76	1117.80

Table 6.21: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of λ , when $\Delta = 25$.

effort is required to certify that the next fault will not occurred before Δ) and that for all the values of θ considered here, the procedure reaches the desired reliability ($\tilde{\delta}$). For z_s/Δ no trend is observed. Therefore, no conclusion can be drawn in this case.

The effect of changing the reliability level in our procedure can be observed in Table 6.22 where we show \bar{s}/N , $\tilde{\delta}$ and \overline{TTT} for different reliability levels, when $\Delta = 25$. This is the same as in previous cases: for lower reliability levels release can

	\bar{s}/N	$\tilde{\delta}$	\overline{TTT}
$1 - \delta = 0.99$	1.00	1.00	3411.24
$1 - \delta = 0.95$	0.92	0.99	2398.81
$1 - \delta = 0.90$	0.78	0.92	1117.80

Table 6.22: Mean number of failures to be observed before we stop testing (\bar{s}), observed reliability ($\tilde{\delta}$) and mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 25$.

be done earlier (\bar{s} and \overline{TTT} is smaller) and as $1 - \delta$ increases to 1, the procedure tends to be exhaustive (in fact, for $1 - \delta = 0.99$ here, the procedure is exhaustive).

Since N follows a Poisson distribution, we can also compute the posterior distribution of N , given the times between failures $X_1 = x_1, \dots, X_m = x_m$, and use this to calculate the probability of having k remaining faults when we decide to

stop testing. This is shown in Table 6.23, for different values of k and m , where we assume that N follows a Poisson distribution with mean $\theta = 20$. As observed in the

	$m = 5$	$m = 10$	$m = 15$	$m = 20$
$k = 1$	0.00000008	0.00000008	0.0004	0.0369
$k = 5$	0.0001	0.0007	0.0636	0.5967
$k = 10$	0.0159	0.0521	0.5713	0.9842

Table 6.23: Probability of having at most k remaining faults given that m have already been observed when N follows a Poisson distribution with mean $\theta = 20$.

Jelinski-Moranda model, the probability of having at most 1 fault left is very small even if $m = 20$, i.e., the expected number of faults has been observed. Therefore, if we consider a stopping rule based on the probability of having at most k remaining faults, we must carefully choose k since it may be possible that the usual 0.90 or 0.95 reliability level is never reached. A stopping rule based on this approach is developed in Chapter 8.

6.2.3 Case 3: N known and fixed, θ Beta distributed

We now consider the case where N is supposed to be fixed and θ is a random variable that has a prior Beta distribution with parameters $\alpha > 0$ and $\beta > 0$. This case was studied in Section 5.3.2. As shown there, the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ is monotone increasing with respect to z . Therefore, there exists a unique optimal release time $z_n \geq 0$ for our procedure. The default settings for the simulations carried out in this section can be seen in Table 6.24. First note

Number of simulated failures:	$N = 20$
Reliability level:	$1 - \delta = 0.90$
Prior mean of θ :	$\frac{\alpha}{\alpha + \beta} = 0.001$
Number of simulations:	$r = 100$

Table 6.24: Default settings for simulations.

that the choice of the prior distribution determines the scale of the times between failures in the problem (small values of θ produce large times between failures). In this case, the sample mean of the times between failures that have been simulated is equal to $\bar{x} = 192.13$. However, as mentioned in Section 6.1.2, the choice of the prior distribution does not have great effect in the posterior distribution as long as the posterior distribution is updated with new failure observations.

Like in the Jelinski-Moranda case, we compute the usual metrics (see Table 6.4) for different values of Δ . Table 6.25 shows some results for different values of Δ .

The conclusions are similar to those extracted from the corresponding case of the

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\Delta = 25$	0.81	0.45	0.92	3347.56	1584.21
$\Delta = 50$	0.91	0.42	0.92	9899.14	2351.78
$\Delta = 100$	0.96	2.47	0.90	24606.20	2934.71

Table 6.25: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of Δ .

Jelinski-Moranda model. Therefore, if we choose a large Δ for our procedure, then both \bar{s}/N and \overline{TTT} increase. Moreover, for all the values of Δ considered here, the procedure reaches the desired reliability ($\tilde{\delta}$). The increase of the value of \bar{z} with Δ has the same interpretation given in the Jelinski-Moranda model. Thus, we refer to Section 6.1.2 for details.

The system containing more faults requires more effort to certify that the next fault will not occurred before Δ . Thus, an increase of both \bar{s} and \overline{TTT} with N is expected. The usual metrics (see Table 6.4) for different values of N , when $\Delta = 25$, are shown in Table 6.26. We can observe that, in fact, all \bar{s} , \bar{z} and \overline{TTT} increase with

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$N = 5$	0.20	0.00	0.93	0.00	206.68
$N = 10$	0.48	1.31	0.95	340.08	647.88
$N = 15$	0.74	1.52	0.91	2382.41	1292.99
$N = 20$	0.81	0.45	0.92	3347.56	1584.21

Table 6.26: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of N , when $\Delta = 25$.

N , and that for all the values of N considered here the procedure reaches the desired reliability ($\tilde{\delta}$). With respect to \bar{z}_s/Δ no real trend can be observed. Therefore, no conclusion can be drawn in this case.

In Table 6.27 we can observe the effect of increasing the reliability level in our procedure. We show \bar{s}/N , $\tilde{\delta}$ and \overline{TTT} for different reliability levels when $\Delta = 25$. As in previous cases, lower reliability levels require less fault discoveries to stop testing (and thus less testing effort) and as long as $1 - \delta$ increases to 1, the procedure tends to be exhaustive.

6.2.4 Case 4: N Poisson and θ Beta distributed (full Bayesian approach)

We finish the study of our certification procedure for the Run model by considering that both N and θ are random variables. We assume that N has prior Poisson

	\bar{s}/N	$\tilde{\delta}$	\overline{TTT}
$1 - \delta = 0.99$	1.00	1.00	3580.49
$1 - \delta = 0.95$	0.91	0.99	2375.35
$1 - \delta = 0.90$	0.81	0.92	1584.21

Table 6.27: Mean number of failures to be observed before we stop testing (\bar{s}/N), observed reliability ($\tilde{\delta}$) and mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 25$.

distribution with parameter $\lambda \geq 0$ and θ has prior Beta distribution with parameters $\alpha > 0$ and $\beta > 0$. As shown in Section 5.3.2, the the conditional residual lifetime of X_{n+1} given $X^{(n)} = x^{(n)}$ is monotone increasing with respect to z . Therefore, there exists a unique optimal release time $z_n \geq 0$ for our procedure. The default settings for the simulations carried out in this section can be seen in Table 6.28. For the

Prior mean of θ :	$\frac{\alpha}{\alpha + \beta} = 0.001$
Prior distribution of N :	Poisson(20)
Reliability level:	$1 - \delta = 0.90$
Number of simulations:	$r = 100$

Table 6.28: Default settings for simulations.

results below we will specify only the values that differ from the default ones.

We first compute the usual metrics (see Table 6.4) for different values of Δ . The results are shown in Table 6.29. We can observe that choosing a large Δ produces

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\Delta = 25$	0.78	7.60	0.87	767.72	1536.25
$\Delta = 50$	0.90	7.99	0.94	1549.20	2386.17
$\Delta = 100$	0.97	8.39	0.95	4550.24	3497.89

Table 6.29: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of Δ .

an increase in both \bar{s}/N and \overline{TTT} . The increase of the value of \bar{z} with Δ and the difference in scale with respect to the failure times has the same interpretation as in previous cases. Note also that when $\Delta = 25$ the procedure does not reach the desired reliability ($\tilde{\delta}$). This suggests that a larger number of simulations should be

performed. However, simulations of this particular case are very time-consuming. For that reason we have decided to stick to the current number of simulations.

In Table 6.30 we illustrate the effect of increasing the number of faults in the system. We show the usual metrics (see Table 6.4) for different values of λ , when $\Delta = 25$. The conclusions are the same as in previous cases: for a system having

	\bar{s}/N	\bar{z}_s/Δ	$\tilde{\delta}$	\bar{z}	\overline{TTT}
$\lambda = 5$	0.28	2.57	0.95	20.47	279.45
$\lambda = 10$	0.50	7.51	0.88	267.59	837.30
$\lambda = 15$	0.67	9.11	0.90	520.24	1245.47
$\lambda = 20$	0.78	7.60	0.87	767.72	1536.25

Table 6.30: Mean percentage of failures to stop testing (\bar{s}/N), mean true release time as a fraction of Δ (\bar{z}_s/Δ), observed reliability ($\tilde{\delta}$), overall mean release time (\bar{z}) and mean total time of test (\overline{TTT}) for different values of λ , when $\Delta = 25$.

more faults more effort is required to certify that the next fault will not occurred before Δ . This can be observed in an increase in \bar{s} and \overline{TTT} with θ . Note that when $\lambda = 10$ and $\lambda = 20$ the procedure does not reach the desired reliability ($\tilde{\delta}$). Thus, also in this case a larger number of simulations should be performed.

The effect of changing the reliability level of our procedure can be observed in Table 6.31. As in above cases for lower reliability levels release can be done earlier

	\bar{s}/N	$\tilde{\delta}$	\overline{TTT}
$1 - \delta = 0.99$	1.00	1.00	3747.03
$1 - \delta = 0.95$	0.90	0.99	2470.83
$1 - \delta = 0.90$	0.78	0.87	1536.25

Table 6.31: Mean percentage of failures to be observed before we stop testing (\bar{s}/N), observed reliability ($\tilde{\delta}$) and mean total time of test (\overline{TTT}) for different reliability levels, when $\Delta = 25$.

(\overline{TTT} and \bar{s}/N are smaller) and as $1 - \delta$ increases to 1, the procedure tends to be exhaustive. In this case, the procedure is in fact exhaustive when $1 - \delta = 0.99$.

Since N follows a Poisson distribution, we can also compute the posterior distribution of N given the times between failures $X_1 = x_1, \dots, X_m = x_m$ and use it to calculate the probability of having k remaining faults when we decide to stop testing. This can be observed in Table 6.32 for different values of k and m when N is assumed to have a prior Poisson distribution with mean $\lambda = 20$ and θ has a prior Beta distribution with mean 0.001. Also in this case, the probability of having at most 1 remaining fault is very small even if $m = 20$ (the expected number of faults has been observed). Therefore, if we consider a stopping rule based on the probability of having at most k remaining faults, then we must choose k carefully

	$m = 5$	$m = 10$	$m = 15$	$m = 20$
$k = 1$	0.000008	0.0001	0.0054	0.1109
$k = 5$	0.00402	0.0309	0.2599	0.8216
$k = 10$	0.1465	0.4224	0.8759	0.9982

Table 6.32: Probability of having at most k remaining faults given that m have already been observed when N has a prior Poisson distribution with mean $\lambda = 20$ and θ has a prior Beta distribution with mean 0.001.

since it may happen that the usual 0.90 or 0.95 reliability level is never reached.

CHAPTER 7

MODEL-BASED TESTING FRAMEWORK

Model-based testing is a software testing method consisting of automatic generation of efficient tests using models of the system (see e.g. [Andrews et al. \(2005\)](#), [Brinksma and Tretmans \(2001\)](#), [El-Far and Whittaker \(2001\)](#) and [Lee and Yannakakis \(1996\)](#)). Formalization of testing theory was first presented in [De Nicola and Hennessy \(1983\)](#). A few years later in [Bernot et al. \(1991\)](#) a formal theory based on abstract data type specifications was introduced, establishing the foundations of functional testing. As mentioned in [Section 1.1.3](#), functional testing focuses on black-box testing since only the input-output relation is tested. There it is assumed that the tester has no knowledge about the internal structure (implementation) of the system under test and test cases are generated using software specifications only. We now focus on structural (model-based) testing. Here we assume that some details about the software implementation are known to the tester. We use this knowledge to execute test cases which are generated based on *models* describing part of the behaviour of the system. Thus, the software can be tested in a more efficient way. In particular, we are interested in models that describe the control flow over the system *components*. However, we abstract from the testing of the components themselves (which can be done by functional testing). Both test strategies (black-box and model-based) consider that the software is subjected to a set of inputs that produces a corresponding set of outputs from which software failures can be identified. Besides this, in model-based testing we have the notion of software component. Thus, we can also identify the components of the system that behave according to the software specifications which is not possible in the black-box approach.

We model software systems as *labelled transition systems* (a special class of *Petri nets*) where each transition in a labelled transition system represents a software component. Among the rich literature on labelled transition systems based testing we mention [Brinksma \(1989\)](#), [Chow \(1978\)](#), [Howden \(1975\)](#), [Huang \(1975\)](#) and [Tretmans \(1992\)](#). We assume that the software components either have correct or erroneous behaviour. A component can be started if we follow a path leading to the component. If the component behaves correctly, it ends after a certain time producing an output result. We assume that there is a way to determine whether the execution of a component is according to the specifications, for example by functional testing, and a fault in a component can be detected e.g. by generating an exception or by a time out. Therefore, we only discover a fault if we start the corresponding component. Thus, in this context, testing means executing software components and observing whether they behave correctly or not. We model a software fault as a symbolic labelling of a Petri net transition which can only be discovered when the transition is fired. Our approach can be seen as a stepping stone for more realistic models with several parallel threads (cf. [Bowman and Gomez \(2005\)](#), [Milner \(1980\)](#), [Plotkin \(1983\)](#) and [Schneider \(1999\)](#)).

In the model-based testing literature all kind of test strategies have been considered in order to avoid exhaustive testing which is not always feasible in practice (see e.g. [Zhu et al. \(1997\)](#)). These strategies are usually expressed in terms of coverage where a certain amount of components are covered during testing. The difference with existing approaches (cf. [Andrews et al. \(2005\)](#), [Belli et al. \(2006\)](#), [Denise et al. \(2004\)](#)) is that we provide a statistical stopping rule, that may depend on the underlying way of walking through the system, which allows us to stop earlier with a certain statistical reliability. Hence, our statistical procedure can also be considered as a coverage method (over the components) but with a *statistical* measure of quality. Moreover, our procedure becomes exhaustive when 100% reliability is required. Despite the extensive literature on statistical stopping criteria for functional (black-box) testing ([Chen et al. \(1999\)](#), [Di Bucchianico et al. \(2008\)](#), [Lee et al. \(2001\)](#) and [Moralì and Soyer \(2003a\)](#) to mention some of them), there do not seem to be similar criteria for structural testing. Our statistical procedure should not be confused with the common statistical testing techniques developed in [Thevenod-Fosse et al. \(1995\)](#). The term *statistical testing* is normally used for the probability of coverage (components, branches, etc.) while we are using it for metrics like the remaining number of faults in the system. For example, our procedure stops if the probability of having a predetermined number of remaining faults is smaller than a certain confidence limit. Our underlying test strategy is also statistically based, in the sense that the selection of the next transition to be tested is chosen at random. Moreover, we have a reduction algorithm to reduce the model based on observed (and repaired) components. As we will see in Chapter 8, the reduction algorithm is *efficient* since the mean number of runs to reach exhaustiveness is significantly reduced when the algorithm is applied for the testing strategy.

In the remainder of this chapter we introduce the assumptions and basic concepts that we consider for model-based testing. The model we use for describing the behaviour of software systems (labelled transition systems) is introduced in Section 7.1. In Section 7.2 we illustrate with an example how we can map the abstract model to a real application. In Section 7.3 we define what a software *fault* is for us and comment on different choices for the probability distribution of the total number of faults in the system. In Section 7.4 we define how the system is tested. In Section 7.5 we introduce the concepts of a walking function and the walking function update. Finally, in Section 7.6 we summarize the new notation introduced in this chapter since it is used in the following ones. Some of the results shown in this chapter are also presented in [Corro Ramos et al. \(2008\)](#).

7.1 Labelled transition systems and a diagram technique for representation

As mentioned in the introduction of this chapter, our main goal is to develop statistical software release procedures based on the architecture of software systems. We assume that software systems can be considered as a set of components running as one sequential process. We use a special kind of *labelled transition systems* to model

such system, i.e., a graph model with labelled arcs. Labelled transition systems are widely studied in computer science and can be defined as follows.

Definition 7.1 (Labelled transition system). *A labelled transition system (LTS) is a triple $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$, where*

- (1) \mathcal{S} is a non-empty finite set whose elements are called states,
- (2) \mathcal{T} is a non-empty finite set whose elements are called transitions,
- (3) $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{T} \times \mathcal{S}$ is a non-empty finite set whose elements are defined as follows: for all $t \in \mathcal{T}$ there exist a unique pair of states $(s, s') \in \mathcal{S} \times \mathcal{S}$ such that $(s, t, s') \in \mathcal{R}$,
- (4) there is exactly one state $i \in \mathcal{S}$ (called the initial state) such that there is no triple $(s, t, i) \in \mathcal{R}$,
- (5) there is at least one state $f \in \mathcal{S}$ (called final state) such that there is no triple $(f, t, s) \in \mathcal{R}$,
- (6) for any non-initial state $s \in \mathcal{S} \setminus \{i\}$ there is a sequence $(s_1, t_1, s_2, \dots, s_n, t_n, s_{n+1})$ such that $(s_k, t_k, s_{k+1}) \in \mathcal{R}$, for all $1 \leq k \leq n$, $s_1 = i$, and $s_{n+1} = s$, i.e., any non-initial state is reachable from the initial one.

When we do not wish to specify the name of a transition we say that there is an arc from s to s' . Given an LTS, for all $s \in \mathcal{S}$, we define the preset and the postset of s as

$$\bullet s = \{u \in \mathcal{S} \mid \exists t \in \mathcal{T} : (u, t, s) \in \mathcal{R}\}$$

and

$$s \bullet = \{v \in \mathcal{S} \mid \exists t \in \mathcal{T} : (s, t, v) \in \mathcal{R}\},$$

respectively.

We assume that a software component can be started if we follow a *path* leading to the component. A path in an LTS can be defined in the following way.

Definition 7.2 (Path in an LTS). *A path in an LTS is either the empty sequence, denoted by ε , or a sequence $p = (s_1, t_1, \dots, s_n, t_n, s_{n+1})$ such that, for all $1 \leq k \leq n$, $(s_k, t_k, s_{k+1}) \in \mathcal{R}$. We may use $first(p)$ and $last(p)$ to denote the first and the last states of p , i.e., the states s_1 and s_{n+1} , respectively. A subpath of a path p is a subsequence p' of p such that p' is also a path and it starts and ends with a state. A path is said to be linear if for all s_k , $1 < k < n+1$, it follows that $|\bullet s_k| = |s_k \bullet| = 1$. We say that a path $p = (s_1, t_1, \dots, s_n, t_n, s_{n+1})$ is a cycle if $s_1 = s_{n+1}$. An LTS is acyclic if it does not contain cycles. An LTS is said to be a one-path LTS if $|\bullet s| = 1$ for all non-final states $s \in \mathcal{S}$.*

In fact, the LTS defined above has the properties of an *S-net*, which is a special structured Petri net (cf. [Desel and Esparza \(1995\)](#)), and it is simultaneously a *workflow net*, another type of Petri net used to model business processes (cf. [van der](#)

Aalst and van Hee (2002)). Labelled transition systems can be seen as a subclass of Petri nets (see e.g. Desel and Esparza (1995) and Reisig (1985)) where each transition has exactly one incoming and one outgoing arc. Petri nets are known as abstract models of concurrent systems. The interest for Petri nets is caused by their multiple applications in very different areas, including workflow management, data analysis, logistics, diagnosis, reliability engineering, concurrent programming and software design. Since we do not use concurrency here, we do not need the full functionality of Petri nets. For that reason we have chosen the above class of labelled transition systems as a model for software. We use a *diagram technique* for our LTS that is derived from Petri nets (see Reisig (1985)). The LTS is displayed as a graph with two types of nodes: circles for states and rectangles for state-transitions (or simply transitions). Transitions are connected by directed arcs to two states: an input state and an output state. Further, we introduce an extra node, called or-construct and represented by a diamond, as a shortcut for a pattern like in Figure 7.1. Note that it is easy to replace the or-construct by standard

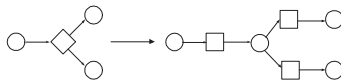


Figure 7.1: Replacement of the or-construct (diamond) by standard constructs.

constructs: the or-construct is replaced by one new place, and a new transition is added for each input or output place of the or-transition - connecting that place with the newly added one.

