CWI Quarterly Volume 7 (3) 1994, pp. 203 – 244

A General Introduction to Software Reliability

Mark C. van Pul
Measurement and Computational Applications
Koninklijke/Shell-Laboratorium Amsterdam
P.O.Box 38000, 1030 BN Amsterdam, The Netherlands

The past two decades have witnessed an enormous growth of literature on software reliability theory. The preoccupation with model-building has resulted in a large number of theoretical models which seem to lack immediate appeal from a statistical point of view. The aim of this paper, which is entirely based on the author's PhD-thesis (VAN PUL (1993)), is to attempt to reduce the gap between statistical theory and applications in the area of software reliability.

In Section 1 a general introduction to software reliability is given. We discuss some basic concepts, models and assumptions and describe some different approaches to the subject. Section 2 provides most of the mathematical framework. The relevant notions of counting process theory and martingales are presented. Section 3 gives a brief sketch of some of the technical results, recently obtained and more thoroughly described in VAN PUL (1993). Asymptotic properties of maximum likelihood estimators are analysed for a broad class of models, the validity of the parametric bootstrap is derived in a software reliability context and a new, more sophisticated software reliability model incorporating imperfect repair is introduced. In Section 4 we consider an application. The results of a software reliability case study at Philips Medical Systems (PMS) are discussed.

1. INTRODUCTION

To-day it is hard to think of any area in modern society in which computer systems do not play a dominant role. In space- and air-navigation, defence, telecommunication and health-care, to name a few, computers have taken over the most life-critical tasks. Unlike most human beings, computers seem to do their job perfectly, at all times and under all conditions. But do they really? Well, most of the time, they do. Sometimes, however, for some reason

a zillion dollar satellite goes off course, a patriot rocket misses its target or a large telephone exchange gives up. Possible sources for such dissatisfactory behaviour are physical deterioration or design faults in hardware components. In the fifties and sixties a general reliability theory was built for hardware. Another source for malfunctioning of computer systems is the presence of bugs in the software that controls the system. A beginning with the modelling of the reliability of software was only made in the early seventies.

Obviously, also in the case of less delicate computer applications, all customers want a high degree of reliability to be guaranteed. Of course, every software-house claims to design and produce software in such a structured and sophisticated way that the result is a perfect computer program. As in general the logical complexity of software is much larger than that of hardware, proving the correctness of a piece of software is in most cases an impossible task. Software developers have to admit that in practice a completed program is never perfect, but, more likely, still full of bugs. Therefore, the software is tested intensively for quite a span of time before it is finally released. Here a difficult trade-off occurs between costs and schedule on the one side and quality on the other. The test time, which can mount up to more than a third of the total development time, seems not productive and therefore extremely expensive. Besides, there exists the risk, that a competitor will release the same product a bit earlier. On the other hand, the sales of an unreliable product will be disappointing and can do more bad than good to the image of the software-house. It seems to make sense to study the evolution of the reliability of computer software during the test and development phase: does the software already satisfy certain criteria or how long should testing be continued?

In this paper mathematical and statistical aspects of software reliability theory are presented. This first section provides a general introduction to software reliability theory. Two excellent handbooks on this topic are MUSA *et al.* (1987), and ROOK (1990). In the next Section 1.1 we discuss how software reliability relates to hardware reliability. In Section 1.2 some important features related to software reliability are defined. In Section 1.3 we describe several approaches – both static and dynamic – to estimation or prediction of the reliability of a piece of software. The dynamic approach of counting process models will be used throughout the rest of this paper. Classical assumptions and models in this approach are discussed in Section 1.4.

1.1. Software versus hardware

The field of hardware reliability has been established for some time. Some useful references are: SHOOMAN (1968), MANN *et al.* (1974), BARLOW & PROSCHAN (1975), LAWLESS (1982), and ASCHER & FEINGOLD (1984). One might ask how software reliability relates to it. In reality, the distinction between hardware and software reliability is somewhat artificial. Both may be defined in the same way. Therefore, you may combine hardware and software reliability to get system reliability. Both depend on the environment.

The source of failures in software is design faults, while the main source in

hardware has generally been physical deterioration. However, the models and methods developed for software reliability could really be applied to any design activity, including hardware design. Once a software defect is properly fixed, it is in general fixed for all time. Failures usually occur only when a program is exposed to an environment that it was not developed or tested for. Software reliability tends to change continuously during test periods, due to the addition of problems in new code or due to the removal of problems by repair actions. Hardware reliability has, apart perhaps from an initial burn-in or end of useful life period, a much greater tendency towards a constant value.

Also in hardware the presence of design faults is possible, but the design reliability concept has not been applied to hardware to any extent. The probability of failure due to wear and other physical causes has usually been much greater than that due to an unrecognised design problem. It was possible to keep hardware design failures low because hardware was generally less complex logically than software. Hardware design failures had to be kept low because retrofitting of manufactured items in the field was very expensive. The emphasis in hardware reliability is starting to change now, however. Awareness of the work that is going on in software reliability, plus a growing realisation of the importance of design faults may be having an effect.

Despite the forgoing differences, we can develop software reliability theory in a way that is compatible with hardware reliability theory. Thus, system reliability figures may be computed using standard hardware combinatorial techniques (SHOOMAN (1968), LLOYD & LIPOW (1977)). In this paper attention is restricted to problems related to the modelling of the reliability of software only.

1.2. Basic concepts

First of all, we have to make clear what we mean, when using vague terms like software faults and software reliability. In this section we give some intuitive descriptions; more formal definitions and mathematical expressions will be given later on.

We speak of a *software failure*, if the program-output deviates from what it should be according to the customer. This means that also errors in the specification can lead to software failures. A failure is a dynamic thing; the program has to be executed to detect software failures.

A *software fault* (or *bug*) is an error in the program source-text, which when the program is executed under certain conditions can cause a software failure. A software fault is hence generated at the moment a programmer or system analyst makes a mistake.

Often one defines the *reliability* of a piece of software as the probability of failure-free execution of the software for a specified time in a specified environment. An operating system, for instance, with a reliability of 95% for 8 hours for an average user, should work 95 out of 100 periods of 8 hours without problems.

A characteristic that is strongly correlated with the reliability is the expected

failure intensity, sometimes called the *rate of occurrence of failures* (ROCOF), which is defined as the expected number of occurring failures per unit of time. Strictly speaking, the expected failure intensity is the first derivative of the mean value function, which represents the expected cumulative number of failures detected up to each point in time.

Between the terms software fault and software failure there exists a cause and effect relation. The terms software reliability and failure intensity both give a measure for the quality of the software. Other interesting measures, that are directly related to these two, are the mean time between failures (MTBF) and the time to release (TTR).

1.3. Different approaches

Since the early seventies, many researchers have paid attention to the problem of estimation and prediction of software reliability. They used various starting-points, assumptions, and techniques; all aiming at the same goal. In this section we briefly discuss four different approaches to software reliability.

(i) *Fault seeding.* One can estimate the number of inherent faults in software programs by an empirical method, variously called fault seeding, error seeding or "bebugging" (MILLS (1972), BASIN (1973), GILB (1979) and RUDNER (1977)). The test-leader introduces a certain number of artificial faults into the program in some suitable random fashion, unknown to the people who will test the software. It is assumed that these seeded faults are equivalent to the inherent faults in terms of difficulty of detection. Inherent and seeded faults discovered are counted separately. The number of inherent faults can be predicted by using the observed proportion of seeded faults found to total seeded faults. The reasoning is based on the concept that with equal difficulty of discovery, the same proportions of both types of faults will have been discovered at any point in time. Unfortunately, it has proved difficult to implement seeding in practice. It is not easy to introduce artificial faults that are equivalent to inherent faults in difficulty of discovery. In general, it is much easier to find the seeded faults. Consequently, the number of inherent faults is usually underestimated with this technique.

(ii) *Iterated testing.* An other approach is suggested in NAGEL & SKRIVAN (1982) and NAGEL *et al.* (1984). They investigate software reliability by what they call replicated testing or repetitive run experimentation. The idea is to test a large number of identical copies of a software program simultaneously and independently of each other. In this way one can calculate the average number of bugs found as a function of time very precisely. This function can be used for the prediction of the number of bugs that will be found before a certain point in time in the future. There exists, however, a stronger motivation for this approach. Obviously, there will be larger and smaller faults in the software program. Larger faults tend to cause software failures earlier than the smaller ones. A consequence of this is that the larger faults tend to be present in most of the

test runs, while smaller faults occur only in a few of them. The key issue of this method is that one can obtain a good empirical estimation of the distribution of the occurrence rates of the different bugs. This information enables one to model the growth of reliability in an appropriate way. A large disadvantage of this approach, however, is that to be of any use iterated testing is enormously time consuming and therefore rarely used in practical situations.

(iii) *Static complexity analysis.* Following a completely other direction, one can estimate the number of faults from program characteristics only. It has long be assumed that the size of a program has the most effect on the number of inherent faults it contains. AKIYAMA (1971), THAYER (1976), MOTLEY & BROOKS (1977) and FEUER & FOWLKES (1979) have verified this hypothesis. They all more or less consider models in which the number of inherent faults is proportional to some power of the program length. Possibly, other measures of program complexity than just program length can improve the prediction of number of inherent faults. Complexity measures form an active current research area. Very popular are McCabe's cyclomatic number (MCCABE (1976)) and Halstead's effort (HALSTEAD (1977)). Other metrics can be found in GRADY & CASWELL (1987). However, most of the complexity measures developed to date show a high correlation with program size. They provide little improvement over just program size alone in predicting inherent faults remaining at the start of system test (SUNOHARA *et al.* (1981)).

(iv) *Error-counting and debugging models.* We consider the following experiment. A computer program is tested for a specified length of time. Inputs are selected randomly from the input-space in a way that is representative for the operational profile. Either the program produces the correct output, or a software failure occurs. That is, the software produces the wrong answer or no answer at all. After the detection of a failure, the CPU-clock is stopped and the program is sent to a team of debuggers. When the fault is found and fixed, available data concerning fault and failure are gathered in a database. After this, the CPU-clock is started again and testing continues with a new input until the end of the test period is reached. Among others, the following data-items are of interest: (a) The failure times, times at which the failures occur. This could be measured in seconds CPU-time, days real-time or even by the sequence number of the test input. (b) A description of the failure (or of the priority of its effects), so that a classification of the effects of errors is possible. It can be of use to distinguish failure intensities of different types of failures. (c) A description of the fault (the cause, like errors in specification, design or code). Also here a classification can make sense. Certain kinds of faults can indicate for instance a lack of accuracy or knowledge of the programmer or the misinterpretation of possibly ill-posed specifications. One should try to prevent such systematic errors in future. (d) The lo-

cation of the fault in the source text. When the program consists of a number of modules, it is possible to find local differences in the failure intensity and hence to concentrate the testing effort in the most critical regions. (e) A measure of the size of the correction of a fault (for example in bytes or new lines added or changed code or in man-hours). When modelling imperfect repair (that is, there exists a positive chance of introducing new faults when repairing an old one) such information would be very useful. Only the failure times are essential to be able to conclude something about the evolution of the reliability during the testing process. The class of Error-Counting and Debugging Models consists of relatively simple models, considering the test experiment as described above, characterised by the fact that they are only based on certain test data, such as the occurrence times of failures. These error-counting and debugging models do not explicitly depend on factors like the length and the structure of the program, the language in which it is written, the skill of the programmer, etcetera. By using the information obtained from the experiment one can estimate the parameters of the underlying model, in particular the total number of faults initially present in the software. Certain functions of these model-parameters will yield estimates of other interesting quantities (such as the failure intensity, the reliability, the mean time between failures and the release time). In practice, however, decisions about when to stop testing are rarely based solely on critical values for such quantities. More often to find an optimal stopping time, the reliability model is extended by associating cost functions, modelling the cost of testing versus the costs of faults in the field. An optimal stopping rule will tell to stop testing as soon as the cost of discovering and fixing the remaining faults is greater than the cost of repairs in the field.

As stated earlier, approaches (i) and (ii) are nice theoretical concepts, but they have considerable disadvantages that make them less practical. The static approach (iii) and the dynamic approach (iv) are both used in practice. In this paper we restrict ourselves mainly to the mathematical and statistical aspects of approach (iv), that is of error-counting and debugging models.

1.4. Assumptions and models

Efforts in describing the evolution of the reliability of computer software during testing resulted in the proposal of dozens of error-counting and debugging models over the past twenty years. Each individual model is completely characterised by a certain set of assumptions. Sometimes, we assume that failures in the software will occur independently and that when a failure is detected, the fault is fixed immediately with no new faults introduced. This is the case for some very well-known models: the *Jelinski-Moranda (JM) model* (Jelinski & Moranda (1972)), the *Goel-Okumoto (GO) model* (Goel & Okumoto (1979)) and the *Littlewood (L) model* (Littlewood (1980)).

The JM model is the oldest and one of the most elementary software reliability models introduced so far. In this model the failure rate of the program

is at any time proportional to the number of remaining faults and each fault makes the same contribution to the failure rate.

In the GO model the failures occur according to a non-homogeneous Poisson process. The failure rate does not depend on the debugging process; it is a simple deterministic function which decreases exponentially in time. Both the JM model and the GO model are in some sense special cases of a more general model, the model of Littlewood.

The main difference with respect to the two previous models is the fact that Littlewood does not assume that each fault makes the same contribution to the failure intensity. He allows each fault to have its own occurrence probability. Littlewood's argument for this is that larger faults will produce failures earlier than small ones. The way, however, in which Littlewood assigns occurrence rates to the faults, is rather *ad hoc*. In practice, it turns out that for many data-sets estimates based on the L model are not better than those based on the JM model.

Assumptions like independence of the occurring faults, negligible repair time and perfect repair are of course, not very realistic. It is unknown how large the influence on the results is of such an assumption as the independence. Without this assumption, however, the mathematical problem becomes a lot more complicated. With respect to the assumption of negligible repair time (that is: stop CPU-clock when failure detected) one can add that there are ways of transforming execution-time models into real-time models (MUSA (1975)). Moreover, immediate repair is not essential if we take care to count failures due to the same software fault only once. A new and interesting idea seems to be the modelling of imperfect repair and software growth simultaneously. With software growth we mean the phenomenon that a piece of software is not a static object, but on the contrary changes in time. Not only does each repair cause a change in the software, but also in practice at certain moments in the testing phase we will add new modules to the software as well.

In the *Poisson Growth & Imperfect Repair (PGIR) model* (VAN PUL (1991)) it is assumed that the expected number of new faults introduced at a certain point in time, is proportional to the size of the change in the software at that moment. This assumption makes it possible to model imperfect repair and software growth simultaneously. Besides, the model will account for dependencies between faults.

A more formal treatment of these software reliability models will be given in Section 2.4, after we have introduced the necessary mathematical notations and concepts. Finally, for a complete chronological catalogue of the most popular software reliability models introduced since 1972 we refer to MUSA *et al.* (1987).

2. MATHEMATICAL FRAMEWORK

An important statistical problem is the comparison of different models. This is usually done by goodness of fit testing. The test statistics involved are in our situation rather complex, and the derivation of their distributions can cause

considerable difficulties.

Not only the choice of the best model does confront us with many questions, but also the estimation of the parameters in the chosen model is a difficult problem. We usually use the maximum likelihood estimation procedure for this purpose. We derive the chance (likelihood) to get the data under the parameters and maximise this likelihood as a function of the parameters. The derivation of the likelihood function is not always possible analytically and the numerical computation of its maxima can be unstable. In practice we will use iterative techniques to approximate the roots of the system of likelihood equations.

The sequel of this paper will consider the problem of parameter estimation. First, we give some further relevant mathematical background. In Section 2.1 we present some important counting process theory results. We shed some light on the way we treat asymptotics and limit theory in Section 2.2. In Section 2.3 we describe the method of maximum likelihood estimation in the context of counting processes and discuss some of the main statistical problems in parameter estimation. We illustrate the theoretical concepts with some examples and place the software reliability models, mentioned in Section 1.4, in a mathematical context in Section 2.4.

2.1. Counting process theory

We are going to model the occurrence of discrete, random events in continuous time. We fix $\mathcal{T} = [0, \tau]$ for a given finite terminal time τ, $0 < \tau < \infty$. Recalling approach (iv) of Section 1.4, note that we are observing a non-deterministic process through the fixed time window $\mathcal{T}$. The fact that the number of faults detected in $\mathcal{T}$ will be stochastic is the reason why we cannot use classical maximum likelihood theory for i.i.d. observations in deriving asymptotic results. Therefore we introduce a powerful mathematical instrument which we will use to solve these problems: the theory of counting processes and martingales. For a complete summary we refer to ANDERSEN & BORGAN (1985), JACOD & SHIRYAEV (1987) or ANDERSEN *et al.* (1993). Before we are able to introduce the important notions of martingales, counting processes and their intensities, we have to give some other definitions first.

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. A *filtration* or *history* $(\mathcal{F}_t : t \in \mathcal{T})$ is an increasing, right-continuous family of sub σ-algebras of $\mathcal{F}$. The σ-algebra $\mathcal{F}_t$ is interpreted as follows: it contains all events whose occurrence or not is fixed by time t. We write correspondingly $\mathcal{F}_{t-}$ for the available data just before time t. A *stochastic process* X is just a time-indexed collection of random variables $(X(t) : t \in \mathcal{T})$. The process X is called *adapted* to the filtration if $X(t)$ is $\mathcal{F}_t$-measurable for each t and *cadlag* if its sample paths $(X(t, \omega) : t \in \mathcal{T})$ for almost all ω are right continuous with left-hand limits. The set of all cadlag functions on $\mathcal{T}$ is often denoted by $D(\mathcal{T})$, the Skohorod space of weak convergence theory. See BILLINGSLEY (1968). The *self-exciting filtration* $\mathcal{F}_t$ of a stochastic process X is the σ-algebra generated by $X(s), s <= t$. Finally, a stochastic process X is called *integrable* if for all $t \in \mathcal{T}$

$$\mathbb{E}\big(|X(t)|\big) < \infty$$

and *predictable* if as a function of $(t,\omega) \in \mathcal{T} \times \Omega$, it is measurable with respect to the σ-algebra on $\mathcal{T} \times \Omega$ generated by the left continuous adapted processes.

So suppose a filtration $(\mathcal{F}_t : t \in \mathcal{T})$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is given. A *martingale* is a cadlag, adapted stochastic process m which is integrable and satisfies the martingale property:

$$\mathbb{E}\big(m(t)|\mathcal{F}_s\big) = m(s), 0 \leq s \leq t. \tag{1}$$

That is, the increment of the stochastic process $m(t)$ over an arbitrary time interval $(t, t+h]$ given the past has zero expectation. A *counting process* n is a stochastic process which can be thought of as registering the occurrences in time of a number of discrete events. More formally, a counting process is an adapted cadlag process, zero at time zero, with piecewise constant and non-decreasing paths, having jumps of size one only. We say that n has *intensity process* λ, if λ is a predictable process and

$$m(t) := n(t) - \int_0^t \lambda(s)ds \tag{2}$$

satisfies the martingale property (1). The integral in the right-hand side of (2) is often referred to as the *cumulative intensity process* or *compensator* of n. We can consider a martingale as being a pure noise process. The systematic part of a counting process is its compensator, a smoothly varying and predictable process, which subtracted from the counting process leaves unpredictable zero-mean noise. Though m is pure noise, m^2 has a tendency to increase over time. The systematic component (compensator) of m^2 is called the *predictable variation process* of m and denoted by $< m >$. More generally, for martingales m_1 and m_2 the *predictable covariation process* $< m_1, m_2 >$ is defined as the unique finite variation cadlag predictable process such that $m_1 m_2 - < m_1, m_2 >$ is a martingale, zero at time zero. If h_1 and h_2 are predictable processes, then $\int h_1 dm_1$ and $\int h_2 dm_2$ are martingales and

$$< \int h_1 dm_1, \int h_2 dm_2 > = \int h_1 h_2 d < m_1, m_2 >.$$

Martingales have been studied intensively during the past few decades and a lot of nice mathematical properties have been derived by now. Some very important martingale results are Kurtz' theorems, Lenglart's inequality and the Martingale Central Limit Theorem (MCLT), which can be seen as analogues of the law of large numbers and the usual Central Limit Theorem in the classical i.i.d. case. These results, which will be essential in the proofs of in probability and weak convergence for the non-i.i.d. case, will be stated explicitly in the next section. For a more comprehensive treatment of these and other martingale results we refer to ANDERSEN *et al.* (1993).