7.2 Example of modelling software as a labelled transition system

This example represents a simplified generic medical workflow of a hospital. The business processes of the hospital are supported by an information system that is mainly used for updating the electronic patient records, the planning of activities and the application of medical protocols. In this process the patient is central and each new illness of a patient is a new case that flows through the process. Each activity in the process is associated with a software service performed by a software component. We now describe the process in more detail. It starts with the intake activity. Then a doctor observes the patient and makes a diagnosis. According to this diagnosis the patient is released (no illness or not treatable) or a plan is made for further investigations (testing) or a therapy is chosen. Each test or therapy has its own specific activities and according to the outcomes and the plan, it is decided to continue with testing or a therapy, or the patient goes back to the doctor for a new diagnosis, in which case the whole process may be repeated. This process is displayed in Figure 7.2. In fact, the model is made in *Yasper*, a Petri net tool (see van Hee et al. (2006) for details). Since each activity is associated with a software function embodied in a component, a trace through this process model is at the

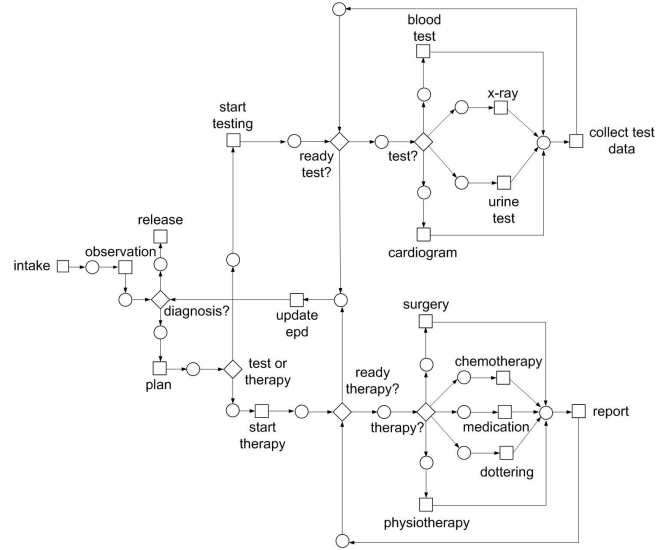


Figure 7.2: Generic medical workflow of a hospital.

same time a test run. We assume that when we call a software service associated with an action we are able to see whether the function is correct or not. We will return to this example in Section 8.3 to illustrate a real application of our whole approach.

7.3 Error distribution

As explained in the previous section, we use a special type of labelled transition systems for modelling software behaviour. Our main assumption is to represent software components as transitions that either behave correctly or have an error. We do not specify what an error is, in the sense that we do not classify it (see Section 1.1.2). Note that we abstract from the real world by assuming that there is a way to determine (perfectly without mistake) whether the execution of a transition is conforming to the specifications (for example, by functional testing). Therefore, for us an error is a symbolic labelling of a transition. Thus, the population of interest here is a finite set of transitions, denoted by \mathcal{T} , where two different types of transitions are considered: *error-marked* and *error-free*.

Definition 7.3 (Error). *A symbolic marking of transitions in an LTS is a function $M : \mathcal{T} \rightarrow \{0, 1\}$ such that*

$$M(t) = \begin{cases} 1 & \text{if } t \text{ is error-marked,} \\ 0 & \text{if } t \text{ is error-free.} \end{cases}$$

The set $\mathcal{N} = \{t \in \mathcal{T} \mid M(t) = 1\}$ is called the error set and the total number of error-marked transitions is denoted by N .

We assume that the error-marking is the result of a *random process* done (by accident) by the programmers and that it is unknown to the tester. We define the *error-marking process* as a *Bernoulli process* consisting of repeatedly performing independent and identical Bernoulli trials. In this case, a Bernoulli trial consists of selecting *without replacement* a transition $t \in \mathcal{T}$ and labelling it as an error with unknown probability $\theta \in (0, 1)$. If $T > 0$ denotes the total number of transitions in \mathcal{T} and t_j is the j^{th} transition sampled from \mathcal{T} , then the random variable

$$Y_j = \begin{cases} 1 & \text{if } t_j \text{ is error-marked,} \\ 0 & \text{if } t_j \text{ is error-free,} \end{cases} \quad (7.1)$$

for all $j = 1, \dots, T$, represents the *marking* of the corresponding transition. Note that the random variables Y_1, \dots, Y_T defined in this way are independent and identically distributed Bernoulli random variables with parameter θ . Therefore,

$$\mathbb{P}[Y_j = y_j] = \theta^{y_j} (1 - \theta)^{1 - y_j}, \quad (7.2)$$

for $y_j = 0, 1$ and $j = 1, \dots, T$. In this way, after T trials, the Bernoulli error-marking process finishes providing us a set of T transitions divided into error-marked and error-free. We also adopt here the Bayesian approach introduced in Section 5.2. Thus, we consider the marking probability as a random variable, denoted by Θ . In particular, we assume that Θ follows a Beta distribution with parameters $a > 0$ and $b > 0$. Therefore, the prior density function of Θ is given by

$$f(\theta) = \frac{\theta^{a-1} (1 - \theta)^{b-1}}{\beta(a, b)}, \quad (7.3)$$

for all $\theta \in (0, 1)$, where $\beta(a, b)$ is the Euler Beta function. It is well-known that the choice of this prior distribution is suitable for Θ since it has support on $(0, 1)$ and its flexibility (two-parameter distribution) can describe many different types of situations. Moreover, the Beta distribution is a *conjugate prior* (see Section 5.2) for Bernoulli likelihoods as we will see now. Conditional on $\{\Theta = \theta\}$, the random variables Y_j , for all $j = 1, \dots, T$, are i.i.d. from a Bernoulli distribution with success parameter θ . Thus, we can write the joint probability mass function of $Y_1 = y_1, \dots, Y_n = y_n$ (denoted as in previous chapters by $Y^{(n)} = y^{(n)}$) given $\Theta = \theta$, as follows:

$$\begin{aligned} \mathbb{P}\left[Y^{(n)} = y^{(n)} \mid \Theta = \theta\right] &= \prod_{j=1}^n \mathbb{P}[Y_j = y_j \mid \Theta = \theta] \\ &= \prod_{j=1}^n \theta^{y_j} (1 - \theta)^{1 - y_j} \\ &= \theta^{\sum_{j=1}^n y_j} (1 - \theta)^{n - \sum_{j=1}^n y_j}. \end{aligned} \quad (7.4)$$

Therefore, the posterior density function of Θ , denoted by $g(\theta | y^{(n)})$, can be computed using (7.3), (7.4) and Bayes formula and it is given by

$$g(\theta | y^{(n)}) = \frac{\theta^{\tilde{a}-1}(1-\theta)^{\tilde{b}-1}}{\int_0^1 \vartheta^{\tilde{a}-1}(1-\vartheta)^{\tilde{b}-1} d\vartheta}, \quad (7.5)$$

where $\tilde{a} = \sum_{j=1}^n y_j + a$ and $\tilde{b} = n - \sum_{j=1}^n y_j + b$. Therefore, the posterior distribution of Θ is also a Beta distribution with parameters \tilde{a} and \tilde{b} . Next we study the cases where the probability distribution of N (the total number of faults in the system) is Binomial and Poisson, respectively.

7.3.1 Binomial distribution of error-marked transitions

With the definition of the marking of a transition given in (7.1), the total number of error-marked transitions in \mathcal{T} can be expressed as $N = \sum_{j=1}^T Y_j$, where $T = |\mathcal{T}|$. We are interested in predicting the number of remaining error-marked transitions in the system given that we have already seen n transitions in total and $m_n = \sum_{j=1}^n y_j$ are error-marked. This can be done by computing the probability mass function of N given $Y^{(n)} = y^{(n)}$. Application of the Bayes rule and the Law of Total Probability yields

$$\begin{aligned} \mathbb{P}[N = j | Y^{(n)} = y^{(n)}] &= \mathbb{P}[Y^{(n)} = y^{(n)} | N = j] \times \\ &\times \frac{\int_0^1 \mathbb{P}[N = j | \Theta = \theta] f(\theta) d\theta}{\int_0^1 \mathbb{P}[Y^{(n)} = y^{(n)} | \Theta = \theta] f(\theta) d\theta}, \end{aligned} \quad (7.6)$$

where $j \geq \sum_{j=1}^n y_j$. We now calculate all the probabilities in (7.6). First note that, conditional on $\{\Theta = \theta\}$, the random variables Y_1, \dots, Y_T are i.i.d. Bernoulli random variables with parameter θ . Thus, given $\Theta = \theta$, the random variable N is binomially distributed with parameters T and θ . Therefore,

$$\mathbb{P}[N = j | \Theta = \theta] = \binom{T}{j} \theta^j (1-\theta)^{T-j}. \quad (7.7)$$

Note that also that $\mathbb{P}[Y^{(n)} = y^{(n)} | \Theta = \theta]$ is the result of a binomial experiment where the order is taken into account. Thus,

$$\mathbb{P}[Y^{(n)} = y^{(n)} | \Theta = \theta] = \theta^{m_n} (1-\theta)^{n-m_n}. \quad (7.8)$$

Finally, $\mathbb{P}[Y^{(n)} = y^{(n)} | N = j]$ can be computed as follows. First note that this is the probability of having a sequence (y_1, \dots, y_n) of zeros and ones, given that there are j error-marked transitions in a population of transitions of size T . We

know that these j errors are randomly distributed (without replacement) over the T transitions. Therefore, it is the same derivation as for a hypergeometric experiment but we take the order in sequences into consideration. Thus, if we assume that there are m_n error-marked transitions in a sample of size n , then we can write

$$\begin{aligned} \mathbb{P}\left[Y^{(n)} = y^{(n)} \mid N = j\right] &= \frac{\binom{T-j}{n-m_n} (n-m_n)! \binom{j}{m_n} m_n!}{\binom{T}{n} n!} \\ &= \frac{\binom{T-j}{n-m_n} \binom{j}{m_n}}{\binom{T}{n} \binom{n}{m_n}}. \end{aligned} \quad (7.9)$$

Hence, substitution into (7.6) yields

$$\mathbb{P}\left[N = j \mid Y^{(n)} = y^{(n)}\right] = \binom{T-n}{j-m_n} \frac{\int_0^1 \theta^j (1-\theta)^{T-j} f(\theta) d\theta}{\int_0^1 \theta^{m_n} (1-\theta)^{n-m_n} f(\theta) d\theta}. \quad (7.10)$$

Note that, in fact, the random sequence $Y^{(n)}$ is *exchangeable*, i.e., any permutation of $Y^{(n)}$ has the same joint probability distribution as any other permutation (see e.g. Schervish (1995)[p.28]). Therefore, we have that $\mathbb{P}\left[N = j \mid Y^{(n)} = y^{(n)}\right] = \mathbb{P}\left[N = j \mid \sum_{j=1}^n Y_j = m_n\right]$, as we will see now. Conditional on $\{\Theta = \theta\}$ the random variable $\sum_{j=1}^n Y_j$ is binomially distributed with parameters n and θ . Therefore,

$$\mathbb{P}\left[\sum_{j=1}^n Y_j = m_n \mid \Theta = \theta\right] = \binom{n}{m_n} \theta^{m_n} (1-\theta)^{n-m_n}. \quad (7.11)$$

Moreover, in $\mathbb{P}\left[\sum_{j=1}^n Y_j = m_n \mid N = j\right]$ the order in the sequence $Y^{(n)}$ does not play a role anymore. Thus,

$$\mathbb{P}\left[\sum_{j=1}^n Y_j = m_n \mid N = j\right] = \frac{\binom{T-j}{n-m_n} \binom{j}{m_n}}{\binom{T}{n}}. \quad (7.12)$$

Therefore, if we compute $\mathbb{P}\left[N = j \mid \sum_{j=1}^n Y_j = m_n\right]$, then we obtain exactly the same result as in (7.10). In this way we show that, in order to predict the distribution of N , we only require $\sum_{j=1}^n Y_j$ and not the whole sequence $Y^{(n)}$. Note also that if new data is collected, then the posterior distribution of Θ can be computed as

shown in (7.5). Therefore, we can use the posterior distribution of Θ as a new prior in order to compute (7.10) again. This procedure of collecting data and updating the distribution of Θ can be done in several stages. This idea is used in Chapter 8 to define fully sequential certification procedures. In particular, if we assume a prior Beta distribution for Θ , then its density function is given by (7.3) and we can write (7.10) as follows:

$$\mathbb{P}\left[N = j \mid Y^{(n)} = y^{(n)}\right] = \binom{T-n}{j-m_n} \frac{\int_0^1 \theta^{j+a-1} (1-\theta)^{T-j+b-1} d\theta}{\int_0^1 \theta^{m_n+a-1} (1-\theta)^{n-m_n+b-1} d\theta}. \quad (7.13)$$

Note that when $a = b = 1$, the Beta distribution is in fact the Uniform distribution on $(0, 1)$ and (7.13) can be written as follows:

$$\begin{aligned} \mathbb{P}\left[N = j \mid Y^{(n)} = y^{(n)}\right] &= \binom{T-n}{j-m_n} \frac{j!(T-j)!(n+1)!}{m_n!(n-m_n)!(T+1)!} \\ &= \frac{\binom{T-n}{j-m_n} \binom{n}{m_n} n+1}{\binom{T}{j}} \frac{1}{T+1}. \end{aligned} \quad (7.14)$$

The choice of a prior Uniform distribution on the interval $(0, 1)$ for Θ is often considered since it represents total ignorance about possible outcomes of Θ . In case that we have some prior knowledge about the value of Θ , this should be reflected in the prior distribution. Note that we can use (7.14) to compute $\mathbb{P}\left[N > m_n \mid Y^{(n)} = y^{(n)}\right]$, i.e., the probability of having remaining error-marked transitions in the system. A certification procedure based on the remaining number of error-marked transitions when we decide to stop testing is presented in Section 8.1.

7.3.2 Poisson distribution of error-marked transitions

Let us consider an LTS where the total number of transitions, denoted by T , is very large. Suppose also that from some past experience it is known that the error-marking probability θ is very small. In fact, we may assume that when $T \rightarrow \infty$ and $\theta \rightarrow 0$, the product $T\theta$ tends to a non-zero constant λ . Under these conditions, a Binomial random variable converges in distribution to a Poisson random variable (see e.g. Ross (2007)[Section 2.2.4] for details). Therefore, we may assume that N (the total number of error-marked transitions) follows a Poisson distribution with parameter λ . Like in the binomial case we adopt a Bayesian approach (see Section 5.2). Thus, we consider the parameter of the Poisson distribution as a random variable, denoted by Λ . We assume here that Λ follows a Gamma distribution with parameters $a > 0$ and $b > 0$. Therefore, the prior density function of Λ is given by

$$f(\lambda) = \frac{\lambda^{a-1} e^{-\lambda/b}}{b^a \Gamma(a)}, \quad (7.15)$$

for all $\lambda > 0$, where $\Gamma(a)$ is the Gamma function of a . It is well-known that the choice of this prior distribution is suitable for Λ since it has support on $(0, \infty)$ and its flexibility (two-parameter distribution) can describe many different types of situations. Moreover, the Gamma distribution is a conjugate prior for Poisson likelihoods (see e.g. DeGroot (2004)[Theorem 1, p.164]). We now compute the probability mass function of N given $Y^{(n)} = y^{(n)}$. Application of the Bayes rule and the Law of Total Probability yields

$$\begin{aligned} \mathbb{P} [N = j \mid Y^{(n)} = y^{(n)}] &= \\ &= \frac{\mathbb{P} [Y^{(n)} = y^{(n)} \mid N = j] \int_0^\infty \mathbb{P} [N = j \mid \Lambda = \lambda] f(\lambda) d\lambda}{\sum_{\ell=0}^T \mathbb{P} [Y^{(n)} = y^{(n)} \mid N = \ell] \int_0^\infty \mathbb{P} [N = \ell \mid \Lambda = \lambda] f(\lambda) d\lambda}. \end{aligned} \quad (7.16)$$

Note that $\mathbb{P} [Y^{(n)} = y^{(n)} \mid N = j]$ is given in (7.9) and

$$\mathbb{P} [N = j \mid \Lambda = \lambda] = \frac{\lambda^j e^{-\lambda}}{j!}, \quad (7.17)$$

for all $j \geq 0$. Note now that

$$\begin{aligned} \int_0^\infty \mathbb{P} [N = j \mid \Lambda = \lambda] f(\lambda) d\lambda &= \int_0^\infty \frac{\lambda^{j+a-1} e^{-\lambda(1+\frac{1}{b})}}{b^a \Gamma(a) j!} d\lambda \\ &= \frac{\Gamma(j+a)}{\Gamma(a) j! b^a (1+\frac{1}{b})^{-(j+a)}}. \end{aligned} \quad (7.18)$$

Substitution into (7.16) yields

$$\mathbb{P} [N = j \mid Y^{(n)} = y^{(n)}] = \frac{\binom{T-j}{n-m_n} \binom{j}{m_n} \frac{\Gamma(j+a)}{j! (1+\frac{1}{b})^{-(j+a)}}}{\sum_{\ell=0}^T \binom{T-\ell}{n-m_n} \binom{\ell}{m_n} \frac{\Gamma(\ell+a)}{\ell! (1+\frac{1}{b})^{-(\ell+a)}}}. \quad (7.19)$$

As mentioned in the case of the Binomial distribution of the error-marked transitions, we can use (7.19) to compute the probability of having remaining error-marked transitions in the system when we decide to stop testing. A certification procedure based on this probability is presented in Section 8.1.

7.4 Testing process

In this section we describe our testing process introducing the assumptions and basic concepts considered for this and the following chapters. We define a *run* as a path

in an LTS. If a path ends without discovering an error-marked transition, then it is said to be a *successful* run, and if it ends in an error-marked transition, then it is said to be a *failure* run. Thus, at most one software fault can be found in one run. The definition of run is as follows:

Definition 7.4 (Run in an LTS). *Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS with a unique initial state, denoted by i . A run in L , denoted by σ , is a path $\sigma = (i, t_1, \dots, s_n, t_n, s_{n+1})$ such that $s_k \neq s_j$, for all $k \neq j$, where $1 \leq k \leq n$, $1 \leq j \leq n$ and $s_1 = i$. A run is said to be a *successful run* if and only if all transitions in σ are error-free, (i.e., $M(t_\ell) = 0$, for all $1 \leq \ell \leq n$) and either s_{n+1} is a final state in L (i.e., $|s_{n+1} \bullet| = \emptyset$) or σ is a cycle (i.e., there exists exactly one $1 \leq k \leq n$ such that $s_k = s_{n+1}$). A run is said to be a *failure run* if and only if t_n is the only error-marked transition in σ , i.e., $M(t_\ell) = 0$, for all $1 \leq \ell < n$, and $M(t_n) = 1$. We denote by Σ the set of all runs in L .*

The fault finding process consists of executing runs (which are either successful or failure) in labelled transitions systems. As soon as a fault is discovered, it is *repaired* before we continue testing. That means that the labelling of the transition at hand is changed from 1 to 0. Thus, in this case, we can speak of *perfect repair*. Moreover, we also assume that we do not introduce new faults during the reparation. Therefore, the number of error-marked transitions in the system decreases as long as the system is being tested. The time spent repairing faults is not considered here either. Thus, we can speak of *immediate repair* of faults in this case. All the assumptions related to software testing that we consider now are shown in Table 7.1.