REMARK 1. To get a better understanding of these theoretical concepts we close this section by constructing a special class of counting processes: the order statistics processes. It will turn out that this class contains most of (but not all) the popular software reliability models investigated so far. Let us consider a sample of N independent, identically distributed survival (or failure) times, $S_1, \ldots, S_N$, from a continuous survival function $S(t)$ with hazard rate function $z(t)$. Hence $z(t) = f(t)/(1 - F(t))$ where $F(t)$ is the cumulative distribution function and $f(t)$ the density of the S_i. Typically in survival analysis problems, complete observation of $S_1, \ldots, S_N$ is not possible. In our situation one observes only those S_i that occur in a fixed time interval $[0, \tau]$. We therefore define the counting process $n(t)$ for $t \in [0, \tau]$ as

$$n(t) := \#\{i : S_i \leq t\} = \sum_{i=1}^{N} I\{S_i \leq t\},$$

where $I\{.\}$ denotes the indicator function. Thus the stochastic process $n(t)$ is a non-decreasing integer valued function of time with jumps of size one only; it is right continuous and $n(0) = 0$. Furthermore we define the stochastic process $Y(t)$, $t \in [0, \tau]$, by

$$Y(t) := \#\{i : S_i \geq t\} = \sum_{i=1}^{N} I\{S_i \geq t\} = N - n(t-).$$

Hence $Y(t)$ is the number at risk just before time t, or the size of the risk set. We define the intensity process $\lambda(t)$, the rate at which the counting process $n(t)$ jumps, as:

$$\lambda(t) := Y(t)z(t) = \big(N - n(t-)\big)\, z(t). \tag{3}$$

Note that the intensity process λ is random, through dependence on the past of the stochastic process n. Given $\mathcal{F}_{t-} := \sigma\{n(s), s < t\}$, the strict past of n, however, λ is a predictable process: that is to say, given $\mathcal{F}_{t-}$ we know $\lambda(t)$ already, but not yet $n(t)$ for instance. It is not difficult to check that in this case the process

$$m(t) := n(t) - \int_0^t \lambda(s)ds$$

indeed satisfies the martingale property (1). Counting process models, counting the occurrences of i.i.d. events, and hence having an underlying intensity of the form (3), we will call $N - n$ homogeneous.

2.2 Asymptotics and limit theory

A major part of Section 3 will deal with the study of asymptotic properties of estimators. Therefore we should make clear how we treat asymptotics; especially as our approach is rather unorthodox. A novel aspect of our approach,

namely, is the fact that – in order to treat asymptotic theory – instead of increasing the time variable or the number of data as is usually the case, we (conceptually) increase one of the model parameters itself.

Recall the formulation of the mathematical problem as sketched in Section 2.1. On a fixed time-interval $\mathcal{T} = [0, \tau]$, we are observing a counting process n with underlying intensity process λ. In the sequel we always assume that this stochastic intensity function, depending on the past of n, is a member of some parametric family:

$$\lambda(t) := \lambda\big(t, \theta, n(t-)\big), \theta \in \Theta \subseteq \mathbb{R}^k.$$

We assume the true parameter-value θ_0 is contained in Θ. In all typical cases $\theta_0 = (N_0, \psi_0)$, where N_0, the parameter of most interest, represents the scale or size of the problem (sometimes $N_0 = n(\infty)$), while ψ_0 is a nuisance vector parameter. To apply asymptotics it obviously would not make sense to let *tau*, the stopping time, grow to infinity. In the long run all failure times will have occurred before time *tau* and the estimate of the total number of faults N will trivially be equal to the true number of faults. It makes more sense to (conceptually) increase the number of faults N, itself. As we are particularly interested in parameter estimation when N is large, we consider the reparametrisation

$$N = \nu\gamma, \ \nu \to \infty. \tag{4}$$

In (4), ν denotes the known scale of the problem (size of the population or some complexity measure of the software program), which we let go to infinity. Furthermore, γ represents the unknown proportion coefficient (of ill people or software bugs), which we are going to estimate. In order to make the parametrisation (4) profitable we have to put a constraint on λ. $N - n$ homogeneity (i.e. (3)) would be sufficient but is too strong. We require only that the intensity function λ is simultaneously linear in both N and n, that is:

$$\lambda\big(t, (N, \psi), n(t-)\big) = \alpha^{-1}\lambda\big(t, (\alpha N, \psi), \alpha n(t-)\big), \alpha > 0. \tag{5}$$

Models with intensity function satisfying (5) will be called (N, n) *homogeneous*. We now consider a sequence of counting processes n_ν with increasing ν and underlying (N, n) homogeneous intensities λ_ν and define for $t \in [0, \tau]$

$$x_\nu(t) := \nu^{-1} n_\nu(t).$$

The idea behind the transformation (4) and (N, n) homogeneity (5) is that $\nu^{-1}\lambda_\nu$ only depends on ν via x_ν. We therefore define

$$\beta\big(t, (\gamma, \psi), x_\nu(t-)\big) := \nu^{-1}\lambda_\nu\big(t; (\nu\gamma, \psi), \nu x_\nu(t-)\big).$$

A function $\beta : \mathcal{T} \times \mathcal{D}(\mathcal{T}) \to \mathbb{R}$ is called *non-anticipating* if $\beta(t, x)$ depends only on $x_{|[o,t)}$, the past of the stochastic process x up to but not including time t. It will turn out to be of great importance that under weak smoothness conditions on β, that are readily satisfied for the models most used in practice, the stochastic process $x_\nu(t)$ will converge in probability uniformly on $[0, \tau]$ to a deterministic function $x_0(t)$, which is the solution of the integral equation

$$x(t) = \int_0^t \beta\big(t, (\gamma, \psi), x(s-)\big) ds. \tag{6}$$

This will follow directly from the next theorem:

THEOREM 1. *Let $\beta(t; x)$ be a non-anticipating and non-negative function of $t \in \mathcal{T}$ and $x \in D(\mathcal{T})$. We assume that*

$$\sup_{s \le t} \beta(t, x(s-)) \le C_1 + C_2 \sup_{s<t} |x(s)|,$$

$$\sup_{s \le t} |\beta(t, x(s-)) - \beta(t, y(s-))| \le C_3 \sup_{s<t} |x(s) - y(s)|,$$

for all x, $y \in D(\mathcal{T})$, and for certain constants C_1, C_2 and C_3. Let $a_\nu \to \infty$ be a sequence of positive constants. Let n_ν be a counting process with underlying intensity process $\lambda_\nu(t) = a_\nu \beta(t, a_\nu^{-1} n_\nu(t-))$, for all $t \in \mathcal{T}$. Finally, let x_0 be the unique solution of the integral equation

$$x(t) = \int_0^t \beta(t, x(s-)) ds.$$

Then we have for all $t \in \mathcal{T}$ as $\nu \to \infty$:

$$\sup_{s<=t} |a_\nu^{-1} n_\nu(t) - x_0(t)| \xrightarrow{\mathbb{P}} 0.$$

□

This theorem, also known as Kurtz' first theorem, is just the law of large numbers for counting processes. Kurtz' second theorem states that under slightly stronger conditions a central limit theorem result can be obtained, i.e.: $\sqrt{\nu}(n_\nu/\nu - x_0) \to Z$ in distribution where Z is a Gaussian process. Both results can be found in KURTZ (1983). One of our main goals in Section 3 will be to find estimators for the model parameters and to derive asymptotic properties such as consistency and asymptotic normality. An estimator θ_ν for θ_0 is said to be *consistent* if $\theta_\nu \to \theta_0$ in probability and *asymptotically normal* if $\sqrt{\nu}(\theta_\nu - \theta_0) \to \mathcal{N}(0, \Sigma)$ in distribution. When deriving these kind of properties in a non-i.i.d. situation we will need the following important martingale results:

THEOREM 2. *Let m be a local square integrable martingale. Then for any $\eta > 0$ and $\delta > 0$ we have:*

$$\mathbb{P}\big(\sup_{t \in \mathcal{T}} |m| \ge \eta\big) \le \frac{\delta}{\eta^2} + \mathbb{P}\big(< m > (\tau) \ge \delta\big).$$

□

THEOREM 3. *Consider a sequence n_ν of counting processes with intensity process λ_ν and a sequence of predictable processes H_ν. Define for $t \in [0, \tau]$,*

$$Z_\nu(t) := \int_0^t H_\nu(s)\big(dn_\nu(s) - \lambda_\nu(s)ds\big).$$

Suppose as $\nu \to \infty$ *for all* $t \in \mathcal{T}$*:*

$$< Z_\nu > (t) \xrightarrow{\mathbb{P}} G(t), \tag{7}$$

where G *is a continuous function, and suppose that for all* $\epsilon > 0$ *as* $\nu \to \infty$*:*

$$\int_0^\tau H_\nu^2(t)\lambda_\nu(t)I\{|H_\nu(t)| > \epsilon\}dt \xrightarrow{\mathbb{P}} 0. \tag{8}$$

Then Z_ν *converges in distribution to* Z *in the space* $D(\mathcal{T})$*, where* Z *is a Gaussian martingale with variance function* G *and* $Z(0) = 0$. □

Theorem 2, Lenglart's inequality, tells us that we can bound the probability of a large value of m anywhere in the whole time-interval $\mathcal{T}$ in terms just of the probability of a large value of $< m >$ in the endpoint τ. One says that m is *dominated* by $< m >$. See LENGLART (1977). Theorem 3 is a special case of the martingale central limit theorem (MCLT), saying that two conditions are required for a local square integrable martingale to be approximately Gaussian. Condition (7) states that its predictable variation process converges in probability to a deterministic function; condition (8) states that the jumps of Z_ν become small as $\nu \to \infty$. For more general formulations of the MCLT and proofs we refer to REBOLLEDO (1980) or HELLAND (1982).

2.3. Maximum likelihood estimation

We observe the counting process $n(t)$ on $[0, \tau]$ with underlying (N, n) homogeneous parametric intensity process

$$\lambda(t) = \lambda\big(t, (N, \psi), n(t-)\big). \tag{9}$$

The question is now, of course, how to find estimators for N and ψ. We will use the method of Maximum Likelihood Estimation (MLE) for this purpose. Using the fact that $\lambda(t)dt$ represents the conditional probability given the strict past that the counting process $n(t)$ jumps in the interval $[t, t + dt]$, we can write for the likelihood:

$$\begin{aligned} L_\tau(N, \psi) &\propto \prod_{0<t<\tau} \big((\lambda(t)dt)^{dn(t)}(1 - \lambda(t)dt)^{1-dn(t)}\big) \\ &\propto \exp\Big(\int_0^\tau \log\lambda(t)dn(t) - \int_0^\tau \lambda(t)dt\Big). \end{aligned} \tag{10}$$

For a standard definition of the product integral in the upper expression of (10) we refer to GILL & JOHANSEN (1990). The lower expression in (10) is also known as Jacod's formula (ANDERSEN *et al.* (1993)).

REMARK 2. Let us again consider the special class of order statistics processes described in Remark 1. Thus, given is a population consisting of an unknown number N of failure times S_i. We now furthermore assume that it is given that these failure times are taken from some continuous, parametric distribution function, say $F = F_\psi$. We only observe those failure times S_i, which take a value in $[0, \tau]$. That is, we observe only the order statistics $T_i := S_{(i)}, i = 1 \ldots n$, where n is random, and know that $S_{(n+1)} > \tau$. Equivalently stated in terms of counting processes, we observe the counting process $n(t)$ on $[0, \tau]$ with underlying intensity process

$$\lambda(t) = \big(N - n(t-)\big) z(t, \psi), \tag{11}$$

where $z(t, \psi) = f_\psi(t)/(1-F_\psi(t))$. Again the problem is, how to estimate N and ψ. Of course, we could again use Jacod's expression for the likelihood, but we could also follow the classical approach. Conditioned on the fact that the failure times S_i are i.i.d. F_ψ, the total number of failures in $[0, \tau]$ namely is binomially distributed with parameters N and $F_\psi(\tau)$. Furthermore, conditioned on the fact that S_i is observed, it has a truncated distribution $F_\psi(t)/F_\psi(\tau)$. This leads to the following alternative expression for the likelihood function:

$$L_\tau(N, \psi) \propto \binom{N}{n} F_\psi(\tau)^n (1 - F_\psi(\tau))^{N-n} n! \prod_{i=1}^{n} \frac{f_\psi(T_i)}{F_\psi(\tau)}, \tag{12}$$

where the extra factor $n!$ in front of the product of truncated densities in (12) is explained by the fact that the T_i are the order statistics of the S_i.

Maximisation of expressions (10) (or (12)) is usually done by setting partial derivatives of the log-likelihood to zero and solving the resulting system of highly non-linear likelihood (or score) equations:

$$\frac{\partial}{\partial N} \log L_\tau(N, \psi) = 0, \tag{13}$$

$$\frac{\partial}{\partial \psi} \log L_\tau(N, \psi) = 0. \tag{14}$$

We have assumed in (13) that the model is also meaningful for non-integer N. The direct algebraic solution of the system of non-linear equations (13)–(14) is usually impossible. The best we can realistically hope for is to solve these equations for a subset of the parameters in terms of the remaining parameters. The remaining parameters are then estimated using numerical methods. The *Newton-Raphson procedure* (CARNAHAN & WILKES (1973)) is based on first order Taylor series expansions and has two attractive features. First, if the method converges, it will do so very fast (quadratically). Secondly, convergence is assured if the initial estimate is close enough to a root of the system. This also represents the main drawback of the method: for some initial estimates the method will diverge. This problem becomes severe as the region of feasible

values is infinite and high dimensional. Other iterative methods using gradient information (FLETCHER & REEVES (1964)) often also do not display suitable convergence in software reliability practice. In fact, investigations (MUSA *et al.* (1987)) indicated that searching schemes incorporating gradient information are not particularly well suited for the problem at hand. The *downhill simplex method* (NELDER & MEAD (1965)) does not use any gradient information; it uses only function evaluations (no derivatives). It is very robust and nearly always converges to at least a local minimum (of $-\log L_\tau(N,\psi)$). The main drawback here is that the method lacks any form of acceleration, per se, and therefore tends to be slow. MUSA *et al.* (1987) suggest to combine the methods of Newton-Raphson and Nelder-Mead. The idea is that after a limited number of Nelder-Mead iterations the estimate of the solution is close enough to the roots of the system of equations for the fast Newton-Raphson method to converge. Some care should be exercised that the procedure did not accidentally locate a minimum or a saddle-point. More seriously, we should make sure that a global maximum and not a local one has been found. The latter concern can be a problem when using a small sample (of failure times) to estimate multiple parameters simultaneously. Here it is a good idea to have the procedure pick several different starting values and use the estimate with the largest value of $\log L_\tau(N,\psi)$. Once the sample size is reasonably large, multiple solutions to the system of likelihood equations (13)–(14) are rare in typical software reliability models (MOEK (1984), MUSA *et al.* (1987)), but not impossible (BARENDREGT & VAN PUL (1994)).

2.4. Some software reliability models revisited

In this section we discuss the software reliability models, briefly described in Section 1.4, in more detail. In all examples we assume that software failures are observed during a fixed time-interval $[0,\tau]$ only. We mean by T_i the failure time of the i-th occurring failure, while $t_i = T_i - T_{i-1}$ denotes the interfailure time; that is the time between the $(i-1)$-th and i-th failure. The unknown number of faults initially present in the software is denoted by N_0.

EXAMPLE 1. *The Jelinski-Moranda model (JM).* In the JM-model introduced by JELINSKI & MORANDA (1972) the failure rate of the program is at any time proportional to the number of remaining faults and the removal of each fault makes the same contribution to the decay in failure rate. So in terms of counting processes we can write:

$$\lambda^{JM}(t) = \beta_0\big(N_0 - n(t-)\big),$$

where $\lambda(t)$ denotes the failure rate at time t and where $n(t-)$ denotes the cumulative number of faults detected up to but not including t. The interfailure times t_i are independent and exponentially distributed with parameter $\lambda_i = \phi_0(N_0 - i + 1)$. Defining

$$\beta^{JM}(t,(\gamma,\phi),x) := \phi\big(\gamma - x(t-)\big),$$

it is easy to check that the JM-model is (N, n) homogeneous, that is has a failure intensity λ of the special form (5). Solving the integral equation (6) yields:

$$x_0^{JM}(t) := \gamma\big(1 - e^{-\phi t}\big).$$

In fact, the JM intensity is even $N-n$ homogeneous (of the special form (2.3)) with parametric hazard

$$z^{JM}(t, \phi) := \phi,$$

associated to the exponential distribution. Hence in this case the failure times T_i can be considered as the order statistics of independent and identically distributed exponentials with parameter ϕ. Using (10) we can write down the log-likelihood for the JM model:

$$\log L(N, \phi) = \sum_{i=1}^{n(\tau)} \log \phi(N - i + 1) - \sum_{i=1}^{n(\tau)+1} \phi(N - i + 1)t_i,$$

hence the likelihood equations become

$$\frac{\partial}{\partial N} \log L(N, \phi) = \sum_{i=1}^{n(\tau)} \frac{1}{N - i + 1} - \phi\tau = 0; \tag{15}$$

$$\frac{\partial}{\partial \phi} \log L(N, \phi) = \frac{n(\tau)}{\phi} - \sum_{i=1}^{n(\tau)+1} (N - i + 1)t_i = 0. \tag{16}$$

It was shown by Moek (1983) that this system of equations will have a unique solution $(\hat{N}, \hat{\phi})$ if and only if the data satisfy:

$$\frac{1}{\tau} \sum_{i=1}^{n(\tau)+1} (i - 1)t_i > \frac{n(\tau) - 1}{2}.$$

Moek's criterion will be satisfied with probability one as N_0 grows larger, if the model is true.

Example 2. *The Goel-Okumoto model (GO).* In the GO model, suggested by Goel & Okumoto (1979), the failures occur according to a non-homogeneous Poisson-process with failure rate

$$\lambda^{GO}(t) := N_0 \phi_0 e^{-\phi_0 t}.$$

Notice that λ does not depend on n; it is a simple deterministic function of time. One can check that the expected number of failures in $[0, \infty)$ equals

$$\mathbb{E}\Big(\int_0^\infty \lambda^{GO}(s)ds\Big) = N_0.$$

Thus we have N_0 faults or sources of failures, each producing failures at an exponentially decreasing rate. The GO model is obviously not $N-n$ linear, but with

$$\beta^{GO}(t,(\gamma,\phi),x) := \gamma\phi e^{-\phi t},$$

the GO model satisfies (5), having the same deterministic solution x_0 of (6) as in the JM case. This means that the JM and GO model are asymptotically equivalent (*indistinguishable*). In fact, the models are indistinguishable in a stronger sense, since the distribution of the process n in the GO model, conditional on $n(\infty)=k$, is the same as the distribution of the process n in the JM model with $N_0=k$. This means that on the basis of one realisation you cannot distinguish between the models at all. Note that in the GO model the failure rate never becomes zero. This is supposed to reflect the fact that a detected error may or may not be removed and may cause additional errors. However, the exponential hazard rate is completely arbitrary and we feel that the JM model is much more realistic than the GO model. Perhaps the GO model should be considered as an easily analysable approximation to the JM model.

EXAMPLE 3. *The Littlewood model (L).* The main difference in the L model, introduced by LITTLEWOOD (1980), is the fact that each fault does not make the same contribution to the failure rate λ. He treats the occurrence rate ϕ_i of an individual bug as a stochastic variable. Littlewood's model is an empirical Bayesian model and he himself suggests a gamma distribution $\Gamma(a_0,b_0)$ for the a-priori probability distribution of the ϕ_i. It can be derived that the failure rate of the program at time t is then given by

$$\lambda^L(t) := \frac{a_0}{b_0+t}\big(N_0 - n(t-)\big).$$

So as in the JM model, λ depends on the past of the counting process n. Again the failure intensity is (N,n) homogeneous and with

$$\beta^L(t,(\gamma,\alpha,\beta),x) := \frac{a}{b+t}\big(\gamma - x(t-)\big)$$

the solution of the integral equation (6) becomes

$$x_0^L := \gamma\big(1-(1-t/b)^{-a}\big).$$

Moreover, λ^L is of the special form (3), with hazard $z(t,a,b)=a/(b+t)$ and the associated distribution is the Pareto-distribution (mixture of exponentials).

3. SOME RECENT RESULTS

In this section we give an overview of some of the theoretical results that were recently presented in VAN PUL (1993). In Section 3.1 a rather general class of parametric intensity functions is considered. Following the lines of BORGAN (1984) sufficient (but weak) conditions are derived under which

some important asymptotic properties (i.e. consistency, asymptotic normality and efficiency) of the maximum likelihood estimators can be proved.

In Section 3.2 we prove that for this general class of intensity functions the parametric bootstrap works, i.e. is asymptotically consistent. Furthermore we investigate how well the maximum likelihood estimators behave in practice using simulated data according to the Jelinski-Moranda model and we compare the coverage percentage of confidence intervals constructed with the asymptotic normal and the Wilks test statistic and one using the bootstrap method.