Assumptions
Perfect fault detection
At most one fault can be found in one run
Simultaneous faults do not occur
As soon as a run ends (successful or failure) we start a new run
Immediate repair
Perfect repair
New faults are not introduced during reparation

Table 7.1: Software reliability assumptions for model-based testing.

We now define a special test procedure that specifies a probability distribution on each branching point of an LTS. We called this procedure a *walking function*.

Definition 7.5 (Walking function). *Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS. A walking function for L is a function $w : \mathcal{T} \rightarrow [0, 1]$ such that for all non-final states $s \in \mathcal{S}$, it follows that $\sum_{t \in s \bullet} w(t) = 1$. We denote by \mathcal{W} the set of all walking functions.*

Note that, according to Definition 7.1, each transition t in an LTS has exactly one incoming state s . Thus, for t fixed, there do not exist two states $s_1, s_2 \in \mathcal{S}$ such that $t \in s_1 \bullet$ and $t \in s_2 \bullet$. Note also that if s is a final state, then $s \bullet = \emptyset$ and a run stops when it arrives at s (see Definition 7.4). For that reason the walking function is not defined for final states. The walking function can be interpreted as follows. All the transitions connected to the current state are weighted with nonzero probabilities and therefore, all of them are reachable. When the system is in a certain state, the next transition to be executed is chosen by a weighted random drawing based on the walking function. After each successful run, the walking function may be updated in order to produce a new one. This new walking function assigns probability zero to some already executed transitions so that for the next execution those transitions will not fire. Note that a zero probability transition can also be considered as a non-existing transition. The update of the walking function is done by the following procedure:

Definition 7.6 (Walking function update). *Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS and Σ the set of all runs in L . A Walking Function Update (WFU) function is a function $U : \mathcal{W} \times \Sigma \rightarrow \mathcal{W}$ such that for all $w \in \mathcal{W}$ and $\sigma \in \Sigma$, if $w(t) = 0$ for some $t \in \mathcal{T}$, then $U(w, \sigma)(t) = 0$.*

Thus, an update means that no transitions are added but transitions may get blocked. A detailed description of a walking function and WFU function, as well as a proof of exhaustiveness for the test procedure, are given in Section 7.5. However, as mentioned in Section 1.1.4, software systems are becoming so complex that it is often unfeasible to perform exhaustive testing in practice. In such a situation, *statistical procedures* must be considered in order to avoid exhaustiveness. Statistical certification procedures are studied in detail in Chapter 8.

Let us now consider the random variable X_j representing the label (error-free or error-marked) of the j^{th} tested transition. Thus, we can define the *testing process* as the random sequence $X^{(T)} = (X_1, \dots, X_T)$. In fact, $X^{(T)}$ is a permutation of the elements in $\{1, 2, \dots, T\}$. Note that the testing process is executed by the tester and depends on the walking strategy. Note also that, besides the testing process, we defined in (7.1) a different process, namely the *error-marking process*, as the random sequence $Y^{(T)} = (Y_1, \dots, Y_T)$ of i.i.d. Bernoulli random variables (with success parameter θ). The error-marking process is executed by the programmers and, as we will see below, it does not depend on the walking strategy. In this context, $x^{(n)} = (x_1, \dots, x_n)$, for all $n = 1, \dots, T$, is the *test history* and $m_n = \sum_{j=1}^n y_{x_j}$ is the number of *observed* error-marked transitions. Our main assumption here is to consider the testing process to be *non-anticipative*, i.e., it does not depend on future observed transitions but it may depend on the past (test history). Thus, if $\mathcal{U} = \{1, 2, \dots, T\} \setminus \{x_1, \dots, x_n\}$ is the set of non-tested transitions, then for any $k \in \mathcal{U}$, we assume that

$$\begin{aligned} & \mathbb{P} \left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, Y_1 = y_1, \dots, Y_T = y_T \right] \\ &= \mathbb{P} \left[X_{n+1} = k \mid X^{(n)} = X^{(n)}, Y_{x_1} = y_{x_1}, \dots, Y_{x_n} = y_{x_n} \right]. \end{aligned} \quad (7.20)$$

Therefore, the next transition to be observed, given by X_{n+1} , is independent of the unseen transition marks. Although it is *intuitively obvious*, we will prove now that the error-marking of transitions at the beginning (caused by the programmers) gives the same distribution as error-marking on the fly (when a transition is tested). We first need the following result.

Lemma 7.1. *Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS with $|\mathcal{T}| = T \geq 1$. For any test history $x^{(n)} = (x_1, \dots, x_n)$, $n = 1, \dots, T$, error-marking process $y^{(T)} = (y_1, \dots, y_T)$ and $k \in \mathcal{U}$ (transition k is not tested), define the random vector $\tilde{Y} = (Y_{\ell_1}, \dots, Y_{\ell_{T-1}})$ such that $k \notin \{\ell_1, \dots, \ell_{T-1}\}$. Then, with the convention that $X^{(0)} = 0$, for any $y \in \{0, 1\}$ it follows that*

$$\mathbb{P} \left[Y_k = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y} \right] = \mathbb{P} \left[Y_k = y \mid X^{(n-1)} = x^{(n-1)}, \tilde{Y} = \tilde{y} \right] .$$

Proof. First note that

$$\begin{aligned} \mathbb{P} \left[Y_k = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y} \right] &= \frac{\mathbb{P} \left[Y_k = y, X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y} \right]}{\mathbb{P} \left[X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y} \right]} \\ &= \frac{\mathbb{P} \left[X_n = x_n \mid X^{(n-1)} = x^{(n-1)}, Y_k = y, \tilde{Y} = \tilde{y} \right]}{\mathbb{P} \left[X_n = x_n \mid X^{(n-1)} = x^{(n-1)}, \tilde{Y} = \tilde{y} \right]} \times \\ &\quad \times \frac{\mathbb{P} \left[X^{(n-1)} = x^{(n-1)}, Y_k = y, \tilde{Y} = \tilde{y} \right]}{\mathbb{P} \left[X^{(n-1)} = x^{(n-1)}, \tilde{Y} = \tilde{y} \right]} . \end{aligned} \tag{7.21}$$

By (7.20) we have that

$$\begin{aligned} &\mathbb{P} \left[X_n = x_n \mid X^{(n-1)} = x^{(n-1)}, Y_k = y, \tilde{Y} = \tilde{y} \right] \\ &= \mathbb{P} \left[X_n = x_n \mid X^{(n-1)} = x^{(n-1)}, Y_{x_1} = y_{x_1}, \dots, Y_{x_n} = y_{x_n} \right] , \end{aligned}$$

and similarly

$$\begin{aligned} &\mathbb{P} \left[X_n = x_n \mid X^{(n-1)} = x^{(n-1)}, \tilde{Y} = \tilde{y} \right] \\ &= \mathbb{P} \left[X_n = x_n \mid X^{(n-1)} = x^{(n-1)}, Y_{x_1} = y_{x_1}, \dots, Y_{x_n} = y_{x_n} \right] . \end{aligned}$$

Substitution into (7.21) yields

$$\begin{aligned} \mathbb{P} \left[Y_k = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y} \right] &= \frac{\mathbb{P} \left[X^{(n-1)} = x^{(n-1)}, Y_k = y, \tilde{Y} = \tilde{y} \right]}{\mathbb{P} \left[X^{(n-1)} = x^{(n-1)}, \tilde{Y} = \tilde{y} \right]} \\ &= \mathbb{P} \left[Y_k = y \mid X^{(n-1)} = x^{(n-1)}, \tilde{Y} = \tilde{y} \right] . \end{aligned}$$

□

Application of Lemma 7.1 and the independence property of Y_1, \dots, Y_T (given the success parameter θ) yields the following corollary.

Corollary 7.1. *Under the conditions of Lemma 7.1, it follows that*

$$\mathbb{P}[Y_k = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}] = \mathbb{P}[Y_k = y] = \theta^y (1 - \theta)^{1-y}.$$

Therefore, the error labelling of the next transition to be observed is independent of the transitions tested and the error labelling of the rest of transitions. With the above corollary we can now prove the following.

Theorem 7.2. *Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS with $|\mathcal{T}| = T$. For any $x^{(n)} = (x_1, \dots, x_n)$, $y^{(n)} = (y_1, \dots, y_n)$ and $y \in \{0, 1\}$, it follows that*

$$\mathbb{P}\left[Y_{X_{n+1}} = y \mid X^{(n)} = x^{(n)}, Y_{x_1} = y_1, \dots, Y_{x_n} = y_n\right] = \theta^y (1 - \theta)^{1-y},$$

where $\theta \in (0, 1)$ is the error-marking probability of Y_j , for all $j = 1, \dots, T$, as defined in (7.1).

Proof. Let us now consider $\tilde{Y} = (Y_{x_1}, \dots, Y_{x_n})$ and $\mathcal{U} = \{1, 2, \dots, T\} \setminus \{x_1, \dots, x_n\}$. Then, by application of the Law of Total Probability we have that

$$\begin{aligned} & \mathbb{P}\left[Y_{X_{n+1}} = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \sum_{k \in \mathcal{U}} \mathbb{P}\left[Y_k = y, X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right]. \end{aligned} \quad (7.22)$$

We define now \bar{Y} as the complement sequence of \tilde{Y} , i.e., $\text{dom}(\tilde{Y}) \cap \text{dom}(\bar{Y}) = \emptyset$ and $\text{dom}(\tilde{Y}) \cup \text{dom}(\bar{Y}) = \mathcal{T}$. Note that, in particular, Y_k is an element of \bar{Y} . Thus, we can apply the Law of Total Probability and write (7.22) as follows:

$$\begin{aligned} & \mathbb{P}\left[Y_{X_{n+1}} = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \sum_{k \in \mathcal{U}} \sum_{\bar{y}} \mathbb{P}\left[Y_k = y, X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}, \bar{Y} = \bar{y}\right] \times \\ & \quad \times \mathbb{P}\left[\bar{Y} = \bar{y} \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \sum_{k \in \mathcal{U}} \sum_{\bar{y}: y_k = y} \mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}, \bar{Y} = \bar{y}\right] \times \\ & \quad \times \mathbb{P}\left[\bar{Y} = \bar{y} \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right]. \end{aligned} \quad (7.23)$$

By (7.20) we have that X_{n+1} is independent of the marking of the unseen transitions. Therefore,

$$\mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}, \bar{Y} = \bar{y}\right] = \mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right]. \quad (7.24)$$

Since (7.24) does not depend on \bar{y} we can write (7.23) as follows:

$$\begin{aligned} & \mathbb{P}\left[Y_{X_{n+1}} = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \sum_{k \in \mathcal{U}} \mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \sum_{\bar{y}: y_k = y} \mathbb{P}\left[\bar{Y} = \bar{y} \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right]. \end{aligned} \quad (7.25)$$

Note that for the second sum in (7.25) only the sum in the component y_k is fixed (which is equal to y). Therefore, we have that

$$\sum_{\bar{y}: y_k = y} \mathbb{P}\left[\bar{Y} = \bar{y} \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] = \mathbb{P}\left[Y_k = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right]. \quad (7.26)$$

Substitution into (7.25) yields

$$\begin{aligned} & \mathbb{P}\left[Y_{X_{n+1}} = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \sum_{k \in \mathcal{U}} \mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \mathbb{P}\left[Y_k = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right]. \end{aligned} \quad (7.27)$$

Finally, by Corollary 7.1 and by (7.2), we can write (7.27) as

$$\begin{aligned} & \mathbb{P}\left[Y_{X_{n+1}} = y \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \sum_{k \in \mathcal{U}} \mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \mathbb{P}\left[Y_k = y\right] \\ &= \theta^y (1 - \theta)^{1-y} \sum_{k \in \mathcal{U}} \mathbb{P}\left[X_{n+1} = k \mid X^{(n)} = x^{(n)}, \tilde{Y} = \tilde{y}\right] \\ &= \theta^y (1 - \theta)^{1-y}. \end{aligned} \quad (7.28)$$

□

Direct application of the Law of Total Probability in Theorem 7.2 yields the following result.

Corollary 7.3. *Under the conditions of Theorem 7.2, it follows that*

$$\mathbb{P}\left[Y_{X_{n+1}} = y\right] = \theta^y (1 - \theta)^{1-y}.$$

Note that by assuming only that X_{n+1} may depend on all known information from the past, Theorem 7.2 holds for *all possible walking strategies*. Moreover, we have also proved that marking the transitions in advance (by the programmer) is equivalent to marking them on the spot, i.e., when you reach them. This property relies crucially on the assumption that all transitions are i.i.d. error-marked with equal probability.

7.5 Walking Strategies

In this section we introduce two executable examples of walking function updates for labelled transition systems. A walking function update for general labelled transition systems is presented in Section 7.5.1. The main idea is that after each successful run we want to increase the probability of visiting *new* transitions. For that reason, for the next run we may *discard* some already visited parts of the labelled transition system in such a way that the *reduced* system remains a labelled transition systems. We show that after a finite number of updates all the transitions are visited, so that the updating procedure is exhaustive. We develop a similar walking function update for a special subclass of labelled transition systems (acyclic workflow transition systems) in Section 7.5.2.

7.5.1 Walking function update for labelled transition systems

We first give an informal description of a WFU function for LTS and successful runs. Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS with walking function w . If L is a one-path LTS, then we stop after the first successful run since we reach the final state. Therefore, we assume that our system is not one-path. Given a successful run $\sigma = (i, t_1, \dots, s_n, t_n, s_{n+1})$ in L we look for the last state, say s_k , in the sequence σ with at least two outgoing arcs. Since L is not a one-path LTS such a state always exists. We update w to a new walking function w' by setting $w'(t_k) = 0$. By setting $w'(t_k) = 0$ we avoid to run t_k the next time we reach s_k . We do the same for transitions after t_k until we reach either a state with more than one incoming transition (if any) or s_{n+1} (in this situation it is a *final* state). An example of the application of the WFU function is described in Figure 7.3. Suppose that $(s_0, t_0, s_1, t_1, s_2, t_2, s_1)$ is a subpath of a successful run σ in L . We update w by setting $w'(t_2) = 0$ since s_2 is the last state in the run with more than one outgoing arc. Note that this is equivalent to removing t_2 from L . The formal description of the WFU function is

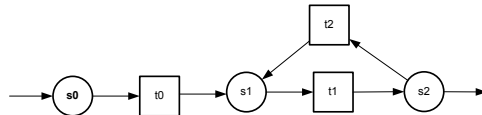


Figure 7.3: Sample of an LTS with a cycle $p = (s_1, t_1, s_2, t_2, s_1)$. Transition t_2 can be removed.

given in Algorithm 1. We now study the validity of the walking function update

Algorithm 1: WFU function for an LTS and a successful run

```

input  :  $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ ,  $\sigma = (s_0, t_0, s_1, t_1, \dots, t_n, s_{n+1})$ ,  $w$ 
output:  $w' = U(w, \sigma)$ 

1 var  $tail : Int$ 
  var  $s : Int \rightarrow \mathcal{S}$ 
  var  $w' : \mathcal{T} \rightarrow [0, 1]$ 
  begin
2    $s_0 := i$ ;  $tail := n$ ;  $w' := w$ 
   while  $(tail \geq 0) \wedge (|s_{tail} \bullet| \leq 1)$  do
3      $tail := tail - 1$ 
4   end
5   while  $(tail \leq n \wedge |\bullet s_{tail}| \leq 1)$  do
6      $w'(t_{tail}) := 0$ ,  $tail := tail + 1$ 
7   end
8 end

```

procedure. Note that the WFU function described in Algorithm 1 assigns, when possible, probability zero to some already visited parts of the corresponding LTS. We refer to the system after the update of the WFU as the *reduced system*.

Definition 7.7. (*Reduced system*) Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS with WFU function U , w a walking function for L and σ a successful run. The reduced system L' with respect to $w' = U(w, \sigma)$ is the triple $(\mathcal{S}', \mathcal{T}', \mathcal{R}')$ such that $\mathcal{T}' = \{t \in \mathcal{T} \mid w'(t) > 0\}$, $\mathcal{S}' = \mathcal{S} \setminus \{s \in \mathcal{S} \mid \bullet s \subset \tilde{\mathcal{T}} \wedge s \bullet \subset \tilde{\mathcal{T}}\}$, where $\tilde{\mathcal{T}} = \{t \in \mathcal{T} \mid w'(t) = 0\}$, and $\mathcal{R}' = \mathcal{R} \cap (\mathcal{S}' \times \mathcal{T}' \times \mathcal{S}')$.

The next result shows that the reduced system after updating w remains an LTS. The main issue is to prove that any state is reachable from the initial one (condition (6) in Definition 7.1).

Theorem 7.4. Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS with WFU function U as defined by Algorithm 1. If w is a walking function for L , then for any successful run σ in L there exists at least one transition t in σ such that $U(w, \sigma)(t) = 0$. Moreover, the reduced system L' with respect to U and w remains an LTS and after a finite number of iterations we test all transitions.

Proof. If L is a one-path LTS (see Definition 7.1), then there is nothing to prove since after a successful run we stop our procedure and we do not update w . Assume that L is not one-path and denote by $\sigma = (i, t_0, s_1, \dots, t_n, s_{n+1})$ a successful run in L . Since L is not a one-path LTS, there exists a state s_k in σ , with $0 \leq k \leq n$, where $s_0 = i$, such that $|s_k \bullet| > 1$. According to Algorithm 1 we choose the last state in the sequence σ with more than one outgoing arc and let it be s_k . We consider the path $p = (s_k, t_k, s_{k+1}, \dots, t_n, s_{n+1})$. We update w by setting $w'(t_k) = w'(t_{k+1}) =$

$\dots = w'(t_n) = 0$. We now prove that the reduced system, denoted by L' , remains an LTS. It suffices to verify that every state $x \in L'$ and not in p is reachable from the initial one. Note that there always exists a path v from i to x in L . We consider the following cases:

1. If p and v have no states in common, then v is also a path in L' and we are done.
2. Now assume that p and v have at least one state in common.
 - (a) Suppose first that s_{n+1} is a final state, i.e., $|s_{n+1} \bullet| = 0$. Obviously x is not reachable from s_{n+1} and since s_k is the only state in p with two outgoing arcs in σ , s_k is the only common state of p and v . Therefore, v is a path from i to x via s_k but not via t_k . Thus, v is also a path in L' .
 - (b) Cycle case: suppose now that σ contains a cycle, i.e., s_{n+1} is observed twice in σ and $|s_{n+1} \bullet| > 0$.
 - i. If s_k is also in v , then two cases are possible. Either v is a path from i to x via s_k but not via t_k , in which case v is also a path in L' , or v is a path from i to x via t_k which means (since s_k is the last state in σ with two outgoing arcs) that v passes through s_{n+1} . Therefore, there exists a path v' from i to x via s_{n+1} (which does not include the cycle) that is also a path in L' .
 - ii. If s_k is not in v , then we have the following situation. Since p and v have some nodes in common and since s_k is the only state in p with at least two outgoing arcs, the intersection of p and v must start in a state, say s_ℓ , with more than one incoming arc (i.e., $|\bullet s_\ell| \geq 2$) that occurs in the sequence of p after s_k (i.e., $k + 1 \leq \ell \leq n + 1$). Moreover, $(s_\ell, t_\ell, \dots, t_n, s_{n+1})$ is a subsequence of both p and v . In any case, s_{n+1} is also a state in v . Hence, there exists a path v' in L' from i to x via s_{n+1} .

For the last statement in the theorem recall that we discard at least one transition after a successful run. Failure runs may not reduce L , but since the number of error marked transitions is finite, after a finite number of runs we visit all the transitions and thus an exhaustive procedure is defined. \square

Note that σ must be a successful run, otherwise Theorem 7.4 is not true. This condition is illustrated in Figure 7.4. Suppose that $(s_0, t_0, s_1, t_1, s_2)$ is a subpath of a failure run σ in L , where t_2 is an error marked transition. According to Algorithm 1, we would update the walking function w by setting $w'(t_1) = 0$. However, the reduced system is no longer an LTS because s_2, s_3, \dots would be unreachable.