Finally, of course, one of our ultimate goals will be the study of more realistic models. In Section 3.3 we present the so-called Poisson Growth & Imperfect Repair model (PGIR) (VAN PUL (1991)). We combined the modelling of imperfect repair and software growth in a natural way. Furthermore to a certain extent the model will account for dependencies between faults. The model has attractive statistical properties besides.

3.1. Asymptotic properties of the MLE

We consider a sequence of models $(\lambda_\nu, m_\nu, x_\nu), \nu = 1, 2, \ldots$ as defined in the previous section. For reasons of notational convenience we take $\theta := (\gamma, \psi)^T \in \Theta, \Theta \subset \mathbb{R}^p$ for some integer p. In the sequel we assume that the intensity function λ_ν is of the form:

$$\lambda_\nu(t;\theta) = \nu\beta(t,\theta,x_\nu), \tag{17}$$

where $\beta : [0,\tau]\times\Theta\times K \to \mathbb{R}^+$ is an arbitrary non-negative and non-anticipating function. In fact, in most practical cases $\beta(t,\theta,x_\nu)$ will depend only on $x_\nu(t-)$. On $K := D([0,\tau])$, the space of right-continuous functions on $[0,\tau]$ with left limits (so-called *cadlag* functions), we put the usual supremum norm. The likelihood function $L_\nu(\theta,t)$ now becomes for $\theta \in \Theta$, $t \in [0,\tau]$ and $\nu = 1, 2, \ldots$:

$$L_\nu(\theta,t) := \exp\Big(\int_0^t \log\nu\beta(s,\theta,x_\nu)dn_\nu(s) - \nu\int_0^t \beta(s,\theta,x_\nu)d"s\Big). \tag{18}$$

Furthermore, we define for $\theta \in \Theta, t \in [0,\tau]$, $\nu = 1, 2, \ldots$:

$$C_\nu(\theta,t) := \log L_\nu(\theta,t), \tag{19}$$

$$U_{\nu i}(\theta,t) := \frac{\partial}{\partial\theta_i}C_\nu(\theta,t), \tag{20}$$

$$I_{\nu ij}(\theta,t) := \frac{\partial^2}{\partial\theta_i\partial\theta_j}C_\nu(\theta,t), \tag{21}$$

$$R_{\nu ijk}(\theta,t) := \frac{\partial^3}{\partial\theta_i\partial\theta_j\partial\theta_k}C_\nu(\theta,t). \tag{22}$$

Consider the following global conditions:

(G1) For all $x \in K$ and for all $\theta \in \Theta$ the intensity function β satisfies

$$\sup_{t \leq \tau} \beta(t, \theta, x) < \infty.$$

(G2) (*Lipschitz continuity*) For all $\theta \in \Theta$ there exists a constant L, such that for all $x, y \in K$ and all $t \in [0, \tau]$

$$|\beta(t, \theta, x) - \beta(t, \theta, y)| \leq L \sup_{s \leq t} |x(s) - y(s)|.$$

Under the global conditions (G1)–(G2) the stochastic process $x_\nu(t)$, as defined in Section 2.2, converges uniformly on $[0,\tau]$ in probability to $x_0(t)$ as $\nu \to \infty$, where $x_0 \in D([0, \tau])$ is the unique solution of

$$x(t) = \int_0^t \beta(s, \theta_0, x) ds.$$

This was proved by KURTZ (1983). Next, we consider the following local conditions:

(L1) There exist neighbourhoods Θ_0 and K_0 of θ_0, x_0 respectively, such that the function $\beta(t, \theta, x)$ and its derivatives with respect to θ of the first, second and third order exist, are continuous functions of θ and x, bounded on $[0, \tau] \times \Theta_0 \times K_0$.

(L2) The function $\beta(t, \theta, x)$ is bounded away from zero on $[0, \tau] \times \Theta_0 \times K_0$.

(L3) The matrix $\Sigma = \sigma_{ij}(\theta_0)$ is positive definite, with for $i, j \in 1,2,..,p$, $\theta \in \Theta_0$:

$$\sigma_{ij}(\theta) := \int_0^\tau \frac{\partial}{\partial \theta_i} \beta(s, \theta, x_0) \frac{\partial}{\partial \theta_j} \beta(s, \theta, x_0) \beta(s, \theta, x_0) ds. \tag{23}$$

We are now able to formulate the main result.

THEOREM 4. *Consider a counting process with intensity function $\lambda(t; N, \psi)$, where (N, ψ) denotes an unknown p-dimensional parameter. As in Section 2.2 we can define an associated sequence of experiments by letting $\nu \to \infty$. Let $\theta_0 = (\gamma_0, \psi_0)$ be the true value of the parameter. Assume that for all ν the intensity function $\lambda_\nu(t; \theta)$ in the ν-th experiment is of the form (17) for a certain function β satisfying conditions (G1)–(G2) and (L1)–(L3). Then we have:*

(i) Consistency of ML-estimators: With probability tending to 1, the likelihood equations

$$\frac{\partial}{\partial \theta} \log L_\nu(\theta, \tau) = 0, \nu = 1, 2, \ldots \tag{24}$$

have exactly one consistent solution $\hat{\theta}_\nu$. Moreover this solution provides a local maximum of the likelihood function (18).

(ii) Asymptotic normality (LAN) of the ML-estimators: Let $\hat{\theta}_\nu$ be the consistent solution of the maximum likelihood equations (24), then

$$\sqrt{\nu}(\hat{\theta}_\nu - \theta_0) \to_{D(\theta_0)} \mathcal{N}(0, \Sigma^{-1}), \nu \to \infty,$$

where Σ is given by (23) and can be estimated consistently from the observed information matrix I_ν, given in (21).

(iii) Local asymptotic normality (LAN) of the model: With $U_\nu = U_\nu(\theta_0, \tau)$ given by (20), we have for all $h \in \mathbb{R}^p$:

$$\log \frac{d\mathbb{P}_{\theta_\nu}}{d\mathbb{P}_{\theta_0}} - \nu^{-\frac{1}{2}} h^T U_\nu + \frac{1}{2} h^T \Sigma h \to_{\mathbb{P}_{\theta_0}} 0, \nu \to \infty, \tag{25}$$

where $\theta_\nu = \theta_0 + \nu^{-\frac{1}{2}} h$ and $\nu^{-\frac{1}{2}} U_\nu \to_D \mathcal{N}(0, \Sigma)$.

(iv) Asymptotic efficiency of the ML-estimators: $\hat{\theta}_\nu$ is asymptotically efficient in the sense that it is regular and the limit distribution for any other regular estimator $\tilde{\theta}_\nu$ for θ_0 satisfies

$$\sqrt{\nu}(\tilde{\theta}_\nu - \theta_0) \to_{D(\theta_0)} Z + Y,$$

where $Z \sim_d \mathcal{N}(0, \Sigma^{-1})$, Z and Y independent. (For a definition of the regularity of an estimator we refer to Van der Vaart (1987), Ibragimov & Khas'minskii (1979).

Proof. Although our model (17) is not a special case of the multiplicative intensity model considered in Borgan (1984), the proof of Theorem 4, which is given in van Pul (1993), follows the lines of Borgan (1984) and is omitted here. Borgan starts with slightly weaker conditions and uses the same standard argumentation as given by Cram'er (1946), who derived similar results for the classical case of independent, identically distributed (i.i.d) random variables. Compared with the i.i.d. case the difference is that in the present context Lenglart's inequality is used to establish the convergence in probability results (instead of the law of large numbers in the classical case), while we have to use the martingale central limit theorem to establish the weak convergence result, which in the classical case is proved by the central limit theorem for i.i.d. random variables. □

Remark 3. In van Pul (1993) it is shown that (25) still holds for sequences $\theta_\nu = \theta_0 + \nu^{-1/2} h + o(h)$, that is the model satisfies the strong local asymptotic normality (SLAN) property). Together with the asymptotic normality this implies strong regularity of the maximum likelihood estimator. □

REMARK 4. A nearly immediate consequence of these results about the asymptotic distribution of the maximum likelihood estimator $\hat{\theta}_\nu$ is the fact that the *Wald test statistic*

$$-(\hat{\theta}_\nu - \theta_0)^T I_\nu(\hat{\theta}_\nu, \tau)(\hat{\theta}_\nu - \theta_0),$$

with I_ν given by (21), is asymptotically chi-squared distributed with p degrees of freedom under the simple hypothesis $H_0 : \theta = \theta_0$. With C_ν, U_ν and I_ν given by (19)–(21) the *Rao test* (or *score*) *statistic*

$$-U_\nu(\theta_0, \tau)^T I_\nu(\theta_0, \tau)^{-1} U_\nu(\theta_0, \tau)$$

and the *Wilks test* (or *likelihood ratio*) *statistic*

$$2\big(C_\nu(\hat{\theta}_\nu, \tau) - C_\nu(\theta_0, \tau)\big) \tag{26}$$

have the same asymptotic distribution as the Wald test statistic. Equivalence of these tests can be shown by the arguments of RAO (1973).

3.2. Consistency of the bootstrap

Simulations of the Jelinski-Moranda model show (see VAN PUL (1992B)) that asymptotic convergence to the normal distribution is appearing very slowly and that for values of ν not extremely large the empirical distribution functions of the components of $\hat{\theta}_\nu$ can be significantly skew. Hence, confidence intervals based on approximate normal test statistics will turn out to be disappointing. One solution, which is already suggested in VAN PUL (1992A), is to make use of Wilks likelihood ratio test statistic,

$$2\big(\log L_\nu(\hat{\theta}_\nu, \tau) - \log L_\nu(\theta_0, \tau)\big), \tag{27}$$

which can be proved to be asymptotically $\chi^2(p)$ distributed and is often thought to have faster convergence.