7.5.2 Walking function update for acyclic workflow transition systems

We now present a WFU function update for a special type of LTS. Since this subclass is in fact a special class of *workflow nets*, we call it workflow transition systems.

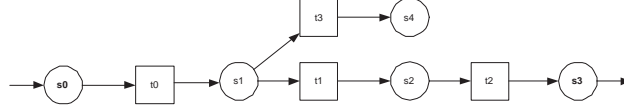


Figure 7.4: Sample of an LTS where $p = (s_0, t_0, s_1, t_1, s_2)$ is a subpath of a failure run. The WFU function given by Algorithm 1 cannot be applied.

Definition 7.8 (Workflow transition system). A workflow transition system (*WTS*) is an LTS with the additional requirements that there is a unique final state f and that for every state $s \neq f$ there is a path from s to f .

We first give an informal description of the WFU function for *acyclic* (see Definition 7.1) workflow transition systems. Let $W = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an acyclic WTS with walking function w . We assume that W is not one-path since testing a one-path WTS is trivial. Given a successful run σ in W we look for the first state, say s , in σ with at least two outgoing arcs. Since W is not one-path such a state always exists. Setting s as a marker, called “head”, we move forward through σ . If the next state has exactly one incoming and one outgoing arc, then we move to the following state. If we reach a state, say s' , with at least two outgoing arcs but only one incoming, then we set s' as “head”. We continue the same procedure until we find a state, say \tilde{s} , with at least two incoming arcs. Such a state always exists because there is exactly one final state, there are no cycles and the final state can be reached from any other state. We update w to a new walking function w' by setting $w'(t) = 0$, for all t in σ between s' and \tilde{s} . When this update has been done we continue moving forward through σ looking for a new “head” and applying the same procedure until we reach the final state f . Note that the workflow property is guaranteed because of acyclicity. An example of the application of the WFU function is shown in Figure 7.5. Suppose that $\tilde{\sigma} = (b, t_0, s_1, t_1, s_2, t_2, e)$ is a subpath of a successful run σ in W . We update w by setting $w'(t_0) = 0$. Note that this is equivalent to removing t_0 from W . Similarly, in the situation illustrated in Figure 7.6, we update w by setting

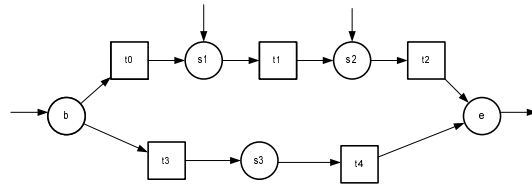


Figure 7.5: Sample of an acyclic WTS where $\tilde{\sigma} = (b, t_0, s_1, t_1, s_2, t_2, e)$ is a subpath of a successful run. Since s_1 and s_2 have only one outgoing arc, transition t_0 can be removed.

$w'(t_0) = 0$ and $w'(t_2) = 0$. If in both cases (b, t_3, s_3, t_4, e) was a subpath of σ , then we would update w by setting $w'(t_3) = 0$ and $w'(t_4) = 0$. Note that the update procedure is valid only for an acyclic WTS. Suppose that $(b, t_0, s_1, \dots, s_4, t_4, b)$ is a

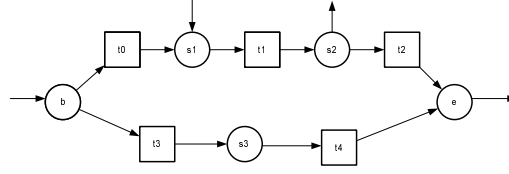


Figure 7.6: Sample of an acyclic WTS where $\tilde{\sigma} = (b, t_0, s_1, t_1, s_2, t_2, e)$ is a subpath of a successful run. Since s_1 has only one outgoing arc and s_2 has two outgoing arcs, transitions t_0 and t_2 can be removed.

cycle as it is shown in Figure 7.7, and suppose it is also a subpath of σ . According to Algorithm 2 we would update w by setting $w'(t_0) = 0$. However, the reduced system is no longer a WTS because t_5 could not be executed. The formal description of

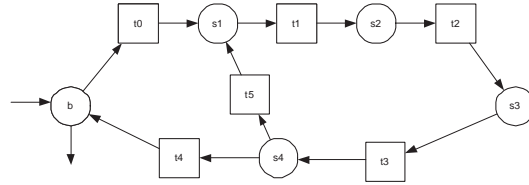


Figure 7.7: Sample of a WTS where $p = (b, t_0, s_1, \dots, s_4, t_4, b)$ is a cycle. The WFU function given by Algorithm 2 cannot be applied.

this WFU function is given in Algorithm 2. Similarly to the general case, we now study the validity of the procedure for an acyclic WTS. Reduced WTS are defined in a similar way as in Definition 7.7, therefore we skip a formal definition here.

Theorem 7.5. *Let $W = (S, T, \mathcal{R})$ be an acyclic WTS with WFU function U as defined by Algorithm 2. If w is a walking function for W , then for a successful run σ in W there exists at least one transition t in σ such that $U(w, \sigma)(t) = 0$. Moreover, the reduced system W' with respect to U and w remains a WTS and after a finite number of updates we visit all transitions.*

Note that the update procedure may be applied several times moving forward through a successful run until the final state is reached. However we present a proof for the case where the system is reduced only once. Given this proof, the proof for multiple reductions is straightforward.

Proof. If W is a one-path WTS, then there is nothing to prove since after a successful run we stop our procedure and we do not update w . Assume that W is not a one-path WTS and denote by $\sigma = (i, t_0, s_1, \dots, t_n, f)$ a successful run in W . Since W is not a one-path WTS, there exists a state s_k in σ , with $0 \leq k \leq n$, where $s_0 = i$, such that $|s_k \bullet| > 1$. We can assume without loss of generality that s_k is the “head” marker in Algorithm 2 and let s_ℓ , with $k < \ell \leq n$, be the first state in σ after s_k with

Algorithm 2: WFU function for an acyclic WTS and a successful run

```

input :  $L = (\mathcal{S}, \mathcal{T}, \mathcal{R}), \sigma = (s_0, t_0, s_1, t_1, \dots, t_n, s_{n+1}), w$ 
output:  $w' = U(w, \sigma)$ 

1 var  $head, current : Int$ 
  var  $s : Int \rightarrow \mathcal{S}$ 
  var  $w' : \mathcal{T} \rightarrow [0, 1]$ 
  begin
2    $s_0 := i; s_{n+1} := f; head := 0; current := 0; w' := w$ 
   while ( $current \leq (n + 1)$ ) do
3     if ( $|\bullet s_{current}| > 1$ ) then
4       for ( $x = head$  to  $current - 1$ ) do  $w(t_x) := 0$ 
5     endif
6     if ( $|s_{current}\bullet| > 1$ ) then
7        $head := current$ 
8     endif
9      $current := current + 1$ 
10  end
11 end

```

more than one incoming arc. We consider the path $p = (s_k, t_k, s_{k+1}, \dots, t_{\ell-1}, s_\ell)$. Note that p is a linear subpath of σ . Therefore, we update w by setting $w'(t_k) = \dots = w'(t_{\ell-1}) = 0$. We now prove that the reduced system, denoted by W' , remains a WTS. Consider an arbitrary state x in W that is not in p . Therefore, x is also a state in W' and there exists a path v from i to f via x in W . If p is not a subpath of v , then v is also a path in W' and we are done. Suppose now that p is a subpath of v . Either p is a subpath from i to x or from x to f . Suppose that it is a subpath from i to x . Since p is a linear path and passes via s_ℓ to x and $|\bullet s_\ell| > 1$, there exists at least one other path from i to s_ℓ such that p is not a subpath of it (due to acyclicity). Assume now that p is a subpath from x to f . Since $|s_k\bullet| > 1$ there exists at least another path from s_k to f such that p is not a subpath of it (again due to acyclicity). Therefore, W' is a WTS. The final statement follows as in the proof of Theorem 7.4. \square

To end this section, we now illustrate with a simple example the advantage of using Algorithm 2 for acyclic WTS. We have shown in Section 7.5.1 that Algorithm 1 is valid for general LTS. Therefore, if we do not have any information about whether the system is an acyclic WTS or not, then we apply Algorithm 1. However, if we know that the system is an acyclic WTS it is more efficient to use Algorithm 2 since after a successful run it reduces at least the same number of transitions as Algorithm 1. This is depicted in Figure 7.8. Suppose the path $\sigma = (i, t_0, s_1, t_1, s_2, t_2, f)$ is a successful run. According to Algorithm 1 we update the walking function w by setting $w'(t_2) = 0$, *i.e.*, the system is reduced by one transition. Nevertheless, if we apply Algorithm 2, then we update w by setting $w'(t_1) = 0$ and $w'(t_2) = 0$,

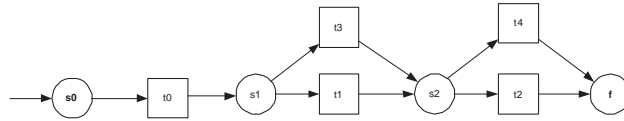


Figure 7.8: Sample of an acyclic WTS where $\sigma = (i, t_0, s_1, t_1, s_2, t_2, f)$ is a successful run. Transitions t_1 and t_2 can be removed by Algorithm 2. Only t_2 can be removed by Algorithm 1.

reducing thus the system by two transitions.

7.6 Common notation

All the concepts introduced throughout this chapter will be used in the following chapters. For that reason the most common notation is summarized in Table 7.2 so that the reader may refer back to this table should confusion about notation arise.

Notation	Definition
$L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$	Labelled Transition System (LTS)
s	state: element of \mathcal{S}
i	initial state of L
f	final state of L
t	transition: element of \mathcal{T}
T	total number of transitions in \mathcal{T}
N	initial number of error-marked transitions
θ	error-marking probability
$M(\cdot)$	error-marking function
Y_j	marking of transition j
X_j	tested transition in j^{th} step (given by the walking strategy)
n	number of observed transitions
m_n	number of observed error-marked transitions
σ	run in an LTS
Σ	set of all runs in an LTS
$w(\cdot)$	walking function
$U(\cdot, \cdot)$	Walking Function Update (WFU) function

Table 7.2: Common notation in model-based testing.

CHAPTER 8

STATISTICAL CERTIFICATION PROCEDURES

In this chapter we present two certification procedures for the testing framework introduced in Chapter 7. In that chapter we defined two random processes of interest, namely the *marking* process and the *testing* process. In this chapter we consider the process where only the transitions observed for the first time are taken into account. We will refer to this as the *embedded* process. As shown in Corollary 7.1, the output (error-free or error-marked) of the next transition to be observed is independent of the output of the transitions already observed. Therefore, the embedded process is *independent* from the way that the system is tested. Thus, we have reduced model-based testing to black-box testing, although in this case we can keep track of error-free and error-marked transitions which is impossible in black-box testing. We also provide two statistical stopping rules, that are independent of the underlying way of walking through the system, which allows us to stop earlier with a certain statistical reliability. In Section 8.1 we present our first stopping rule, which is based on the probability of having a certain number of remaining error-marked transitions when we decide to stop testing. Like in Chapter 5, we also define the *risk* of taking a wrong decision and prove that when we stop testing the global risk can be controlled. Our second stopping rule is based on the survival probability of the system and it is presented in Section 8.2. Since we now have the notion of *transition* (see Definition 7.1), we can define the survival probability as the probability of consecutively observing a certain number of error-free transitions. In this case, we also prove that the global risk can be controlled. Finally, in Section 8.3 we illustrate our whole approach with an example based on the model of the generic medical workflow of a hospital presented in Section 7.2.

8.1 Certification procedure based on the number of remaining error-marked transitions

Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS, as in Definition 7.1, with $|\mathcal{T}| = T$. Consider the *error-marking process* $Y^{(n)} = (Y_1, \dots, Y_n)$ and the *testing process* $X^{(n)} = (X_1, \dots, X_n)$, as defined in Section 7.3 and Section 7.4, respectively. The process $Z^{(n)} = (Z_1, \dots, Z_n)$, where $Z_j = Y_{X_j}$, for all $j = 1, \dots, n$, is called the *embedded* process. Note that the embedded process only considers the transitions observed for the first time. With the above notation $x^{(n)} = (x_1, \dots, x_n)$ is the *test history* and $\sum_{j=1}^n z_j$ is the number of *observed* error-marked transitions. We have shown in Corollary 7.3 that the random variable Z_j is a Bernoulli random variable with success parameter θ (the marking probability). We now compute a statistical stopping rule that certifies with high confidence that at most $k \geq 0$ error-marked transitions are left in the system when we decide to stop testing. If we fix a *reliability level*, denoted by $1 - \delta$, for some $\delta \in (0, 1)$, and N denotes the random variable representing the total number

of error-marked transitions in the system, then our release procedure consists of observing the minimal number of *new* transitions, denoted by n , such that

$$R_{n,k}(z) = \mathbb{P} \left[\sum_{j=1}^n z_j \leq N \leq k + \sum_{j=1}^n z_j \mid Z^{(n)} = z \right] \geq 1 - \delta. \quad (8.1)$$

The above stopping rule defines a *stopping time* for the embedded process as we explain now. In general, a stopping time for a sequence of random variables Z_1, Z_2, \dots is an integer-valued random variable, denoted by S , such that the event $\{S = s\}$ belongs to the σ -algebra generated by Z_1, \dots, Z_s , for all $s = 1, 2, \dots$ (cf. [Grimmett and Stirzaker \(1988\)\[p.487\]](#)). In this case

$$S = \min \left\{ s \mid R_{s,k}(Z^{(s)}) \geq 1 - \delta, \forall 1 \leq j \leq s - 1 : R_{j,k}(Z^{(j)}) < 1 - \delta \right\},$$

is a stopping time for the embedded process. Note that in order to compute (8.1) we only need the posterior probability distribution of N . In case of a Binomial or a Poisson prior distribution for N , this can be computed using (7.10) and (7.19), respectively. Like in Chapter 5, we are also interested in the *risk* due to a wrong decision. In this case, it can be defined as follows. Our procedure certifies that if $\sum_{j=1}^n z_j$ error-marked transitions have been observed in a sample of n transitions, then, with probability at least $1 - \delta$, there are at most k error-marked transitions left in the system. Therefore, the *risk after observing n transitions* is the probability of having more than k remaining error-marked transitions in the system and can be defined as

$$G_{n,k} = \mathbb{P} \left[N > k + \sum_{j=1}^n Z_j, \forall 1 \leq j \leq n - 1 : R_{j,k}(Z^{(j)}) < 1 - \delta, R_{n,k}(Z^{(n)}) \geq 1 - \delta \right]. \quad (8.2)$$

Note that the risk $G_{n,k}$ is the probability that N exceeds the limit that the stopping rule prescribes when stopping after observing n transitions. Taking the sum over all n , we define the *global risk*

$$G_k = \sum_{n>0} G_{n,k}. \quad (8.3)$$

As done with the procedures described in Chapter 5, we are interested in finding a condition on the local risk that guarantees that the global risk is bounded. The next theorem shows the desired result.

Theorem 8.1. *For any $k \geq 0$, $\delta \in (0, 1)$ and $n \geq 0$ such that the stopping criterion is $R_{n,k}(z) \geq 1 - \delta$, it follows that $G_k \leq \delta$.*

Proof. We can apply the Law of Total Probability and write the risk after observing n transitions in (8.3) as follows:

$$\begin{aligned}
G_{n,k} &= \sum_z \mathbb{P} \left[N > k + \sum_{j=1}^n z_j, \forall 1 \leq j \leq n-1 : R_{j,k}(z^{(j)}) < 1 - \delta, \right. \\
&\quad \left. R_{n,k}(z^{(n)}) \geq 1 - \delta \mid Z^{(n)} = z \right] \mathbb{P} \left[Z^{(n)} = z \right] \\
&= \sum_{z \in \mathcal{A}_n} \mathbb{P} \left[N > k + \sum_{j=1}^n z_j \mid Z^{(n)} = z \right] \mathbb{P} \left[Z^{(n)} = z \right],
\end{aligned} \tag{8.4}$$

where

$$\mathcal{A}_n = \{z \mid R_{n,k}(z) \geq 1 - \delta, \forall 1 \leq j \leq n-1 : R_{j,k}(z) < 1 - \delta\},$$

for any $n \geq 0$. Since we assume that

$$\mathbb{P} \left[\sum_{j=1}^n z_j \leq N \leq k + \sum_{j=1}^n z_j \mid Z^{(n)} = z \right] \geq 1 - \delta,$$

then also

$$\mathbb{P} \left[N \leq k + \sum_{j=1}^n z_j \mid Z^{(n)} = z \right] \geq 1 - \delta.$$

Therefore, we get

$$G_{n,k} \leq \delta \sum_{z \in \mathcal{A}_n} \mathbb{P} \left[Z^{(n)} = z \right]. \tag{8.5}$$

Thus, for the global risk in (8.3) we have that

$$\begin{aligned}
G_k &\leq \delta \sum_{n \geq 1} \sum_{z \in \mathcal{A}_n} \mathbb{P} \left[Z^{(n)} = z \right] = \delta \sum_{n \geq 1} \mathbb{P} \left[Z^{(n)} \in \mathcal{A}_n \right] \\
&= \delta \mathbb{P} \left[\bigcup_{n \geq 1} Z^{(n)} \in \mathcal{A}_n \right] = \delta,
\end{aligned} \tag{8.6}$$

where the last equality holds since \mathcal{A}_n are disjoint for all $n \geq 1$ and testing is always stopped (the probability on the right-hand side of (8.6) is equal to 1 since it is the sum over all $n \geq 1$). \square

8.2 Certification procedure based on the survival probability

In the line of the certification procedure presented in Section 5.3, we present in this section a certification procedure based on the *survival probability* of the system. As mentioned in the previous section, we consider the embedded process. With the notation used in Section 8.1, we define the *k-step conditional survival probability* (for the embedded process) as follows:

$$S_{n,k}(z) = \mathbb{P} \left[\bigcap_{\ell=1}^k \{Z_{n+\ell} = 0\} \mid Z^{(n)} = z \right], \tag{8.7}$$

for all $0 \leq k \leq T - n$. Note that the above probability is the probability of observing k new consecutive error-free transitions given that we already know the output (error-free or error-marked) of n transitions. Note also that the embedded process is *independent* from the way the system is tested since it only considers the new transitions but not when they are discovered (it is defined for one or more runs since just the next new transition counts). Therefore, we may observe many transitions before we will encounter a new error-marked transition. If we fix a reliability level $1 - \delta$, then our release procedure consists of finding the minimal number of new transitions such that $S_{n,k}(z) \geq 1 - \delta$. In this case, we can define the stopping time

$$S = \min \left\{ s \mid S_{s,k}(Z^{(s)}) \geq 1 - \delta, \forall 1 \leq j \leq s - 1 : S_{j,k}(Z^{(j)}) < 1 - \delta \right\},$$

and the *risk after observing n transitions*

$$H_{n,k} = \mathbb{P} \left[\bigcup_{\ell=1}^{\min\{k, T-n\}} \{Z_{n+\ell} = 1\}, \forall 1 \leq j \leq n - 1 : S_{j,k}(Z^{(j)}) < 1 - \delta, \right. \\ \left. S_{n,k}(Z^{(n)}) \geq 1 - \delta \right]. \quad (8.8)$$

Taking the sum over all $n \geq 0$, we define the *global risk*

$$H_k = \sum_{n>0} H_{n,k}. \quad (8.9)$$

As in the previous section, the next theorem shows that if the k -step conditional survival probability is bounded, then the global risk is also kept under control. The proof is very similar to the one of Theorem 8.1.

Theorem 8.2. *For any $k \geq 0$, $\delta \in (0, 1)$ and $n \geq 0$ such that the stop criterion is $S_{n,k}(z) \geq 1 - \delta$, it follows that $H_k \leq \delta$.*

Proof. We can apply the Law of Total Probability and write the risk after observing n transitions in (8.8) as follows:

$$H_{n,k} = \sum_z \mathbb{P} \left[\bigcup_{\ell=1}^{\min\{k, T-n\}} \{Z_{n+\ell} = 1\}, \forall 1 \leq j \leq n - 1 : S_{j,k}(z^{(j)}) < 1 - \delta, \right. \\ \left. S_{n,k}(z^{(n)}) \geq 1 - \delta \mid Z^{(n)} = z \right] \mathbb{P} [Z^{(n)} = z] \\ = \sum_{z \in \mathcal{B}_n} \mathbb{P} \left[\bigcup_{\ell=1}^{\min\{k, T-n\}} \{Z_{n+\ell} = 1\} \mid Z^{(n)} = z \right] \mathbb{P} [Z^{(n)} = z], \quad (8.10)$$

where

$$\mathcal{B}_n = \{z \mid S_{n,k}(z) \geq 1 - \delta, \forall 1 \leq j \leq n - 1 : S_{j,k}(z) < 1 - \delta\},$$

for any $n \geq 0$. Note that, for all $z \in \mathcal{B}_n$, it follows that

$$\mathbb{P} \left[\bigcup_{\ell=1}^{\min\{k, T-n\}} \{Z_{n+\ell} = 1\} \mid Z^{(n)} = z \right] = 1 - S_{n,k}(z).$$

Since we assume that $S_{n,k}(z) \geq 1 - \delta$, we get

$$H_{n,k} \leq \delta \sum_{z \in \mathcal{B}_n} \mathbb{P} \left[Z^{(n)} = z \right]. \quad (8.11)$$

Therefore, for the global risk in (8.9) we have that

$$\begin{aligned} H_k &\leq \delta \sum_{n \geq 1} \sum_{z \in \mathcal{B}_n} \mathbb{P} \left[Z^{(n)} = z \right] = \delta \sum_{n \geq 1} \mathbb{P} \left[Z^{(n)} \in \mathcal{B}_n \right] \\ &= \delta \mathbb{P} \left[\bigcup_{n \geq 1} Z^{(n)} \in \mathcal{B}_n \right] = \delta, \end{aligned} \quad (8.12)$$

where the last equality holds since \mathcal{B}_n are disjoint for all $n \geq 1$ and testing is always stopped (the probability on the right-hand side of (8.12) is equal to 1 since it is the sum over all $n \geq 1$). \square

We are interested in obtaining an expression for the k -step conditional survival probability depending on the posterior distribution of the number of error-marked transitions in the system. The following lemma gives the desired result.

Lemma 8.1. *Let $Z^{(n)} = (Z_1, \dots, Z_n)$ be the random variables representing the error-marking of the transitions given by the test history of the process. If $m_n = \sum_{j=1}^n z_j$, then the k -step conditional survival probability after observing n transitions is given by*

$$S_{n,k}(z) = \sum_{j=m_n}^{T-(n-m_n+k)} \frac{\binom{T-n-j+m_n}{k}}{\binom{T-n}{k}} \mathbb{P}[N = j \mid Z^{(n)} = z].$$

Proof. First note that if we have observed $n - m_n$ error-free transitions and we require also that the next k new transitions to be observed are error-free, then the maximum number of possible error-marked transitions in the system is given by $T - (n - m_n + k)$. Thus, application of the definition of conditional probability and the Law of Total Probability to (8.7) yields

$$S_{n,k}(z) = \sum_{j=m_n}^{T-(n-m_n+k)} \mathbb{P} \left[\bigcap_{\ell=1}^k \{Z_{n+\ell} = 0\} \mid Z^{(n)} = z, N = j \right] \mathbb{P} \left[N = j \mid Z^{(n)} = z \right]. \quad (8.13)$$

Finally, $\mathbb{P} \left[\bigcap_{\ell=1}^k \{Z_{n+\ell} = 0\} \mid Z^{(n)} = z, N = j \right]$ can be computed as follows. First note that this is the probability of observing k consecutive error-free transitions, given the sequence $Z^{(n)} = z$ and that there are $N = j$ error-marked transitions in a population of transitions of size T . Thus, there are $T - n$ unknown transitions from which $j - m_n$ are error-marked. Therefore, a similar derivation as for a hypergeometric experiment yields

$$\mathbb{P} \left[\bigcap_{\ell=1}^k \{Z_{n+\ell} = 0\} \mid Z^{(n)} = z, N = j \right] = \frac{\binom{T-n-j+m_n}{k}}{\binom{T-n}{k}}. \quad (8.14)$$

Substitution into (8.13) yields the desired result. \square

Note that it is possible that before observing a *new* transition we observe many *old* ones and to observe a new one may take a long time. Thus, the stopping rule does not mean to survive k steps but k new transitions which may be much more than k steps. Note now that if we know the posterior distribution of N , then we can compute the k -step conditional survival probability of the system according to the above lemma. For example, if we consider that N follows a Binomial distribution (with random marking probability Θ and density function f), like in Section 7.3.1, then we can use (7.10) to write

$$S_{n,k}(z) = \sum_{j=m_n}^{T-n-k+m_n} \binom{T-n-k}{j-m_n} \frac{\int_0^1 \theta^j (1-\theta)^{T-j} f(\theta) d\theta}{\int_0^1 \theta^{m_n} (1-\theta)^{n-m_n} f(\theta) d\theta}. \quad (8.15)$$

If we do not want to assume any prior knowledge about Θ , we can consider that it has the Uniform distribution on $(0, 1)$ as prior distribution, and thus $f(\theta) = 1$, for all $\theta \in (0, 1)$. In this case, the k -step conditional survival probability is given by

$$S_{n,k}(z) = \frac{n+1}{T+1} \binom{n}{m_n} \sum_{j=m_n}^{T-n-k+m_n} \frac{\binom{T-n-k}{j-m_n}}{\binom{T}{j}}. \quad (8.16)$$

Note that the embedded process with Binomial distribution for the number of error-marked transitions is very easy to simulate. A simulation here consists of generating an array of length T where each element can have two possible values, representing an error-free or an error-marked transition. The distribution of the number of errors in the array follows a Binomial distribution with parameters T and θ . After the array is generated, we can compute the stopping rule for different values of k . In Table 8.1 we show the mean number of transitions and the mean number of error-marked transitions left (and the corresponding standard deviations) as a result of

the the application of the survival probability stopping rule (for $1 - \delta = 0.90$) for several values of k considering an array of length $T = 500$ when Θ is sampled from an Uniform distribution on the interval $(0, 0.1)$ over 100 replications. As one may

Number of survivals	Number of Transitions		Errors Left	
	Mean	Std. Dev.	Mean	Std. Dev.
$k = 1$	42.94	71.94	18.09	3.38
$k = 2$	180.54	215.55	12.96	8.63
$k = 3$	293.16	219.17	8.66	9.00
$k = 4$	354.20	204.86	6.21	8.59
$k = 5$	393.44	186.81	4.59	7.96

Table 8.1: Mean and standard deviation of number of new transitions and number of error-marked transitions left as a result of the the application of the survival probability stopping rule with reliability $1 - \delta = 0.90$ for several values of k when $T = 500$ and Θ is sampled from an Uniform distribution on $(0, 0.1)$ over 100 replications.

expect, the number of observed transitions needed to stop testing increases with the value of k while the number of remaining error-marked transitions after stopping decreases.

8.3 Practical application

In this section we illustrate our whole approach with a real example using the model of the generic medical workflow of a hospital presented in Section 7.2. Although the model used in this example is small, it is enough to illustrate the whole procedure and to show how the approach will work in large cases. We have two main goals in this section:

- (i) Study the effect of the reduction algorithm described in Section 7.5 in the test procedure.
- (ii) Study the performance of the stopping rules described in sections 8.1 and 8.2.

For comparison purposes we introduce two other stopping rules: the *exhaustive* rule and the *error-free* rule. The exhaustive stopping rule says that when all transitions are observed, testing can be stopped and it is used for (i) above. The error-free stopping rule is used for (ii) above and it says that testing can be stopped when all error-marked transitions are found. Note that this is only possible in the experiment but not in reality since the number of faults in the system is unknown.

8.3.1 General setup

We have uniformly distributed 5 errors over the system described in Figure 7.2, i.e., we have given a special label to 5 transitions where all the transitions had the same

probability of having this label. Note that this is a fairly high number of error-marked transitions considering that the total number of transitions in this example is 22. We have performed 20 *paired* experiments using the reduction procedure described in Section 7.5 (further denoted by R) and non-reduction (denoted by NR). Paired means here that the distribution of the error-marked transitions is fixed beforehand and two experiments, one using reduction and one without reduction, are performed for the same configuration of error-marked transitions. We consider the following stopping rules

- $R_1(k, \delta)$: remaining error-marked transitions rule. This denotes the stopping rule described in Section 8.1.
- $R_2(k, \delta)$: survival probability. This denotes the stopping rule described in Section 8.2.
- R_3 : error-free rule.
- R_4 : exhaustive rule.

Note that R_1 and R_2 are *practical* stopping rules while R_3 and R_4 are *theoretical* stopping rules for comparison as explained in the introduction of this section. We have recorded the number of runs provided by each stopping rule and we display the mean and the standard deviation. We have also computed the number of error-marked transitions left due to an early stop. Due to the small number of transitions in the example, the Poisson distribution for the total number of error-marked transitions in the system is not considered (see Section 7.3.2 for details). Therefore, we only consider a Binomial distribution for the number of error-marked transitions. In particular we assume that the prior distribution for the error-marking probability, denoted by Θ in Section 8.1, is Uniform on $(0, 1)$. In this case, the survival probability is given by (8.16). The reliability level is fixed to $1 - \delta = 0.90$. However, for this reliability level, the $R_1(k, 0.10)$ stopping rule can only be computed for $k = 1$, since for any $k > 1$, it follows that $R_{n,k}(z) < 0.90$, for all $n < T$. Thus, the stopping condition (8.1) holds only for $n = T$, which means exhaustiveness. Therefore, using (8.1), we have calculated the probability of having *at most one remaining error-marked transition*. The same remark applies to the $R_2(k, 0.10)$ stopping rule (it can only be computed for $k = 1$).

8.3.2 Performance of the stopping rules

We now study the performance of the stopping rules mentioned in Section 8.3.1. All the results, using reduction (R) and non-reduction (NR), are shown in Table 8.2. The main conclusions we can extract are the following:

1. The reduction algorithm is efficient (see row R_4): the algorithm is reducing exhaustive search from 52.75 to 15.60 mean number of runs.
2. The $R_1(1, 0.10)$ stopping rule is efficient: it is almost at the same level of error-freeness (R_3). However, it has a small error, namely the remaining number

	Number of Runs				Errors Left	
	Mean		Std. Dev.		Mean	
	R	NR	R	NR	R	NR
$R_1(1, 0.10)$	12.75	32.25	1.33	12.00	0.60	0.45
$R_2(1, 0.10)$	13.50	38.30	4.84	25.03	0.75	1.25
R_3	13.25	35.40	1.75	20.25	-	-
R_4	15.60	52.75	1.24	22.02	-	-

Table 8.2: Mean number of runs, standard deviation of runs and mean number of error-marked transitions left, comparing our reduction procedure and the general procedure, for several stopping rules.

of error-marked transitions, on average 0.60 (R) and 0.45 (NR). In any case, it is on average smaller than the fixed bound (the rule says at most $k = 1$ error-marked transition left). Moreover, the $R_1(1, 0.10)$ stopping rule reduces exhaustive search in about 40%.

3. The $R_2(1, 0.10)$ stopping rule is efficient: it is also almost at the same level of error-freeness (R_3) but with an average number of remaining error-marked transitions of 0.75 (R) and 1.25 (NR). It was observed during testing the system for a fixed configuration of error-marked transitions that the stopping rule performed in the following way: either all the error-marked transitions were discovered (exhaustive) or none of them were discovered. This is because the number of transitions in the system is small. Thus, as soon as the first error-marked transition is discovered, the procedure never stops before observing all transitions. If in the beginning of testing we observe 8 consecutive error-free transitions, then the procedure stops, leaving all the error-marked transitions undetected.

The *efficiency of the reduction algorithm* described in Section 7.5 can be statistically quantified via hypothesis testing as follows. Let us first consider the random variables W_R and W_{NR} representing the *number of runs to stop testing using the R_4 stopping rule (exhaustive) when testing is performed with and without reduction*, respectively. If μ_R and μ_{NR} denote the corresponding mean of W_R and W_{NR} , then we are interested in testing

$$H_0 : \mu_R - \mu_{NR} = 0$$

against the alternative

$$H_1 : \mu_R - \mu_{NR} < 0 .$$

In case we do not reject H_0 , we can conclude that the reduction algorithm does not have any influence on exhaustive testing in terms of the expected number of runs to reach exhaustiveness. Since the observations are collected in pairs, we can think of performing the usual paired-sample t-test (see e.g. [Ross \(2005\)](#)[Section 10.5])

for the hypothesis that the two population means compared are equal. This test requires a normal distribution for the data. The Kolmogorov test and the Shapiro-Wilk test (see e.g. Sheskin (2004)[Chapter 7] for details on both tests) have been performed to test the normality of the data and in case where testing is performed using reduction the conclusion was that we reject the hypothesis that the data came from a normally distributed population. In this situation, when we cannot assume a normal distribution for the data, we can use the Wilcoxon signed-rank test (cf. Ross (2005)[Section 14.3] and Sheskin (2004)[Chapter 6]). The Wilcoxon signed-rank test is a non-parametric test that does not require any assumptions about the distribution of the data and can be used to test the hypothesis of equal means. The test has been performed for a significance level $\alpha = 0.05$ and its result yields to rejection of the null hypothesis (p -value $\ll \alpha$). This confirms what we observed in Table 8.2 (row R_4): the reduction algorithm is efficient since it reduces exhaustive searching in terms of mean number of runs.

In a similar way we can statistically test the *efficiency of the R_1 and R_2 stopping rules*. Let us now consider the random variable W_j representing the *number of runs to stop testing using the R_j stopping rule and our reduction algorithm*, for $j = 1, 2, 3$. Note also that, besides the above random variables, we could also define the analogous random variables without using the reduction algorithm. However, we do not consider them here. In a similar way, if μ_j denotes the mean of W_j , then we are interested in testing

$$H_0 : \mu_\ell - \mu_j = 0 ,$$

for all $\ell, j = 1, 2, 3, \ell \neq j$, against a two-sided alternative in this case. If we do not reject H_0 , then we can conclude that the expected performance of the two stopping rules being compared is the same. Also in this case the non-normality of the observations does not allow to perform the paired-sample t-test. Therefore, the Wilcoxon signed-rank test has also been used here. The tests have been performed for a significance level $\alpha = 0.05$ and their results, given by the corresponding p -value, are shown in Table 8.3. We observe that in all cases the result of the test yields to

Stopping rules comparison			
H_0	Test Statistic	p -value	Conclusion
$\mu_1 = \mu_3$	-0.97	0.32	Do not reject
$\mu_2 = \mu_3$	-0.84	0.39	Do not reject
$\mu_1 = \mu_2$	-1.81	0.07	Do not reject

Table 8.3: Comparison of several stopping rules.

non-rejection of the null hypothesis (p -value larger than 0.05). The first two tests confirm what we expected from Table 8.2: the $R_1(1, 0.10)$ and $R_2(1, 0.10)$ rules are *efficient* since their expected performance is equal to the expected performance of the R_3 rule (error-freeness). From the third test we can conclude that the mean number of runs to stop testing using the $R_1(1, 0.10)$ stopping rule is the same for the $R_2(1, 0.10)$ rule. However, the p -value is very small (almost 0.05). Moreover,

as observed in Table 8.2, the $R_1(1, 0.10)$ rule seems to perform better in this case since the number of runs, the standard deviation and the number of error-marked transitions left are smaller than for the $R_2(1, 0.10)$ rule.

CHAPTER 9

TESTING THE TEST PROCEDURE

In the last two chapters we have developed a test procedure for software systems assuming that these can be modelled as a special kind of *labelled transition systems*. We can distinguish two main parameters in our test procedure, namely the *walking strategy* and the *stopping rule*. Different walking strategies and stopping rules are described in Section 7.5 and Chapter 8, respectively. In this chapter we discuss how these two parameters may influence the result of the whole test procedure and how to measure their *quality*. Because walking strategies and stopping rules can be very complex, we have no analytical methods to determine their quality in general. Therefore, we have to resort to *empirical* methods. In order to do so, we need a *population* of labelled transition systems that could be used as benchmark. Instead of fixing some finite set of labelled transition systems, we define a mechanism to generate an infinite population of labelled transition systems, each element having a certain probability of being generated. We will see in Section 9.1 that the whole class of labelled transition systems can be synthesized by repeated application of the so-called *M-operator*. Based on this approach, we describe in Section 9.2 a procedure to test the quality of different test procedures. Finally, in Section 9.3 we report on the status of a software tool that can be used to study in an experimental way different test procedures for software systems that can be modelled as labelled transition systems.

9.1 Generating random models

Labelled transition systems can be synthesized out of smaller labelled transition systems by applying certain rules. We focus on the generation of labelled transition systems by a synthesis rule defined by the so-called *M-operator* (see Definition 9.1). Similar rules can be found in Berthelot (1978), Desel and Esparza (1995), Murata (1989), Suzuki and Murata (1983) and Valette (1979) for Petri nets. The difference with these papers is that we are not interested in preservation of properties like boundedness and liveness, but we wish to preserve *correctness*, i.e., conditions (1) to (6) in Definition 7.1. Such rules are proved to be *sound* and *complete* for labelled transition systems, which means that:

- (i) all generated labelled transition systems are correct (soundness),
- (ii) any LTS can be generated (completeness).

Thus, the set of all labelled transition systems can be synthesized by repeated application of such rules. We first introduce a synthesis rule for labelled transition systems called the *M-operator*.

Definition 9.1 (\mathcal{M} -operator). Let $\tilde{\mathcal{S}}$ be a set of state labels and $\tilde{\mathcal{T}}$ a set of transition labels such that $\tilde{\mathcal{T}} \cap \tilde{\mathcal{S}} = \emptyset$. If $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ is an LTS, then $\mathcal{S} \subseteq \tilde{\mathcal{S}}$ and $\mathcal{T} \subseteq \tilde{\mathcal{T}}$. We denote by \mathcal{L} the set of all labelled transition systems and by $\mathcal{P} = (\tilde{\mathcal{S}} \times \tilde{\mathcal{T}} \times \tilde{\mathcal{S}})^*$ the set of all linear paths. Let $p = (b, t_1, s_1, \dots, s_{n-1}, t_n, e)$ be a linear path such that $\{b, e\} \subseteq \mathcal{S}$, $\{s_1, \dots, s_{n-1}\} \cap \mathcal{S} = \emptyset$ and $\{t_1, \dots, t_n\} \cap \mathcal{T} = \emptyset$. We define the \mathcal{M} -operator as a function $\mathcal{M} : \mathcal{L} \times \mathcal{P} \rightarrow \mathcal{L}$ such that $\mathcal{M}(L, p) = (\mathcal{S}', \mathcal{T}', \mathcal{R}')$, where

- (1) $\mathcal{S}' = \mathcal{S} \cup \{s_1, \dots, s_{n-1}\}$,
- (2) $\mathcal{T}' = \mathcal{T} \cup \{t_1, \dots, t_n\}$,
- (3) $\mathcal{R}' = \mathcal{R} \cup \{(b, t_1, s_1), (s_1, t_2, s_2), \dots, (s_{n-1}, t_n, e)\}$.

Thus, the \mathcal{M} -operator *extends* a given LTS with a linear path of finite length. This leads us to define the following relationship between two labelled transition systems.