Another way to deal with deviations from normality is to make use of bootstrap methods. Suppose we want to construct confidence intervals for a one-dimensional real parameter θ. The concept of parametric bootstrapping in the context of software reliability consists of simulating a so called *bootstrap counting process* according to the failure intensity $\lambda(t, \hat{\theta})$, where $\hat{\theta}$ is the maximum likelihood estimator for θ. Repeating this simulation experiment, say M times, we get bootstrap estimators $\hat{\theta}_i^*, i = 1, \ldots, M$. We define:

$$G_\nu := \mathcal{L}_{\theta_0}\big(\sqrt{\nu}(\hat{\theta}_\nu - \theta_0)\big), \tag{28}$$

$$G_\nu^* := \mathcal{L}_{\hat{\theta}_\nu}\big(\sqrt{\nu}(\hat{\theta}_\nu^* - \hat{\theta}_\nu)\big). \tag{29}$$

We will say that the parametric bootstrap *works* (or is *asymptotically consistent*) if and only if

$$\sup_{x \in \mathbb{R}} |G_\nu^*(x) - G_\nu(x)| \to_{\mathbb{P}_{\theta_0}} 0. \tag{30}$$

See also BICKEL & FREEDMAN (1981) and SINGH (1981). This result will be derived in the next theorem. Note that, as G_ν converges to a continuous distribution function, a consequence of (30) is that confidence intervals for θ based on G^*_ν will have asymptotically the right coverage probabilities:

$$\mathbb{P}\big(\hat\theta_\nu - \frac{Z^*_\nu(1-\frac{\alpha}{2})}{\sqrt{\nu}} \le \theta_0 \le \hat\theta_\nu - \frac{Z^*_\nu(\frac{\alpha}{2})}{\sqrt{\nu}}\big) \to 1-\alpha,$$

for $\nu \to \infty$, where $Z^*_\nu(\alpha) := G^{*-1}_\nu(\alpha)$. In practice one often uses *studentised* versions of (28) and (29), i.e.:

$$G^{ST}_\nu := \mathcal{L}_{\theta_0}\big(\frac{\sqrt{\nu}(\hat\theta_\nu - \theta_0)}{\hat\sigma_\nu}\big),$$

$$G^{ST*}_\nu := \mathcal{L}_{\hat\theta_\nu}\big(\frac{\sqrt{\nu}(\hat\theta^*_\nu - \hat\theta_\nu)}{\hat\sigma^*_\nu}\big),$$

expecting the second order terms of the Edgeworth expansions to be the same too (see for instance HELMERS (1991)). In this section we will determine $\hat\sigma_\nu$ and $\hat\sigma^*_\nu$ simply by substituting respectively $\hat\theta_\nu$ and $\hat\theta^*_\nu$ for θ_0 in the expected information matrix Σ, given by (23). An alternative way to estimate σ consistently, is to make use of the observed information matrix

$$I_\nu(\theta,\tau)\log L_\nu(\theta,\tau). \tag{31}$$

Supposing we are in the counting process context, sketched in the beginning of this section (and hence in particular in our software reliability situation) we can prove (30). The following two lemma's will be very useful:

LEMMA 1. *Under the conditions of Theorem 4, we have SLAN (strong local asymptotic normality), that is: there exist a sequence $U_\nu, \nu = 1,2,\ldots$ such that for all $h \in \mathbb{R}^p$:*

$$\log\frac{dP_{\theta_\nu}}{dP_{\theta_0}} - h^T U_\nu + \frac{1}{2}h^T\Sigma h \to_{\mathbb{P}_{\theta_0}} 0,$$

for $\nu \to \infty$, where $U_\nu \to_d \mathcal{N}(0,\Sigma)$, Σ given by (23), but now with

$$\theta_\nu = \theta_0 + \nu^{-1/2}h + o(\nu^{-1/2}). \tag{32}$$

□

LEMMA 2. *Under the conditions of Theorem 4, asymptotic normality and SLAN imply S-regularity (strong regularity), that is:*

$$\sqrt{\nu}\big(\hat\theta_\nu - \theta_\nu\big) \to_{D(\theta_\nu)} \mathcal{N}(0,\Sigma^{-1})$$

for all sequences θ_ν of the form (32). □

The proofs of Lemma's 1 and 2 are slight modifications of the proofs of Theorem 4, (iii)&(iv), given in VAN PUL (1993), and are therefore omitted here. We are now able to formulate the following result:

THEOREM 5. *The parametric bootstrap is asymptotically consistent.*

PROOF. The asymptotic normality of the MLE yields:

$$\lim_{\nu\to\infty} \sup_{x\in\mathbb{R}} |G_\nu(x) - G(x)| = 0,$$

where $G(x) := \mathcal{N}(0, \Sigma^{-1})$. So to prove (30) it is sufficient to show that for all $\epsilon > 0$:

$$\lim_{\nu\to\infty} \mathbb{P}\big(\sup_{x\in\mathbb{R}} |G^*_\nu(x) - G(x)| > \epsilon\big) = 0. \tag{33}$$

Defining $Z_\nu := \sqrt{\nu}(\hat{\theta}_\nu - \theta_0)$ the asymptotic normality assures that

$$Z_\nu \to_D Z$$

where $\mathcal{L}_{\theta_0}(Z) = G = \mathcal{N}(0, \Sigma^{-1})$. The almost-sure-representation theorem (see for instance POLLARD (1989)) states that there exist $\tilde{Z}_\nu =_d Z_\nu$ and a $\tilde{Z} =_d Z$ such that

$$\tilde{Z}_\nu \to_{a.s.} \tilde{Z}.$$

As (for fixed ν) θ_0 and $\sqrt{\nu}$ are constants, Z_ν is only a function of $\hat{\theta}_\nu$. So, we can write

$$\tilde{Z}_\nu := \sqrt{\nu}(\tilde{\theta}_\nu - \theta_0)$$

for some $\tilde{\theta}_\nu \in \mathbb{R}^p$, that is

$$\tilde{\theta}_\nu := \theta_0 + \nu^{-1/2}\tilde{Z}_\nu = \theta_0 + \nu^{-1/2}\tilde{Z} + o(\nu^{-1/2}), a.s.$$

Now the S-regularity of $\hat{\theta}_\nu$ gives under $P(\tilde{\theta}_\nu)$:

$$\sqrt{\nu}\big(\hat{\theta}_\nu - \tilde{\theta}_\nu\big) \to_D G, a.s.$$

or in other words

$$\tilde{G}^*_\nu := \mathcal{L}_{\tilde{\theta}_\nu}\big(\sqrt{\nu}(\hat{\theta}^*_\nu - \tilde{\theta}_\nu)\big) \to_D G, a.s.$$

But this is to say

$$\sup_{x\in\mathbb{R}} |\tilde{G}^*_\nu(x) - G(x)| \to_{a.s} 0,$$

which implies

$$\sup_{x\in\mathbb{R}} |\tilde{G}^*_\nu(x) - G(x)| \to_{\mathbb{P}} 0. \tag{34}$$

Because $\tilde{G}_\nu$ is a function of $\tilde{\theta}_\nu$ (or equivalently $\tilde{Z}_\nu$) only, and because $\tilde{Z}_\nu =_d Z_\nu$, we have $\tilde{G}^*_\nu =_d G^*_\nu$ and can conclude from (34) that

$$\sup_{x\in\mathbb{R}} |G^*_\nu(x) - G(x)| \to_{\mathbb{P}_{\theta_0}} 0.$$

So we have derived (33) and Theorem 5 is proved completely. □

REMARK 5. The result of Theorem 5 also holds for studentised versions of the parametric bootstrap.

3.3. Modelling imperfect repair

Many of the different software reliability models that have been proposed, are for technical reasons restricted by the assumptions of:

(A1) *Perfect repair:* no new faults are introduced during a repair with probability 1.
(A2) *Fixed software size:* there is no addition of new software during testing.
(A3) *Independence of faults:* faults (and hence their failure times) are independent.

As there exist no perfect testers and programmers, there will always be a positive chance of introducing new faults, while repairing an old one. Secondly, development and testing of software usually takes place simultaneously in practice. Because the addition of software, that has never been tested before, will have an effect on the reliability, it seems reasonable to take also software growth during testing into account. Furthermore certain bugs will prevent parts of the software to be inspected and therefore will hide other bugs, thus violating the assumption of independence of faults. Dropping (A3), however, would cause the mathematical problem to become highly complicated and almost untractable.

In this section we introduce a new model, the Poisson Growth and Imperfect Repair (PGIR) model. We combined the modelling of imperfect repair and software growth in a natural way. Furthermore to a certain extent the model will account for dependencies between faults. The model has attractive statistical properties, besides.

Let $\tau > 0$. We consider a test experiment as described in the introduction. Let $T_0 := 0$ and $T_i, i = 1, 2, \ldots$ the failure times of the occurring failures. Repair takes place immediately after a failure is detected. After each repair the addition of new software is allowed. Due to the correction of a fault and eventually due to the addition of new software at time T_i, there is a change in software of size $K_i, i = 0, 1, \ldots$. We assume that at time T_i apart from deleting one fault, N_i new faults are introduced. We assume that N_i is a stochastic variable, Poisson distributed with mean $\mu K_i, i = 0, 1, \ldots$. We consider the testing process during $[0, \tau]$, observing say $n(\tau)$ faults. Let

$$n(t) := \sum_{i=1}^{n(\tau)} I\{T_i < t\}, t \in [0, \tau], \tag{35}$$

the number of failures detected (faults deleted) during $[0,t]$ and let

$$N(t) := \sum_{i=0}^{n(\tau)} N_i I\{T_i < t\}, t \in [0,\tau], \tag{36}$$

the number of faults introduced during $[0,t]$. We assume that the failure intensity λ, like in the JM model, at any time is proportional to the remaining number of faults, that is:

$$\lambda(t) := \phi[N(t-) - n(t-)], t \in [0,\tau], \tag{37}$$

where ϕ denotes the constant occurrence rate per fault. With use of the data $(T_i, K_i), i = 0,1,\ldots,n(\tau)$, obtained from the experiment as described above, one can estimate the parameters (μ,ϕ) of the underlying PGIR model. We use the maximum likelihood estimation (MLE) procedure for this purpose. The following lemma will be very useful:

LEMMA 3. *For all $m \in \mathbb{N}$ and all $(a_0, a_1, \ldots a_m) \in \mathbb{R}_+^{m+1}$, we have*

$$\sum_{N_0=0}^{\infty} N_0 \frac{{a_0}^{N_0}}{N_0!} \sum_{N_1=0}^{\infty} (N_0+N_1-1)\frac{{a_1}^{N_1}}{N_1!} \ldots \sum_{N_m=0}^{\infty} (N_0+\ldots+N_m-m)\frac{{a_m}^{N_m}}{N_m!}$$

$$= a_0(a_0+a_1)\ldots(a_0+a_1+\ldots+a_m)e^{a_0+a_1+\ldots+a_m}. \tag{38}$$

PROOF. By induction. □

We now return to the derivation of the likelihood function for the PGIR model, as described by (35)–(37). AALEN (1978) showed, that the likelihood function for estimating the parameters of the intensity function of a counting process, observed on a fixed time interval $[0,\tau]$ is given by:

$$L_\tau(\mu,\phi;T_0,T_1,\ldots,T_{n(\tau)}) = \prod_{i=1}^{n(\tau)} \lambda(T_i-)\exp(-\int_0^\tau \lambda(s)ds).$$