Definition 9.2. Let $L_1 = (\mathcal{S}_1, \mathcal{T}_1, \mathcal{R}_1)$ and $L_2 = (\mathcal{S}_2, \mathcal{T}_2, \mathcal{R}_2)$ be two labelled transition systems. We say that L_1 is a sub-LTS of L_2 , or equivalently L_2 is a super-LTS of L_1 , if and only if $\mathcal{S}_1 \subseteq \mathcal{S}_2$, $\mathcal{T}_1 \subseteq \mathcal{T}_2$ and $\mathcal{R}_1 \subseteq \mathcal{R}_2$. We denote this situation by $L_1 \subseteq L_2$.

Note that, in fact, the \mathcal{M} -operator creates a super-LTS of a given LTS. This can be seen as *gluing* a linear path to an LTS and can be defined as follows:

Definition 9.3. Let $p_1 = (s_1, t_1, \dots, t_{n-1}, s_n)$ and $\tilde{p} = (\tilde{s}_1, \tilde{t}_1, \dots, \tilde{t}_{m-1}, \tilde{s}_m)$ be two paths (not necessarily linear) in an LTS such that $s_n = \tilde{s}_1$. We define the glued path of s and \tilde{s} by $s \circ \tilde{s} = (s_1, t_1, \dots, s_n, \tilde{t}_1, \dots, \tilde{s}_m)$. We called \circ the gluing operator.

Note that the glued path $s \circ \tilde{s}$ is also a path in an LTS according to Definition 7.1. The definition of the \mathcal{M} -operator also implies that we add a path that does not have cycles. This is important for the proof of completeness as we will see below. For that reason, the following result is needed. Since the proof is quite straightforward, we skip it here and refer to [Corro Ramos et al. \(2006a\)](#) for details.

Lemma 9.1. Let p be a path in an LTS. If p has cycles, then there exists another non-empty path \tilde{p} , without cycles, such that \tilde{p} is a subpath of p , $\text{first}(\tilde{p}) = \text{first}(p)$ and $\text{last}(\tilde{p}) = \text{last}(p)$.

Finally, the next two theorems show that the \mathcal{M} -operator is sound and complete for labelled transitions systems.

Theorem 9.1 (Soundness). Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an LTS. Suppose that $p = (b, t_1, s_1, \dots, s_{n-1}, t_n, e)$ is a linear path such that $\{b, e\} \subseteq \mathcal{S}$, $\{t_1, \dots, t_n\} \cap \mathcal{T} = \emptyset$ and $\{s_1, \dots, s_{n-1}\} \cap \mathcal{S} = \emptyset$. Then, $\mathcal{M}(L, p)$ is an LTS. Thus, the \mathcal{M} -operator is sound for labelled transition systems.

Proof. Let us consider $\mathcal{M}(L, p)$. This is an extension of L and since the extension of L is a new path from b to e it holds that there is no new nor final state introduced and every node of L is still on a path from i to f . Since b and e are on paths from i to f , there is a path u from i to b and a path v from e to f . So every new node $x \in \{t_1, \dots, t_n\} \cup \{s_1, \dots, s_{n-1}\}$ is on a path from i to f , namely $u \circ p \circ v$. \square

Theorem 9.2 (Completeness). *Let $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{T}}$ be disjoint sets of states and transitions labels. Every LTS over $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{T}}$ can be constructed by repeated application of the \mathcal{M} -operator.*

Proof. Let $L = (\mathcal{S}, \mathcal{T}, \mathcal{R})$ be an arbitrary LTS over $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{T}}$, respectively, with an initial state i and a set of final states $\{f_1, \dots, f_k\}$.

1. Initial step: Consider $L_0 = (\{i, f_1, \dots, f_k\}, \emptyset, \emptyset)$. Note that L_0 is a sub-LTS of L according to Definition 9.2.
2. Iteration: Let $L_n = (\mathcal{S}_n, \mathcal{T}_n, \mathcal{R}_n)$ be an LTS such that $L_n \subseteq L$. Choose a transition $t \in \mathcal{T} \setminus \mathcal{T}_n$. Since L is an LTS, there exists a path p from i to a final state, say f_ℓ , with $1 \leq \ell \leq k$, via t in L . We can consider $p = u \circ v \circ w$, where $first(u) = i$, $last(u) = b$, $first(v) = b$, $last(v) = e$, $first(w) = e$, $last(w) = f_\ell$ and b and e are the only nodes in v that are also in $\mathcal{S}_n \cup \mathcal{T}_n$. Clearly v is not empty (at least it contains one transition). If v is not linear, then by Lemma 9.1 there is a (non-empty) linear path, denoted by \tilde{v} , that is also a subpath of v and it is such that $first(\tilde{v}) = b$ and $last(\tilde{v}) = e$. Note that t may not be a transition on \tilde{v} . Consider now $L_{n+1} = \mathcal{M}(L_n, \tilde{v})$. We have the following properties:
 - (a) $L_{n+1} \subseteq L$: we have added a path \tilde{v} of L to L_n so only nodes from $\mathcal{S} \cup \mathcal{T}$ and arcs from \mathcal{R} are added.
 - (b) L_{n+1} is an LTS: this is trivial by the soundness property of the \mathcal{M} -operator (see Theorem 9.1) since it transforms an LTS into an LTS.
 - (c) If $L_{n+1} = L_n$, then $L_n = L$: note that only $\mathcal{M}(L_n, \varepsilon) = L_n$, where ε denotes the empty sequence. This is the case where we cannot find a transition $t \in \mathcal{T} \setminus \mathcal{T}_n$. Therefore, $\mathcal{T} = \mathcal{T}_n$. By construction, for any $t \in \mathcal{T} \cap \mathcal{T}_n$, we have that $\bullet t \subseteq \mathcal{S}_n$ and $t \bullet \subseteq \mathcal{S}_n$. Therefore, it follows that $L_n = L$.

□

We have shown that the whole population of labelled transition systems can be synthesized by repeated application of the \mathcal{M} -operator. In fact, we can consider that the population of labelled transition systems depends on the following parameters:

1. Number of final states in an LTS. This can be fixed in advance or chosen according to a certain probability distribution.
2. Number of times we apply the \mathcal{M} -operator. This can be fixed in advance or chosen according to a certain stochastic process (Bernoulli, Poisson, etc.).
3. Length of the path added by the \mathcal{M} -operator in each step. This can be chosen according to a certain probability distribution.
4. Selection of the states b and e to which the \mathcal{M} -operator will be applied. This can be done at random over the set of states \mathcal{S} .

Therefore, we can generate random samples from the population of labelled transition systems providing us an infinite benchmark for studying our procedure in an *experimental* way. Further, we can compute all kind of characteristics of the population like the expected value of T (total number of transitions), the expected degree of nodes, the expected length of the longest path, etc. Although the generation of random models is out of the scope of this thesis, we think that this is an interesting area for future research.

9.2 Quality of the procedure

Our major interest is to establish the *quality* of a test procedure, i.e., to determine which are the *best* walking strategies and stopping rules. For that purpose we can use different kind of statistics collected during testing. For example, for walking strategies some metrics of interest are the number of runs to detect all error-marked transitions, the number of runs to observe all transitions, the length of the runs, the number and type of transitions observed, etc. For stopping rules we can compute the number of runs to stop testing, the number of tests with remaining error-marked transitions, the number of error-marked transitions left or the observed reliability. We have presented different walking strategies in Section 7.5 and stopping rules in Chapter 8 but we can define other rules and many different walking strategies. The same applies to the error distribution. We have presented in Section 7.3 a simple way to define and distribute error-marked transitions over an LTS but we can also consider a more complicated (and maybe more realistic) way of doing this. For example, we can define different types of error-marked transitions and consider correlations between them. With all the elements introduced so far, we can define an experimental step-by-step procedure to determine the quality of a certain test procedure as follows:

- (1) Choose a walking strategy and a stopping rule.
- (2) Generate an LTS of the population chosen. For example, the population can be the labelled transition systems introduced in Definition 7.1 and the generation method can be based on the \mathcal{M} -operator and the population parameters mentioned in Section 9.1.
- (3) Distribute a certain number of error-marked transitions over the LTS. We can consider several probability distributions for this, for example the error-marking process described in Section 7.3.
- (4) Test the system according to the walking strategy and stopping rule chosen in step (1) for a given distribution of error-marked transitions. We will refer to this as an *experimental unit*.
- (5) From the previous step we collect test data.
- (6) We repeat steps (4) and (5), say $n_1 > 0$ times.

- (7) We repeat steps (3) to (5), say $n_2 > 0$ times.
- (8) We repeat steps (2) to (5), say $n_3 > 0$ times.
- (9) End the procedure.

The whole procedure consisting of steps (1) to (9) is called an *experiment*. Therefore, the *parameters of the experiment* are the walking strategy, the stopping rule and the integers n_1 , n_2 and n_3 . In this case, we can define the *size* of the experiment as the number of its experimental units which is given by $n = n_1 n_2 n_3$. Note that the smallest experiment is precisely a single experimental unit, i.e., it corresponds to the case where $n_1 = n_2 = n_3 = 1$ and it consists of testing (according to the chosen walking strategy and stopping rule) one time only a single LTS with a fixed distribution of error-marked transitions. An example of an experiment can be observed in Table 9.1. The size of the experiment is n and for each experimental unit we display

		Before stopping			After stopping	
Exp. unit		Runs to stop	Seen transitions	Seen error-marked	Errors left	Survivals
Experiment	1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$
	2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$
	3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$
	⋮	⋮	⋮	⋮	⋮	⋮
	⋮	⋮	⋮	⋮	⋮	⋮
	n	$x_{n,1}$	$x_{n,2}$	$x_{n,3}$	$x_{n,4}$	$x_{n,5}$

Table 9.1: Example of experiment.

under the label *Before stopping* the number of runs to stop testing, the total number of observed transitions and the number of observed error-marked transitions. In the last two columns (*After stopping*) we observe the number of error-marked transitions left when we decide to stop testing and the number of consecutive *new* error-free transitions observed if we decide to perform one more run after testing is stopped. Note that these two metrics are related to the two stopping rules presented in Chapter 8 since they can be used to check the reliability of the procedure as we will explain now. Our stopping rules are statistically based since they allow us to stop testing with a certain statistical reliability. For that reason it is of special interest to study whether the procedure performs according to the required reliability, i.e., how many times the application of a stopping rule yields the correct answer. We consider that the result of an experimental unit with respect to a stopping rule has two possible outcomes: *success*, which occurs when the stopping rule is correct and *failure*, otherwise. For example, for the stopping rule presented in Section 8.1, if we stop testing and there are at most k remaining error-marked transitions in the LTS, then this is considered as a success. Note that the average number of successful experimental units is what we called *observed reliability* in Chapter 6 and Section 9.2 (and denoted by $\tilde{\delta}$ in Chapter 6). Therefore, if we fix a reliability level

$1 - \delta$ for our procedure, then, as shown in Theorem 8.1 and Theorem 8.2 for our stopping rules, the probability of a failure experimental unit is at most δ . Thus, the outcome of the i^{th} experimental unit can be defined as a Bernoulli random variable B_i with success parameter p , which is larger than or equal to $1 - \delta$, i.e.,

$$\mathbb{P}[B_i = b_i] = p^{b_i}(1 - p)^{1 - b_i}, \quad (9.1)$$

for $b_i = 0, 1$, where 0 represents a failure experimental unit, 1 represents a successful experimental unit and $p \geq 1 - \delta$. Note that, given a sample of size n (the number of experimental units that we want to consider), the ML estimator of p , denoted by \hat{p} , is in this case the percentage of successes that are observed in the sample. Therefore, we have that

$$\hat{p} = \frac{1}{n} \sum_{i=1}^n B_i. \quad (9.2)$$

Thus, \hat{p} is the observed reliability. If we assume that the result of one experimental unit does not have effect on the result of another one, then the random variables B_i are i.i.d. for all $i = 1, \dots, n$, and the random variable $B = \sum_{i=1}^n B_i$ is binomially distributed with parameters n and p . Therefore, by (9.2), we have that

$$\mathbb{E}[\hat{p}] = p \geq 1 - \delta, \quad (9.3)$$

and

$$\text{Var}[\hat{p}] = \frac{p(1 - p)}{n} \leq \frac{\delta(1 - \delta)}{n}, \quad (9.4)$$

where the last inequality holds assuming that $\delta < 1/2$. Thus, from (9.2) and (9.3) we can see that the expected number of failure experimental units is precisely np . By the Central Limit Theorem it follows that, as $n \rightarrow \infty$,

$$\frac{\hat{p} - p}{\sqrt{p(1 - p)/n}} \longrightarrow W \sim \mathcal{N}(0, 1), \quad (9.5)$$

where $\mathcal{N}(0, 1)$ denotes the standard normal distribution. Therefore, if we fix a confidence level $\alpha \in (0, 1)$, we can write a confidence interval for p as follows:

$$\left(\hat{p} - z_{\frac{\alpha}{2}} \sqrt{\hat{p}(1 - \hat{p})/n}, \hat{p} + z_{\frac{\alpha}{2}} \sqrt{\hat{p}(1 - \hat{p})/n} \right),$$

where z_γ denotes the γ -percentile of the standard normal distribution. Moreover, since we expect that \hat{p} has a value near to p , we can determine the size of the experiment (the number of experimental units) to ensure that \hat{p} is as close as p as desired with confidence at least α . Thus, if d denotes the radius of the interval centered in p , then n is given by

$$n = \frac{\hat{p}(1 - \hat{p})}{(d/z_{\frac{\alpha}{2}})^2}. \quad (9.6)$$

In this way, we can establish beforehand the size of the experiment in order to reach a prefixed reliability level. Next we introduce a software tool that allows us to implement the approach described in this section in an automated way.

9.3 Stresser: a tool for model-based testing certification

Stresser is a software tool used to study different test procedures for software systems in an experimental way. Stresser works with Petri nets which, as mentioned in Section 7.1, are a generalization of the labelled transition systems introduced in Definition 7.1. Stresser is based on PNML (Petri Net Markup Language) standards (see e.g. Billington et al. (2003)) and can be used together with *Yasper* (see van Hee et al. (2006) for details). *Yasper* is a tool to specify and execute models of processes that allows Stresser to view PNML files as Petri nets. Like the diagram technique introduced in Section 7.1, *Yasper* uses different symbols to distinguish between states and transitions: circles for states and rectangles for transitions. An oriented arrow represents the arc between an input state and an output state. The GUI of Stresser is shown in Figure 9.1. We can distinguish 5 different parts on the GUI. We explain the characteristics of each of them below.

9.3.1 Creating labelled transition systems

To begin with an experiment we need an LTS that models our software system. This can be done in the *Creating nets* frame (see Figure 9.2). We can use an already existing LTS, for example one created with *Yasper*. We can also generate an LTS by specifying the number of transitions of the LTS and the parameters of certain generation rules (like the \mathcal{M} -operator in Definition 9.1). By clicking the *Generate* button, Stresser creates an LTS using the previously specified input data. For example, as initial step, it considers only two states (that will be the initial state and a final state, respectively). Next, we apply the \mathcal{M} -operator to those states to obtain an initial LTS. Afterwards, two states and the length of the path to be added by the \mathcal{M} -operator to the existing LTS are selected. As shown in Section 9.1, the whole class of labelled transition systems can be generated by using this procedure. After generating the LTS, the *Show net* button allows us to see the graphical representation of the created LTS in the default viewer for PNML files (for example, *Yasper*). With the option *Regenerate for each test* selected, a different LTS is generated after the testing of a given LTS is finished. This is done according to the generation parameters specified in the beginning.

9.3.2 Error distribution

After an LTS is generated, we can distribute error-marked transitions over it. Note that for us an error is just a special label given to a transition (see Definition 7.3). This can be done in the *Error distribution* frame (see Figure 9.3). There are two ways of distributing error-marked transitions currently available in Stresser: either we specify in advance a fixed number of error-marked transitions or we specify an error-marking probability for each transition (denoted by θ in Section 7.3). Afterwards, the error-marked transitions are distributed by clicking the *Distribute* button. We can observe how many error-marked transitions have been distributed in the *number of errors distributed* text box.

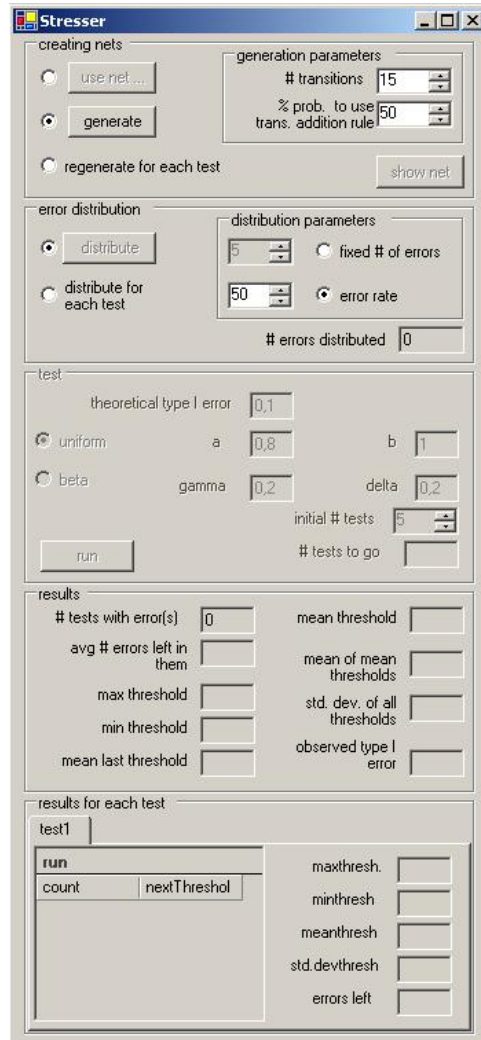
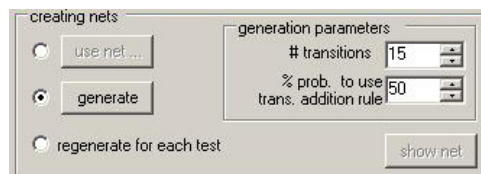
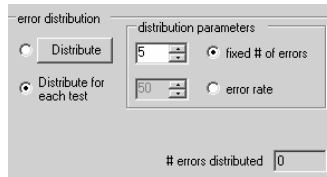
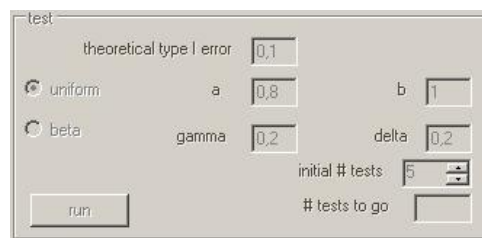


Figure 9.1: Stresser GUI.

Figure 9.2: *Creating nets* frame.

Figure 9.3: *Error distribution* frame.Figure 9.4: *Test* frame.

9.3.3 Parameters of testing: walking strategy and stopping rule

After distributing error-marked transitions over the LTS at hand, we must now decide *how* we wish to test it. This can be done by selecting a walking strategy and a stopping rule. The walking strategy can be based on a probabilistic choice, like in Section 7.5, or in a deterministic choice, for example deciding the next transition to be tested according to a log file. This choice (probabilistic or deterministic) cannot be changed until the testing of the current LTS stops (based on the selected stopping rule). Afterwards, and based on our previous choices, we can test the same LTS with a different distribution of error-marked transitions, test another LTS or end with testing. The parameters needed for their stopping rule can be observed in Figure 9.4.

9.3.4 Collecting results

After testing is completed some important statistics are collected. These statistics are used to measure the quality of the chosen test procedure. In fact, as mentioned in the previous section, we can collect any kind of information that we think we can use to compare different walking strategies and stopping rules. For example, the number of runs to stop testing, the average length of a run, the average number of error-free transitions after an error observation, the number of remaining error-marked transitions, etc. Of special interest is the metric called *observed reliability* in Section 9.2, since this guarantees that the reliability level required in our procedure is reached in reality. In Figure 9.5 we can observe a frame where some of the statistics of interest will be placed.

results			
# tests with error(s)	<input type="text" value="0"/>	mean threshold	<input type="text"/>
avg # errors left in them	<input type="text"/>	mean of mean thresholds	<input type="text"/>
max threshold	<input type="text"/>	std. dev. of all thresholds	<input type="text"/>
min threshold	<input type="text"/>	observed type I error	<input type="text"/>
mean last threshold	<input type="text"/>		

results for each test			
test1			
run			
count	nextThreshold	maxthresh.	<input type="text"/>
		minthresh	<input type="text"/>
		meanthresh	<input type="text"/>
		std.devthresh	<input type="text"/>
		errors left	<input type="text"/>

Figure 9.5: *Results* frame.

9.3.5 Further remarks

Stresser is in an early stage of development. Now the program generates only workflow transition systems (see Definition 7.8). Thus, all the experiments have to be done with this kind of systems. Moreover, it is not using the \mathcal{M} -operator (see Definition 9.1) for generating them but it uses two other synthesis rules (called Transition Addition and Transition Refinement) which are described in Corro Ramos et al. (2006b). The choice between different walking strategies and stopping rules is not possible yet. Stresser runs randomly through the generated WTS, following a path from the initial state to the final state. When an error-marked transition is found, it is repaired (without introducing new error-marked transitions) and a new run is started. Testing is stopped based on the approach presented in Di Bucchianico et al. (2008) that we discussed in Section 5.1.2. In the future, Stresser should be extended incorporating the class of labelled transition systems introduced in Definition 7.1 (and more complex classes of Petri nets) and adding the possibility of choosing between different walking strategies and stopping rules (for example, the ones presented in previous chapters of this thesis).

SUMMARY

The major contribution of this thesis is the development of *new statistical test procedures for certification of software systems* for both black-box and model-based testing. Our main goal is to certify, with high statistical confidence, that software systems do not have certain undesirable properties. In particular we focus on: *fault-free period after last failure observation* and *number of remaining faults in the system*. The procedures developed in the black-box context consider a large family of software reliability growth models: *semi-Markov* models with *independent times between failures*. In model-based testing we use labelled transition systems (a special class of Petri nets) as the model of software. Practical application of our approach is supported with *software tools*. The main results of this thesis can be found in chapters 5 and 6 for black-box testing, and in chapters 7 and 8 for model-based testing.

In Chapter 2 we emphasize the important role of *stochastic processes* in black-box testing. Based on basic properties of stochastic processes we propose a classification scheme for software reliability growth models. We also describe two popular families of models known as *General Order Statistics* (GOS) and *Non-homogeneous Poisson Process* (NHPP) models. We explain how these classes are related in a *Bayesian* way.

In Chapter 3 we present a step-by-step procedure to statistically analyze software failure data. For us statistical analysis of software reliability data should include data description, trend tests, initial model selection, estimation of model parameters, model validation and model interpretation. In particular, the problem of initial model selection has not been studied in details in the software reliability literature. We also focus on model estimation and illustrate some common problems related to Maximum Likelihood (ML) estimation. In general, to obtain the ML estimators of model parameters numerical optimization is required which is often a very difficult problem. Finally, we point out that further research is needed in this area, especially in problems regarding analysis of *interval-time* data and computation of *confidence intervals* for the parameters of the models.

In Chapter 4, we report on the status of a new software reliability tool to perform statistical analyses of software failure data based on the approach described in chapters 2 and 3. The new tool is a joint project of the Laboratory for Quality Software (LaQuSo) of the Eindhoven University of Technology (www.laquso.com), Refis (www.refis.nl) and the Probability and Statistics group of the Eindhoven University of Technology.

In Chapter 5 we present a sequential software release procedure where the certification criterion can be defined as *the next software failure is not observed in a certain time interval*. Our procedure is developed assuming that the failure detection process can be modeled as a *semi-Markov* software reliability growth model with *independent times between failures*. The main result of this chapter shows that under certain conditions the global risk taken in the whole procedure (defined as

the probability to stop testing too early) can be controlled.

In Chapter 6 we study, via simulation, the performance of our certification procedure for the models considered in Chapter 5.

In Chapter 7 we introduce a general framework for model-based testing. We use a model of *labelled transition systems* (a special class of Petri nets) where each transition in a labelled transition system represents a software component. We assume that the transitions can either have a correct or an erroneous behaviour. For us a fault is a symbolic labelling of a transition. Transitions labelled as erroneous are called *error-marked*. We assume that the number of error-marked transitions is unknown and a fault can only be discovered by executing the corresponding error-marked transition. In this context a *test* is defined as the execution of a *run* through the system that either ends without discovering an erroneous transition (successful run), or it ends in an error-marked transition (failure run). Our main assumption is to consider testing to be *non-anticipative*, i.e., it does not depend on future observed transitions but it may depend on the past (test history). Under this assumption, we prove in Section 7.4 that the error-marking of transitions at the beginning (caused by the programmers) gives the same distribution as error-marking on the fly (when a transition is tested) and that this holds for all possible testing strategies. A testing strategy for labelled transition systems based on reduction techniques is described in Section 7.5. The main idea is that after each successful run, we increase the probability of visiting unseen transitions. For that reason, we discard for the next run some already visited parts of the system. We show that after a finite number of updates all the transitions are visited, so that the updating procedure is exhaustive.

In Chapter 8 we describe two statistical certification procedures for the testing framework developed in Chapter 7. We consider the process where only the transitions observed for the first time are taken into account. We refer to this as the *embedded process*. We provide two statistical stopping rules, that are independent of the underlying way of walking through the system, which allows us to stop earlier with a certain statistical reliability. The first rule is based on the probability of having a certain number of remaining error-marked transitions when we decide to stop testing and the second one is based on the survival probability of the system. Like in Chapter 5, we also prove that the global risk can be controlled. Finally, we illustrate our whole approach with an example.

In Chapter 9 we discuss how different testing strategies and stopping rules may influence the result of the whole test procedure and how to measure their *quality*. Since testing strategies and stopping rules can be very complex, we have no analytical methods to determine their quality in general. Therefore, we have to resort to empirical methods. In order to do so, we need a population of labelled transition systems that could be used as benchmark. Instead of fixing some finite set of labelled transition systems, we define a mechanism to generate an infinite population of labelled transition systems, each element having a certain probability of being generated. Based on this approach, we describe a procedure to test the quality of different test procedures. Finally, we present a software tool that can be used to study in an experimental way different test procedures for software systems that can be modelled as labelled transition systems.

BIBLIOGRAPHY

- A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood. Evaluation of competing software reliability predictions. *IEEE Trans. Softw. Eng.*, 12(9):950–967, 1986. [26, 30, and 54]
- J. Achcar, D. Dey, and M. Niverthi. A Bayesian approach using nonhomogeneous Poisson process for software reliability models. In Basu et al., editor, *In Frontiers in Reliability*, 1997. [100]
- J.A. Achcar. A generalized Moranda software reliability model: a Bayesian approach. *Rev. Mat. Estatist.*, 15:115–130, 1997. [104]
- A. Andrews, J. Offutt, and R. Alexander. Testing web applications by modeling with FSMs. *Software Systems and Modeling*, 4(3):326–345, July 2005. [131 and 132]
- J. I. Ansell and M. J. Phillips. *Practical methods for reliability data analysis*, volume 14 of *Oxford Statistical Science Series*. The Clarendon Press Oxford University Press, New York, 1994. [20, 27, and 53]
- H. Ascher and H. Feingold. *Repairable Systems Reliability: Modeling, Inference, Misconceptions and Their Causes*. Marcel Dekker, New York, 1984. [46]
- J.L. Bain, M. Engelhardt, and F.T. Wright. Tests for an increasing trend in the intensity of a Poisson process. *J. Amer. Stat. Assoc.*, 80:419–422, 1985. [52]
- R. E. Barlow and R. Campo. Total Time on Tests Processes and Applications to Failure Data Analysis. In Barlow, Fussel, and Singpurwalla, editors, *Reliability and Fault Tree Analysis*, pages 451–481. SIAM, Philadelphia, 1975. [47]
- V. R. Basili and B. T. Perricone. Software errors and complexity: an empirical investigation. *Commun. ACM*, 27(1):42–52, 1984. [2 and 3]
- B. Beizer. *Software testing techniques*. Van Nostrand Reinhold, New York, 2 edition, 1990. [2]
- F. Belli, C.J. Budnik, and L. White. Event-based modelling, analysis and testing of user interactions: approach and case study. *Software Testing, Verification and Reliability*, 16(1):3–32, 2006. [132]
- G. Bernot, M. C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Softw. Eng. J.*, 6(6):387–405, 1991. [131]
- G. Berthelot. *Vérification de Réseaux de Petri*. PhD thesis, Université Pierre et Marie Curie, Paris, 1978. [167]

- J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The petri net markup language: Concepts, technology, and tools. In *Applications and Theory of Petri Nets 2003: 24th International Conference*, pages 1023–1024, Eindhoven, The Netherlands, June 2003. [173]
- M.A.A. Boon, E. Brandt, I. Corro Ramos, A. Di Bucchianico, and R. Henzen. A new statistical software reliability tool. In P. Groot, A. Serebrenik, and M. van Eekelen, editors, *Proceedings of VVSS2007 - verification and validation of software systems*, pages 125–139, Eindhoven, The Netherlands, March 2007. Technische Universiteit Eindhoven. [5, 54, and 65]
- H. Bowman and R. Gomez. *Concurrency Theory: Calculi an Automata for Modelling Untimed and Timed Concurrent Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. [131]
- E. Brandt, I. Corro Ramos, A. Di Bucchianico, and R. Henzen. Software reliability growth models : systematic descriptions and implementations. In *Proceedings 7th Annual ENBIS Conference*, pages 1–15, Dortmund, Germany, September 2007a. [5, 54, and 65]
- E. Brandt, R. Henzen, I. Corro Ramos, and A. Di Bucchianico. Two case studies in applying statistical models in software development. *Quality Engineering*, 19(4): 339–351, October 2007b. [5, 54, 63, 65, 78, 79, and 84]
- E. Brinksma. A theory for the derivation of tests. In P. H. J. van Eijk, C. A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS: Results of the ESPRIT/SEDOS Project*, pages 235–247. Elsevier Science Publishers North-Holland, 1989. [131]
- E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In F. Cassez et al., editor, *Modelling and Verification of Parallel Processes: 4th Summer School, MOVEP 2000*, volume 2067 of *Lecture Notes in Computer Science*, pages 187–195. Springer, Berlin, 2001. [131]
- T. Chen, A. von Mayrhauser, A. Hajjar, C. Anderson, and M. Sahinoglu. How much testing is enough? Applying stopping rules to behavioral model testing. *HASE*, 00:249, 1999. ISSN 1530-2059. [132]
- T. Chow. Testing software designs modeled by finite-state machines. *IEEE Trans. Softw. Eng.*, 4(3):178–187, 1978. [131]
- R. H. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Softw.*, 7(6):44–54, 1990. [92]
- A. Cohen and H. B. Sackrowitz. Evaluating tests for increasing intensity of a Poisson process. *Technometrics*, 35(4):446–448, 1993. [52]

- I. Corro Ramos, A. Di Bucchianico, L. Hakobyan, and K. van Hee. Synthesis and Reduction of State Machine Workflow Nets. Technical Report CS 06-18, Eindhoven University of Technology, 2006a. [168]
- I. Corro Ramos, A. Di Bucchianico, L. Hakobyan, and K. van Hee. Synthesis and reduction of State Machine Workflow Nets. Technical Report CS 06-18, Eindhoven University of Technology, 2006b. [176]
- I. Corro Ramos, A. Di Bucchianico, L. Hakobyan, and K. M. van Hee. Model driven testing based on test history. *Transactions on Petri Nets and Other Models of Concurrency I*, 1:134–151, 2008. [6 and 132]
- I. Corro Ramos, A. Di Bucchianico, and K.M. van Hee. Statistical approach to software reliability certification. In *Proceedings XXXI Congreso Nacional de Estadística e Investigación Operativa (SEIO 2009)*, pages 1–18 (CD), Murcia, Spain, February 10-13 2009. [5]
- D.R. Cox and H.D. Miller. *The Theory of Stochastic Processes*. Chapman Hall, 1984. [12]
- L. H. Crow. Reliability analysis for complex repairable systems. In F. Proshan and R. J. Serfling, editors, *Reliability and Biometry*, pages 379–410. SIAM, Philadelphia, 1974. [37 and 75]
- P.A. Currit, M. Dyer, and H.D. Mills. Certifying the reliability of software. *IEEE Trans. Softw. Eng.*, 12(1):3–11, 1986. [4, 5, 55, 56, 63, 89, 90, 92, and 93]
- Y. Dai, M. Xie, and K. Poh. Modeling and analysis of correlated software failures of multiple types. *IEEE Transactions on reliability*, 54(1):100–106, March 2005. [20]
- S. R. Dalal and C. L. Mallows. When should one stop testing software? *J. Amer. Statist. Assoc.*, 83(403):872–879, 1988. [89]
- H. A. David and H. N. Nagaraja. *Order Statistics*. Wiley series in probability and statistics. Wiley, New Jersey, USA, third edition, 2003. [28 and 29]
- A. P. Dawid. Present position and potential developments: Some personal views. Statistical theory. The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278–292, 1984. [62]
- R. De Nicola and M Hennessy. Testing equivalence for processes. In J. Díaz, editor, *Proceedings of the 10th Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 548–560, London, UK, 1983. Springer-Verlag. [131]
- H. M. DeGroot. *Probability and Statistics*. Series in Statistics. Addison-Wesley, Reading, Mass., 1986. [61]
- H. M. DeGroot. *Optimal Statistical Decisions*. Wiley, 2004. [140]

- A. Denise, M.-C. Gaudel, and S.-D. Gouraud. A generic method for statistical testing. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, pages 25–34, Washington, DC, USA, 2004. IEEE Computer Society. [132]
- J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995. ISBN 0-521-46519-2. [133, 134, and 167]
- J.V. Deshpande and S.G. Purohit. *Life-Time Data: Statistical Models and Methods*, volume 11 of *Quality, Reliability and Engineering Statistics*. Editor World Scientific, 2005. [55]
- A. Di Bucchianico, Groote J. F., van Hee K., and R. Kruidhof. Statistical certification of software systems. *Comm. Stat. C*, 37:1–14, 2008. [4, 5, 32, 89, 90, 92, 106, 132, and 176]
- W. Du and A. P. Mathur. Categorization of software errors that led to security breaches. In *21st National Information Systems Security Conference*, pages 392–407, 1998. [2]
- J. T. Duane. Learning curve approach to reliability monitoring. *IEEE Transactions on Aerospace*, 2:563–566, 1964. [26, 37, 54, 59, and 60]
- M. Dwass. Extremal processes. *The Annals of Mathematical Statistics*, 35(4):1718–1725, 1964. [41]
- I. K. El-Far and J. A. Whittaker. Model-based Software Testing. In J. J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley, 2001. [131]
- M. Finkelstein, H.G. Tucker, and J.A. Veeh. Confidence intervals for N in the exponential order statistics problem. *Comm. Statist. Theory Methods*, 28(6):1415–1433, 1999. [57, 58, and 76]
- A. Fries and A. Sen. A survey of discrete reliability-growth models. *IEEE Trans. Rel.*, 45:582–604, 1996. [32]
- O. Gaudoin, B. Yang, and M. Xie. Confidence intervals for the scale parameter of the power-law process. *Comm. Stat. Theor. Meth.*, 35:1525–1538, 2006. [60 and 75]
- N. Glick. Breaking records and breaking boards. *The American Mathematical Monthly*, 85(1):2–26, 1978. [41]
- A.L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Trans. Soft. Eng.*, 11(12):1411–1423, 1985. [5, 20, 24, 26, 45, 52, and 54]
- A.L. Goel and K. Okumoto. An analysis of recurrent software failures on a real-time control system. In *Proceedings of the ACM Annual Technical Conference*, pages 496–500, 1978. [26, 35, 54, and 55]

- A.L. Goel and K. Okumoto. Time-dependent error detection rate model for software reliability and other performance measures. *IEEE Trans. Rel.*, R-28:206–211, 1979. [26 and 54]
- S.S. Gokhale, P.N. Marinos, and K.S. Trivedi. Important milestones in software reliability modeling. In *In Proc. of 8th Intl. Conf. on Software Engineering and Knowledge Engineering (SEKE '96)*, pages 345–352, 1996. [25]
- G.R. Grimmett and D.R. Stirzaker. *Probability and Random Processes*. Oxford University Press, 1988. [12, 98, and 156]
- M. Grottke. *Modeling Software Failures during Systematic Testing - The Influence of Environmental Factors*. Shaker Verlag, Aachen, 2003. [4]
- S.A. Hossain and R.C. Dahiya. Estimating the parameters of a non-homogeneous Poisson-process model for software reliability. *IEEE Trans. Rel.*, 42(4):604–612, 1993. [59 and 75]
- W.E. Howden. Methodology for the generation of program test data. *IEEE Trans. Softw. Eng.*, 24:208–215, 1975. [131]
- A. Høyland and M. Rausand. *System Reliability Theory. Models and Statistical Methods*. Wiley Series in Probability and Mathematical Statistics. Wiley, New York, 1994. [48 and 49]
- J.C. Huang. An approach to program testing. *ACM Comp. Surveys*, 7(3):113–128, 1975. [131]
- Z. Jelinski and P. Moranda. Software reliability research. In W. Freiberger, editor, *Statistical Computer Performance Evaluation*, pages 465–497. Academic Press, 1972. [24, 26, 30, 54, and 55]
- H. Joe. Statistical inference for General-Order-Statistics and Nonhomogeneous-Poisson-Process software reliability models. *IEEE Trans. Software Eng.*, 15(11):1485–1490, 1989. [46, 47, 49, 55, 57, 58, 59, 60, 66, 68, 74, and 76]
- H. Joe and N. Reid. Estimating the number of faults in a system. *J. Amer. Stat. Assoc.*, 80(389):222–226, 1985. [55 and 58]
- C. P. Jorgensen. *Software Testing: A Craftsman's Approach*. CRC Press, Inc., Boca Raton, FL, USA, 2002. [1, 2, and 3]
- K. Kanoun, M.R. de Bastos Martini, and J.M. de Souza. A method for software reliability analysis and prediction application to the tropico-r switching system. *Software Engineering, IEEE Transactions on*, 17(4):334–344, 1991. [50, 51, and 66]
- S. Karlin and H.M. Taylor. *A first course in stochastic processes*. Academic Press, 2 edition, 1975. [12, 14, and 16]