As the N_i are independent Poisson distributed stochastic variables with mean μK_i, and defining:

$$a_i := \mu K_i e^{-\phi(\tau-T_i)}, \tag{39}$$

$$b_i := I\{N_0+\ldots+N_{i-1} = i\}, \tag{40}$$

for $i = 1,\ldots,n(\tau)$, we get for the likelihood function under the finer filtration (observing also the sizes of the software changes):

$$L_\tau(\mu,\phi;(T_i,K_i), i = 0,1,\ldots,n(\tau)) =$$

$$= \sum_{N_0=1}^{\infty} \sum_{N_1=b_1}^{\infty} \cdots \sum_{N_{n(\tau)-1=b_{n(\tau)-1}}}^{\infty} \sum_{N_{n(\tau)}=0}^{\infty} \prod_{i=0}^{n(\tau)} \left[\frac{(\mu K_i)^{N_i}}{(N_i)!} e^{-\mu K_i} \right]$$

$$\prod_{i=1}^{n(\tau)} \phi(\sum_{j=0}^{i-1} N_j - i + 1) e^{-\phi\left(\sum_{i=0}^{n(\tau)} (\tau - T_i) N_i + \sum_{i=1}^{n(\tau)} T_i - \tau n(\tau)\right)}$$

$$= \phi^{n(\tau)} \exp\big(\phi \sum_{i=1}^{n(\tau)} (\tau - T_i) - \mu \sum_{i=0}^{n(\tau)} K_i\big)$$

$$\times \sum_{N_0=0}^{\infty} N_0 \frac{{a_0}^{N_0}}{N_0!} \sum_{N_1=0}^{\infty} (N_0 + N_1 - 1) \frac{{a_1}^{N_1}}{N_1!} \cdots$$

$$\sum_{N_{n(\tau)}=0}^{\infty} (N_0 + N_1 + \ldots + N_{n(\tau)} - n(\tau)) \frac{a_{n(\tau)}^{N_{n(\tau)}}}{N_{n(\tau)}!}$$

$$= \phi^{n(\tau)} \exp\big(\phi \sum_{i=1}^{n(\tau)} (\tau - T_i) - \mu \sum_{i=0}^{n(\tau)} K_i\big) \sum_{i=0}^{n(\tau)} a_i \prod_{i=0}^{n(\tau)-1} \left[\sum_{j=0}^{i} a_j \right]$$

$$= (\mu\phi)^{n(\tau)} \exp\big(\phi \sum_{i=1}^{n(\tau)} (\tau - T_i) - \mu \sum_{i=0}^{n(\tau)} K_i (1 - e^{-\phi(\tau - T_i)})\big)$$

$$\prod_{i=0}^{n(\tau)-1} \left[\sum_{j=0}^{i} K_j e^{-\phi(\tau - T_j)} \right]. \qquad (41)$$

We note that if $N_0 + \ldots + N_{i-1} = i$, that is, if $b_i = 1$ (so we have to sum N_i from 1 to ∞), then the coefficient $(N_0 + \ldots + N_i - i)$ in the i-th sum equals zero for $N_i = 0$. So we can take all lower-bounds equal to zero and use Lemma 3 to get (41).

Taking the logarithm of the likelihood function (41) yields

$$\log L_\tau(\mu, \phi) = n(\tau) \log(\mu\phi) - \mu \sum_{i=0}^{n(\tau)} K_i \big(1 - e^{-\phi(\tau - T_i)}\big)$$

$$+\phi \sum_{i=1}^{n(\tau)} (\tau - T_i) + \sum_{i=0}^{n(\tau)-1} \log\big(\sum_{j=0}^{i} K_j e^{-\phi(\tau - T_j)}\big).$$

So we find the likelihood equations:

$$0 = \frac{\partial}{\partial \mu} \log L_\tau(\mu, \phi) = \frac{n(\tau)}{\mu} - \sum_{i=0}^{n(\tau)} K_i\big(1 - e^{-\phi(\tau - T_i)}\big), \tag{42}$$

$$0 = \frac{\partial}{\partial \phi} \log L_\tau(\mu, \phi) = \frac{n(\tau)}{\phi} - \mu \sum_{i=0}^{n(\tau)} K_i(\tau - T_i) e^{-\phi(\tau - T_i)}$$

$$+ \sum_{i=1}^{n(\tau)} (\tau - T_i) - \sum_{i=0}^{n(\tau)-1} \left[\frac{\sum_{j=0}^{i} K_j(\tau - T_j) e^{-\phi(\tau - T_j)}}{\sum_{j=0}^{i} K_j e^{-\phi(\tau - T_j)}} \right]. \tag{43}$$

Solving the system of equations (42)-(43), we get the maximum likelihood estimators $(\hat{\mu}.\hat{\phi})$, where

$$\hat{\mu} := \frac{n(\tau)}{\sum_{i=0}^{n(\tau)} K_i\big(1 - e^{-\hat{\phi}(\tau - T_i)}\big)}$$

and $\hat{\phi}$ the solution of $g(\hat{\phi}) = 0$ with

$$g(\phi) := \frac{1}{n(\tau)} \sum_{i=1}^{n(\tau)} (\tau - T_i) + \frac{1}{\phi} - \frac{\sum_{i=0}^{n(\tau)} K_i(\tau - T_i) e^{-\phi(\tau - T_i)}}{\sum_{i=0}^{n(\tau)} K_i\big(1 - e^{-\phi(\tau - T_i)}\big)}$$

$$- \frac{1}{n(\tau)} \sum_{i=0}^{n(\tau)-1} \left[\frac{\sum_{j=0}^{i} K_j(\tau - T_j) e^{-\phi(\tau - T_j)}}{\sum_{j=0}^{i} K_j e^{-\phi(\tau - T_j)}} \right].$$

Let us consider the PGIR model as given by (35)–(37). Note that we do not observe the process $N(t)$ itself. Thus defining the filtrations

$$\mathcal{F}_{t-} := \{n(s) : 0 \le s < t\},$$

$$G_t := \{n(s), N(s) : 0 \le s < t\},$$

we notice that the intensity λ given in (37) is actually λ^G, the intensity function of the counting process with respect to the filtration G_t. With use of the Innovation Theorem (see AALEN (1978) or BREMAUD (1977)), we get

$$\lambda^{\mathcal{F}}(t) := \mathbb{E}\big(\lambda^G(t) | \mathcal{F}_{t-}\big)$$

$$= \mathbb{E}\big(\phi[N(t-) - n(t-)] | \mathcal{F}Lt-\big)$$

$$= \phi\left[\mathbb{E}(N(t-)|\mathcal{F}_{t-}) - n(t-)\right]. \tag{44}$$

It is not difficult to derive (see VAN PUL (1993)) that

$$\mathbb{E}(N(t-)|\mathcal{F}_{t-}) = \mu \sum_{i=0}^{n(t-)} {K_i}^{-\phi(t - T_i)} + n(t-). \tag{45}$$

From (44) and (45) we now see that the intensity function under the filtration $\mathcal{F}_{t-}$ (only observing the counting process $n(s), 0 \leq s < t$ and the software changes $K_i, i = 0..n(t-)$) is given by

$$\lambda^{\mathcal{F}}(t) := \mu\phi \sum_{i=0}^{n(t-)} {K_i}^{-\phi(t-T_i)}. \tag{46}$$

Furthermore, we have:

THEOREM 6. *Let $\tau > 0$. We assume that the failure data are generated by the PGIR intensity function λ in (46) with true parameter value $\theta_0 := (\mu_0, \phi_0)$, satisfying $0 < \mu_0 < \infty$ and $0 < \phi_0 < \infty$. Then the ML-estimators $\hat{\mu}$ and $\hat{\phi}$ suggested above are consistent, asymptotically normal distributed and efficient.*

PROOF. The proof of Theorem 6 is tedious but routine and therefore omitted here. Choosing:

$$\Theta_0 := [\epsilon_\mu, M_\mu] \times [\epsilon_\phi, M_\phi], 0 < \epsilon_\mu < \mu_0 < M_\mu < \infty,$$

$$0 < \epsilon_\phi < \phi_0 < M_\phi < 1,$$

the desired result follows immediately from Theorem 4 by verifying the conditions (GC1)-(GC2)&(LC1)-(LC3). See VAN PUL (1993) for details. □

REMARK 6. An interesting idea seems to set all the K_i equal to some $\bar{K}$ except for $K_0 >> \bar{K}$. With parameters $N_0 := \mu K_0$ and $\bar{N} := \mu\bar{K}$ the failure intensity becomes

$$\lambda(t; \phi, N_0, \bar{N}) := N_0\phi e^{-\phi t} + \bar{N}\phi \sum_{i=1}^{n(t-)} K_i e^{-\phi(t-T_i)}.$$

In this three parameter model, $\bar{N}$, the average number of faults introduced per repair action, can be interpreted to account for dependencies between faults. Whenever hidden faults become observable because of a fault repair, this can be considered as the introduction of new faults. Finally note that for $\bar{N} = 0$ the above model reduces to the well-known model of Goel-Okumoto.

REMARK 7. The PGIR model can be seen as a a special case within a general class of regression models. In the previous section we assumed that the N_i were Poisson distributed with a parameter depending on a single software measure. Because the process of introducing new faults is so difficult to understand, it seems appropriate to use explanatory variables and apply regression analysis. We therefore suggest the following class of models given by (35)–(37) and

$$\mathbf{N_i} =_d POI(X_i),$$

$$X_i := \exp\left(\beta_1 z_i^1 + \ldots + \beta_m z_i^m\right),$$

where the $z_i^j, j = 1 \ldots m$, are the known realisations of m software measures Z^j (like e.g. size, complexity, number-of-loops) at time T_i and where the $\beta_j, j = 1 \ldots m$, denote the corresponding regression coefficients we have to estimate. Statistical methods are available to investigate whether certain explanatory variables are redundant (or not) and whether their influence is linear, via another power, or say logarithmic.

4. A SOFTWARE RELIABILITY CASE-STUDY

Nowadays software is used in all kinds of systems, that improve the quality of life. A good example of this is the increasing use of computer systems in medical imaging applications such as X-ray, MRI and ultrasound scanners. It is obvious that the software involved should be highly reliable. In this case-study we will investigate the failure behaviour of the application software for a particular system development at Philips Medical Systems (PMS) in Best (The Netherlands) during a particular phase of the testing process.

In Section 4.1 we briefly describe how software development and testing is organised at PMS and which test data were collected. In Section 4.2 we will present most of the collected data; estimations and predictions are given of some characteristics of the software. In Section 4.3 we sketch some of the practical problems encountered, we give some recommendations and concluding remarks.