- V.S. Kharchenko, O.M. Tarasyuk, V.V. Sklyar, and V.Yu. Dubnitsky. The method of software reliability growth models choice using assumptions matrix. In *COMP-SAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, pages 541–546, Washington, DC, USA, 2002. IEEE Computer Society. [5, 24, 27, 45, and 52]
- B. Klefsjo and U. Kumar. Goodness-of-fit tests for the power-law process based on the TTT-plot. *Reliability, IEEE Transactions on*, 41(4):593–598, Dec 1992. [47, 49, and 61]
- G.J. Knafl. Solving Maximum Likelihood equations for two-parameter software reliability models for using grouped data. In *Proceedings of the International Symposium on Software Reliability Engineering.*, pages 205–213. IEEE, 1992. [59, 60, and 75]
- G.J. Knafl and J. Morgan. Solving ML equations for 2-parameter Poisson-process models for ungrouped software-failure data. *IEEE Trans. Rel.*, 45(1):42–53, 1996. [59, 75, 80, and 85]
- W. Koepf. *Hypergeometric summation: an algorithmic approach to summation and special function identities*. Advanced lectures in mathematics. Braunschweig: Vieweg, 1998. [108]
- V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman Hall, 1995. [15 and 16]
- H. Kunitz and H. Pamme. Graphical tools for life time data analysis. *Statistical Papers*, 32(1):85–113, December 1991. [49]
- L. Kuo and T.Y. Yang. Bayesian computation for nonhomogeneous Poisson processes in software reliability. *J. Amer. Statist. Assoc.*, 91(434):763–773, 1996. [39, 40, 56, and 104]
- J. T. Kvaløy and B.H. Linqvist. TTT-based tests for trend in repairable systems data. *Rel. Eng. System Safety*, 60:13–28, 1998. [52]
- N. Langberg and N.D. Singpurwalla. A unification of some software reliability models. *SIAM J. Sci. Statist. Comput.*, 6(3):781–790, 1985. [35, 39, and 42]
- N. A. Langberg, R. V. Leon, and F. Proschan. Characterization of nonparametric classes of life distributions. *The Annals of Probability*, 8(6):1163–1170, 1980. [48]
- H.J. Larson and B.O. Shubert. *Probabilistic models in engineering sciences. Random noise, signals, and dynamic systems*, volume 2. Wiley, 1979. [17]
- S. L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996. [94]

- C.H. Lee, K.H. Nam, and D. H. Park. Optimal software release policy based on Markovian perfect debugging model. *Comm. Statist. Theory Methods*, 30(11): 2329–2342, 2001. [132]
- D. Lee and M Yannakakis. Principles and methods of testing finite state machines-A survey. *Proceedings of the IEEE*, 84:1090–1126, 1996. [131]
- P. M. Lee. *Bayesian Statistics: An Introduction*. Oxford University Press, New York, 1989. [93]
- R.C. Linger. Cleanroom software engineering for zero-defect software. In *ICSE '93: Proceedings of the 15th international conference on Software Engineering*, pages 2–13, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press. ISBN 0-89791-588-7. [92]
- B. Littlewood. Theories of software reliability: How good are they and how can they be improved? *Software Engineering, IEEE Transactions on*, 6(5):489–500, September 1980. [55]
- B. Littlewood and A. J. Mayne. Predicting software reliability [and discussion]. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 327(1596):513–527, 1989. [63]
- B. Littlewood and J. L. Verrall. A Bayesian reliability growth model for computer software. *J. Roy. Statist. Soc. Ser. C Appl. Statist.*, 22:332–346, 1973. [42]
- B. Littlewood and J. L. Verrall. Likelihood function of a debugging model for computer software reliability. *IEEE Trans. Rel.*, 30:145–148, 1981. [58]
- M.R. Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill and IEEE Computer Society, New York, 1996. [9, 10, 30, 35, 37, 45, 52, and 60]
- K. McDaid and S. P. Wilson. Deciding how long to test software. *The Statistician, Journal of the Royal Statistical Society - Series D*, 50(2):117–134, 2001. [89]
- Military Handbook. *Military Handbook, Reliability Growth Management, MIL-HDBK-189*. US Department of Defense. [51, 52, and 79]
- D.R. Miller. Exponential order statistic model of software reliability growth. *IEEE Trans. Softw. Eng.*, 12(1):12–24, 1986. [30]
- H.D. Mills, M. Dyer, and R.C. Linger. Cleanroom software engineering. *Software, IEEE*, 4(5):19–25, Sept. 1987. [92]
- R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980. [131]
- G. Moek. Comparison of some software reliability models for simulated and real failure data. *Int. J. Modelling Sim.*, 4:29–41, 1984. [57]

- N. Morali and R. Soyer. Optimal stopping rules for software testing. *Naval Research Logistics*, 50:88–104, 2003a. [132]
- N. Morali and R. Soyer. Optimal stopping in software testing. *Naval Res. Logist.*, 50(1):88–104, 2003b. [89]
- P.B. Moranda. Event altered rate models for general reliability analysis. *IEEE Transactions on Reliability*, R-28(5), 1979. [26, 54, and 90]
- T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, 1989. [167]
- J. D. Musa and K. Okumoto. Software reliability models: concepts, classification, comparisons, and practice. In J. W. Skwirzynski, editor, *Proc. Electronic Systems Effectiveness and Life Cycle Costing Conference, Norwich, U. K., July 19-31, 1982*, volume F3 of *NATO ASI Series*, pages 395–424, Heidelberg, July 1983. Springer-Verlag. [24, 28, and 35]
- J. D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. In *ICSE '84: Proceedings of the 7th international conference on Software engineering*, pages 230–238, Piscataway, NJ, USA, 1984. IEEE Press. ISBN 0-8186-0528-6. [26, 40, 52, and 54]
- J.D. Musa. Validity of execution-time theory of software reliability. *IEEE Trans. Reliability*, R-28(3):181–191, August 1979. [26 and 54]
- J.D. Musa. *Software Reliability Engineering: More Reliable Software, Faster and Cheaper*. Author House, Bloomington, USA, 2nd edition, 2006. [45]
- J.D. Musa, A. Iannino, and K. Okumoto. *Software Reliability. Measurement, Prediction, Application*. McGraw Hill, New York, 1987. [35 and 45]
- G. J. Myers. *The art of software testing*. Wiley-Interscience, London, 1 edition, 1979. [2]
- M. Ohba. Software reliability analysis models. *IBM J. Res. Develop.*, 28(4):428–443, 1984. [4, 9, 20, 30, and 52]
- H. Okamura, A. Murayama, and T. Dohi. EM algorithm for discrete software reliability models: a unified parameter estimation method. *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*, pages 219–228, March 2004. [32 and 33]
- J.A. Osborne and T.A. Severini. Inference for exponential order statistic models based on an integrated likelihood function. *J. Amer. Statist. Assoc.*, 95(452): 1220–1228, 2000. [57]
- S. Özekici and A. Catkan. A dynamic software release model. *Computational Economics*, 6:77–94, 1993. [89]

- S. Özekici and R. Soyer. Bayesian testing strategies for software with an operational profile. *Naval Res. Logist.*, 48(8):747–763, 2001. [89]
- R. Patton. *Software Testing*. Sams, 2 edition, 2005. [2]
- H. Pham. *System Software Reliability*. Springer Series in Reliability Engineering. Springer, London, 2006. [45 and 52]
- H. R. Pitt. *Measure and Integration for use*. Clarendon Press, Oxford, 1985. [10 and 96]
- G. D. Plotkin. An operational semantics for CSP. In D. Bjørner, editor, *Formal Description of Programming Concepts II*, pages 199–223, North-Holland, Amsterdam, 1983. [131]
- J. H. Poore, H. D. Mills, and D. Mutchler. Planning and certifying software system reliability. *IEEE Softw.*, 10(1):88–99, 1993. [92]
- R. Pyke. Markov renewal processes: Definitions and preliminary properties. *The Annals of Mathematical Statistics*, 32(4):1231–1242, 1961. [15]
- R. Pyke. Spacings. *Journal of the Royal Statistical Society. Series B (Methodological)*, 27(3):395–449, 1965. [28 and 43]
- A. E. Raftery. Inference and prediction for a general order statistic model with unknown population size. *J. Amer. Statist. Assoc.*, 82(400):1163–1168, 1987. [30]
- C.V. Ramamoorthy and F.B. Bastani. Software reliability: Status and perspectives. *Software Engineering, IEEE Transactions on*, SE-8(4):354–371, July 1982. [23]
- M.M. Rao. *Stochastic Processes: General Theory*. Kluwer Academic Publishers, 1995. [12]
- W. Reisig. *Petri Nets: An Introduction*. Springer-Verlag New York, Inc., 1985. [134]
- S.I. Resnick. *Extreme Values, Regular Variation, and Point Processes*. Springer, 2007. [41]
- S.E. Rigdon and A.P. Basu. *Statistical Methods for the Reliability of Repairable Systems*. Wiley, 2000. [11, 16, 33, 34, 49, 50, 51, 61, and 62]
- S.M. Ross. *Stochastic Processes*. Wiley, 2 edition, 1996. [12]
- S.M. Ross. *Introductory statistics*. Elsevier, 2 edition, 2005. [163 and 164]
- S.M. Ross. *Introduction to probability models*. Academic Press, 9 edition, 2007. [10 and 139]
- M.J. Schervish. *Theory of Statistics*. Springer, 1995. [138]

- G.J. Schick and R. W. Wolverton. An analysis of competing software reliability models. *IEEE Transactions of Software Engineering*, 4(2):104–120, 1978. [26, 42, 43, and 54]
- S. Schneider. *Concurrent and Real Time Systems: The CSP Approach (Worldwide Series in Computer Science)*. John Wiley & Sons, September 1999. [131]
- R.W. Selby, V.R. Basili, and F.T. Baker. Cleanroom software development: An empirical evaluation. *IEEE Transactions on Software Engineering*, 13(9):1027–1037, 1987. [92]
- R.J. Serfling. *Approximation Theorems of Mathematical Statistics*. Wiley, 1980. [55]
- J. G. Shanthikumar. A general software reliability model for performance prediction. *Microelectron. Reliab.*, 21:671–682, 1981. [26, 43, and 54]
- D. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. CRC Press, 3 edition, 2004. [62 and 164]
- N.D. Singpurwalla. Determining an optimal time interval for testing and debugging software. *IEEE Trans. Soft. Eng.*, 17(4):313–319, 1991. [89 and 100]
- N.D. Singpurwalla and S.P. Wilson. Software reliability modeling. *International Statistical Review*, (62):289–317, 1994. [23, 25, 52, and 89]
- M. Sullivan and R. Chillarege. Software defects and their impact on system availability - a study of field failures in operating systems. *21st Int. Symp. on Fault-Tolerant Computing (FTCS-21)*, pages 2–9, 1991. [2]
- I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of Petri nets. *Journal of Computer and System Science*, 27(1):51–76, 1983. [167]
- P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet. Software statistical testing. In B. Randell, J. C. Laprie, H. Kopetz, and B. Littlewood, editors, *Predictably Dependable Computing Systems*, pages 253–272. Springer, 1995. [132]
- W. A. Thompson, Jr. On the foundations of reliability. *Technometrics*, 23(1):1–13, 1981. [4, 10, 14, and 90]
- W. A. Thompson, Jr. *Point Process Models with Applications to Safety and Reliability*. Chapman and Hall, New York, 1988. [16, 17, 20, 22, 27, 33, 34, 35, 53, and 58]
- J. Tretmans. *A formal approach to conformance testing*. PhD thesis, University of Twente, Enschede, The Netherlands, 1992. [131]
- K. Tsipenyuk, B. Chess, and G. McGraw. Seven pernicious kingdoms: A taxonomy of software security errors. *IEEE Security and Privacy*, 3(6):81–84, 2005. [2]
- R. Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, 18:35–46, 1979. [167]

- W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, Cambridge, Massachusetts, 2002. [133]
- A.W. van der Vaart. *Asymptotic Statistics*. Cambridge University Press, 2000. [55]
- J. R. van Dorp, T. A. Mazzuchi, and R. Soyer. Sequential inference and decision making for single mission systems development. *Journal of Statistical Planning and Inference*, 62(2):207–218, 1997. [89]
- K. van Hee, O. Oanea, R. Post, L. Somers, and J. M. van der Werf. Yasper: a tool for workflow modeling and analysis. *acsd*, pages 279–282, 2006. [134 and 173]
- M.C.J. van Pul. Asymptotic properties of a class of statistical models in software reliability. *Scand. J. Statist.*, 19(3):235–253, 1992a. [4]
- M.C.J. van Pul. Simulations on the Jelinski-Moranda model of software reliability; application of some parametric bootstrap methods. *Stat. Computing*, 2:121–136, 1992b. [58 and 66]
- M.C.J. van Pul. *Statistical Analysis of Software Reliability Models*, volume 95 of *CWI Tract*. Centrum voor Wiskunde en Informatica, Amsterdam, 1993. [55]
- P. Wang and D.W. Coit. Repairable systems, reliability, trend tests and evaluation. In *RAMS 2005 Proceedings*, pages 416–421, 2005. [52]
- E.A. Weller and L.M. Ryan. Testing for trend with count data. *Biometrics*, 54:762–773, 1998. [52]
- M. Xie. A markov process model for software reliability analysis. *Applied Stochastic Models and Data Analysis*, 6(4):207–213, 1990. [26 and 54]
- M. Xie and G.Y. Hong. Software reliability modeling, estimation and analysis. In *Advances in Reliability*, volume 20 of *Handbook of Statist.*, pages 707–731. North-Holland, Amsterdam, 2001. [45, 55, and 58]
- M. Xie and M. Zhao. On some reliability growth models with graphical interpretations. *Microelectronics and Reliability*, 33(2):149–167, 1993. [26 and 54]
- M. Xie, Y.S. Dai, and K.L. Poh. *Computing Systems Reliability. Models and Analysis*. Kluwer, New York, 2004. [10, 30, 35, 43, 52, 57, and 59]
- S. Yamada and S. Osaki. Nonhomogeneous error detection rate models for software reliability growth. In *Stochastic models in reliability theory (Nagoya, 1984)*, volume 235 of *Lecture Notes in Econom. and Math. Systems*, pages 120–143. Springer, Berlin, 1984. [30 and 36]
- S. Yamada, M. Ohba, and S. Osaki. S-shaped software reliability growth models and their applications. *IEEE Transactions on Reliability*, R-33(4):289–292, 1984. [26 and 54]

-
- S. Yamada, S. Osaki, and H. Narihisa. Discrete models for software reliability evaluation. In *Reliability and quality control (Columbia, Mo., 1984)*, pages 401–412. North-Holland, Amsterdam, 1986. [32]
- L. Yin and K.S. Trivedi. Confidence interval estimation of NHPP-based software reliability models. In *Proc. 10th Int. Symp. Software Reliability Engineering (ISSRE 1999)*, pages 6–11, 1999. [5 and 55]
- J. Zhao and J. Wang. A new goodness-of-fit test based on the Laplace statistic for a large class of NHPP models. *Comm. Statist. Simulation Comput.*, 34(3):725–736, 2005. [62, 66, 76, 82, and 85]
- H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997. [132]

INDEX

- \mathcal{M} -operator, 167
- arc, 133
- certification, 4
- Cleanroom Software Engineering, 92
- completeness, 167
- conditional independence, 94
- counting process, 13
 - occurrence time, 13
- cumulative distribution function, 10
- density function, 11
- embedded process, 155
- error-marking process, 136
- event, 10
- exact data, *see* point-time observations
- exchangeability, 138
- experiment, 171
 - parameters, 171
 - size, 171
- experimental unit, 170
 - failure, 171
 - success, 171
- failure time, 18
- fitted model, 60
- general order statistics, 27
- gluing operator, 168
- goodness-of-fit test, 60
- GOS, *see* general order statistics
- grouped data, *see* interval-time observations
- hazard rate, 11
- immediate repair, 20
- imperfect repair, 20
- increments
 - independent, 16
 - stationary, 17
- intensity function, 13
- interval data, *see* interval-time observations
- interval-time observations, 46
- Kolmogorov existence theorem, 12
- labelled transition system, 132
 - acyclic, 133
 - diagram technique, 134
 - one-path, 133
- likelihood function, 93
- LTS, *see* labelled transition system
- Markov
 - process, 16
 - property, 15
 - renewal moments, 14
 - renewal sequence, 14
- mean-value function, 13
- model selection wizard, 71
- observed reliability, 114
- path, 133
 - cycle, 133
 - linear, 133
- perfect repair, 20
- point-time observations, 46
- Poisson process
 - homogeneous (HPP), 33
 - non-homogeneous (NHPP), 33
 - finite, 35
 - infinite, 35
- posterior distribution, 93
- postset, 133
- prequential likelihood, 62
- preset, 133
- prior distribution, 41, 93
 - conjugate, 93
- probability mass function, 10
- probability space, 10
- R, 65
- random variable, 10

- continuous, 11
- discrete, 10
- record value statistics, 40
- reduced system, 147
- release time, 101
- reliability function, 11
 - conditional, 94
- reliability growth model
 - classification, 23
 - Duane (power-law), 37
 - geometric Moranda, 90
 - Goel-Okumoto, 35
 - Jelinski-Moranda, 30
 - Run, 92
 - Schick-Wolverton, 42
 - Yamada S-shaped, 36
- reliability level, 97
- residual lifetime probability, 11
 - conditional, 94
- risk, 97
 - global, 97
- run
 - failure, 141
 - successful, 141
- running averages plot, 47
- scaled TTT statistic, 48
- selection matrix, 53
- Semi-Markov process, 14
- software failure, 2
- software failure data, 46
 - failure truncated, 46
 - time truncated, 46
- software fault, 2
 - classification, 2
 - severity, 20
- software reliability, 9
 - tool, 65
- software testing
 - black-box, 3
 - clear-box, 3
 - functional, 3
 - model-based, 3
 - structural, 3
 - white-box, 3
- sojourn times, 14
- soundness, 167
- state, 133
 - final, 133
 - initial, 133
- stateful function, 3
- stochastic order, 21
- stochastic process
 - continuous-time, 12
 - discrete-time, 12
- stopping time, 156
- Stresser, 173
- subpath, 133
- survival function, *see* reliability function
- test, 1
 - case, 1
 - procedure, 1
- testing process, 142
- time between failures, 18
- total time on test plot, 47
- transition, 133
 - error-free, 135
 - error-marked, 135
- trend test, 46
 - Laplace, 49
 - Military Handbook (MIL-HDBK-189), 49
- TTT plot, *see* total time on test plot
- u-plot, 60
- ungrouped data, *see* point-time observations
- walking function, 141
 - update, 142
- workflow transition system, 149
- WTS, *see* workflow transition system
- Yasper, 173

ACKNOWLEDGEMENTS

I would like to express my most sincere gratitude to my supervisors Kees van Hee and Alessandro Di Bucchianico for their advice, support and patience during all these years, because without their help I could not have finished this thesis.

I would like to thank Remco van der Hofstad for his useful comments on my thesis which led to a significant improvement of the final version. I would also like to give thanks Ed Brinksma, Martin Newby and Harry van Zanten for revising my thesis and being part of the committee for my defense.

Many thanks to my colleagues of the Probability and Statistics group, the Architecture of Information Systems group and EURANDOM. In particular, thanks to my project mates, all my office mates during these four years and to all the players of the EURANDOM indoor football team for the good times we had. Thanks also to my family and friends for their constant support in the good and in the bad moments.

I would like to have a special thought for my grandparents and my friend Jacob who could only see how I started this journey but now how it finished. Wherever you are: I miss you.

I am deeply grateful to my parents and my sister because they showed me the path I had to follow to become the person that I am now. Thanks for showing me the true value of things in life.

Finally, thanks to Vanessa. Thanks for loving me, for being with me all these years, for your patience all this time and for the passion you put on all the things in life: you fill me with strength and confidence every time I need it.

To everyone I may have forgotten and also to you who are now reading this: Thank you!

Isaac

ABOUT THE AUTHOR

Isaac was born in Sevilla, Spain, on March 24, 1977. He obtained his Master's degree in Mathematics (option Statistics and Operations Research) from the University of Sevilla in June 2001. Between June 2001 and July 2005 he had several jobs in different working areas and countries. He worked as a high-school Mathematics teacher in Sevilla (Spain), as a software programmer in Madrid (Spain) and Vienna (Austria), and as a researcher in Rome (Italy). In July 2005 he started his Ph.D. at the Department of Mathematics and Computer Science of the Eindhoven University of Technology working on the STRESS (Statistical Testing and Reliability Estimation of Software Systems) project under the guidance of Prof. Kees van Hee and Dr. Alessandro Di Bucchianico. He defends his thesis on December 15, 2009. Since August 2009 Isaac works as a scientific researcher at the Institute for Medical Technology Assessment (iMTA) in Rotterdam, The Netherlands.