4.1. Software development and testing at PMS

PMS has started the development of products containing large software packages in the late seventies. Of these products about every year an updated version is released. Each release supports new hardware, introduces new functionality, but also includes internal structural improvements. From annual project-start up to release one can roughly distinguish the following development and test phases:

(a) *development phase 1* (several months)
(b) *pi-test* (several weeks): Preliminary installation test. This is a test to get a basic part of the system operational and ready for a clinical tryout on some probe sites.
(c) *development phase 2* (several months)
(d) *alpha-test* (several weeks): This is a software test performed by the software developers. Hardware is not tested, but only used to test the software.
(e) *SIT* (a few weeks): System integration test. Software is run on the definitive hardware with the aim of testing the complete system.
(f) *beta-test* (a few weeks): During the beta-test the complete system is tested by the software management group, which is responsible for released software products, by service and manufacturing in co-operation with people from outside (application specialists and system-operators of

the instrumentation in hospitals). This test serves mostly as acceptance test. Hereafter the new product is released.

In practice there is some limited, controlled overlap between development- and testing phases. Sometimes the alpha-test and the SIT are combined. From mathematical point of view the pi-test seems to be very appropriate to predict future software behaviour. In practice, however, immediately after the pi-test a second large development phase takes place. In this case-study we will consider therefore failure behaviour of the application software (ASW) during alpha-test; this is also the phase where most of the software failures are detected and corrected. The application software (ASW) is divide into several subsegments. See Table 1. An eloc stands for an executable line of code. Note that during development a large proportion of the elocs is left unchanged. Before alpha-testing

subsegment	# modules	# kilo elocs
1	ca. 330	ca. 60
2	ca. 300	ca. 50
3	ca. 130	ca. 50
4	ca. 180	ca. 40
5	ca. 180	ca. 20
6	ca. 90	ca. 20
7	ca. 80	ca. 20
8	ca. 30	ca. 10
9	ca. 20	ca. 10
10	ca. 10	ca. 10

TABLE 1. Number of modules and kilo elocs for some subsegments of ASW.

starts, the test-leader has made a thorough alpha-test-plan. This plan states exactly which subsegment is tested by whom and when. Time is also reserved for problem solving. The software developers are assumed to prepare the test sessions for the testers by writing tests specs. Software bugs are reported via an (Internal) Problem Report ((I)PR) and gathered in a database. Moreover, testers have to fill in a detailed log-form with information such as times of start and end of test, description of test items, which of them were successful and which not, and eventually other comments concerning (mal) functioning of hardware. Twice a week newly entered PR's are inspected at a so called Software Progress Meeting and assigned to the for this bug most capable problem solver.

Both the pi-test and alpha-test under consideration lasted about 10 weeks (50 working days). During these test-phases for each occurring software bug the following test data were collected in the PR database:

(1) The problem report number that identifies the fault.

(2) The occurrence date of the problem. Note that usually in software reliability failure times are gathered. Here we are confronted with "grouped data", the total number of faults detected per day. We will discuss the mathematical consequences of this in the next section.
(3) The subsegment to which the PR is assigned.
(4) The priority of the fault: routine (RO), urgent (UR) or very urgent (VU). Stack-dumps are typically very urgent; minor display problems routine. The priority of faults may be changed as deadlines come closer. The priority field is hence used to control SW engineers and therefore is not a very objective measure.

Moreover, during the alpha-test via the log-forms record was kept of

(5) The total test effort per day in testing hours.

We had in mind to collect also other test data such as the solving time of a problem (in man-hours), the size of the change in the software (in lines of code) and an estimation of the occurrence probability of each fault. The collection of these data caused either too much practical difficulties or caused that the significance of the data would be highly doubtful, due to inevitable subjectivities. It should be mentioned that also in the data collected some subjectivity (or non-consistency) could not be excluded completely. Moreover, the data of (5) suffer from inaccuracy, due to late or incomplete submission of the log-forms. We will return to this later, when discussing the results.

4.2 Statistical inference on the test data

Before concentrating ourselves completely on the data collected during the pi- and alpha-test of the current release, we will give a global idea of the failure history from project start. After about half a year of development a first test phase, the so-called pi-test, was performed for several weeks. After another development period of several months, a second test phase, the alpha-test began. The complete failure-history from project start is visualised in Figure 1a. The first bug was found and reported near the end of development phase 1. The upper dotted line represents the total number of bugs found versus time; the other three lines correspond to the number of bugs with priority routine (RO), urgent (UR) and very urgent (VU). Time is giving in working days, that is weekends and holidays are not counted.

In Figure 1b the distribution of the ASW-bugs over those subsegments is shown. The figures above the bars represent number of faults per 1000 elocs of existing code. The relatively large number 25.1 for subsegment 8 (in comparison with other subsegments), indicating a high failure intensity, may be explained by the fact that subsegment 8 consists of almost 100% new software and only little reuse of old software. See GRADY & CASWELL (1987) for generally accepted figures of statistical prediction models on productivity and reliability.

In Figure 2 the failure behaviour of individual subsegments during alpha test is compared. The failure rate for most of the subsegments seems to remain

constant during alpha-test. This can be explained by the fact that during the alpha-test the functionality of the system is tested in a very systematic way, part by part. Hence every now and then a new subsegment is inspected. Therefore the number of detected failures will tend to grow linearly in time. The three most critical subsegments 1, 2 and 8, have almost identical failure behaviour. The same can be said about the four smallest subsegments, 6, 7, 9 and 10. The three remaining subsegments, 3, 4 and 5, subsegments of about same size and number of bugs found, show however different failure histories. The failure intensity of subsegment 5 seems to be constant (as seems the case for most of the subsegments in this project), the failure intensity of subsegment 4 is nicely decreasing, whereas the failure intensity of subsegment 3 is still growing. This suggests, that at least for this subsegment, the alpha test was too short. Indeed, in sequel tests (SIT and beta) relatively large numbers of component 3 faults were observed.

We will compare the JM model and the L model, mentioned earlier. We will try to fit both models to the PMS data of the alpha-test. As during alpha-test also testing-hours were registered we will investigate the following two approaches:

(i) Assume the test intensity is constant (time in working days).
(ii) Estimate the test intensity by the number of testing hours.

Estimation of the model parameters is –as usually– done by the method of Maximum Likelihood Estimation (MLE). For practical computations of the likelihood functions of the two models, we refer to VAN PUL (1992A, 1992B). Parameter estimations and standard deviations are given in Table 3. From Table 3 we see that when trying to fit the L model, the parameter *epsilon* turns out to be zero, indicating that the JM model is the best model within the larger Littlewood class. The differences in the estimates of N and ϕ between Table 3a and 3b come from instabilities of the numerical procedures involved and their stop-criteria. Contour curves of the likelihood function in the neighbourhood of the maximum take the form of extremely flat ellipses. The estimation of the failure intensity λ, a software characteristic which seems more relevant than just the total number of bugs N, is more stable. Covariances for the model parameters can be easily derived. See again VAN PUL (1992B). Although we were suspicious about the accuracy of the test-intensity data, it turns out out that using approach (ii) reduces both estimate of N and its variance. Standard deviations for $\hat{N}$ of the pi-test are very reasonable; those of the alpha-test quite large, showing again that the alpha-test was not the most appropriate test phase to apply software reliability theory.

In Figure 3 we give estimates and 95% confidence bands of the mean value function for the alpha-test data, using approach (i). Further research would be needed to develop a prediction band for the future course of the stochastic process n itself. The figure suggests that another test effort of 50 or 100 working days would yield about 400 or 700 new bugs, respectively. In Figure 3 also the

FIGURE 1. ASW-failures since project start: (a) Evolution in time (b) Distribution over the ASW-subsegments.

total number of bugs detected at the end of SIT and beta-test are indicated: respectively 1068 and 1523. We see that these added points are not far from the predicted values.

4.3. Problems, recommendations and concluding remarks

During the course of these case-studies we encountered several practical problems, related to collecting data in the real world of software development. These problems, which are likely to occur in practice with any software development group, in one way or the other obstructed a direct application of the developed software reliability theory. In this section we will mention the five most eye-catching ones, discuss the actions taken to reduce the negative effects they have on our software reliability results, and finally give recommendations how in our opinion the test process should be adapted in order that these problems will not appear at all in future.

(1) *The test process is hardly or not automated.* In software reliability theory we more or less take for granted that the test process is fully automated, which means that test inputs are generated, software bugs detected and

FIGURE 2. ASW-failures during alpha test: Evolution in time for the different ASW subsegments.

all kinds of test data stored automatically. In practice, however, only a few software development companies have realised this kind of testing; more often conventional testing methods are used. In many software applications testing does not lend itself to automation. Based on what a tester perceives on a screen, he will push certain buttons and hence follow a certain test-path. Automatisation is here difficult to establish. There are many consequences of this. First of all, exact failure times of the error counting process will in general not be known but only the number of bugs detected per day (grouped data). This loss of information will lead to less accurate parameter estimations. Secondly, the stream of test data will be afflicted with larger inaccuracies than otherwise would be the case. Figures for test intensity or test effort per day will, if not registered in CPU time automatically, be liable to large subjectivities of individual testers. This will occur for the data concerning priority and subsegment of the faults too, not necessarily due to incapability or unwillingness of testers, but more often due to vagueness or ambiguities in definitions (Is a bug routine, urgent or very urgent, etcetera). The best

pi	RO	UR	VU	TOTAL
1	62	20	4	86
2	65	23	6	94
3	13	3	4	20
4	33	15	2	50
5	12	3	0	15
6	12	1	0	13
7	13	0	0	13
8	73	47	7	127
9	4	1	0	5
10	40	12	5	57
TOTAL	327	125	28	480
alpha	RO	UR	VU	TOTAL
1	82	46	7	135
2	80	46	4	130
3	59	16	2	77
4	47	29	1	77
5	25	37	2	64
6	2	5	0	7
7	6	3	1	10
8	78	32	1	111
9	18	3	0	21
10	12	3	0	15
TOTAL	409	220	18	647

TABLE 2. Classification of ASW-failures during pi-test and alpha-test.

way to overcome all this simply is to automate the whole test process. If this is not possible, one should pay a lot of attention to the data collection. Make clear agreements with the testers and avoid ambiguities in definitions. Finally, motivate the testing team to co-operate and make them aware of the fact that successful application of software reliability theory fully depends on the accuracy of the data.

(2) *Testing is not random at all.* Although this problem is related to the problem of automation (random testing cannot be performed without some automation of the test process), it is not a direct consequence of it and important enough to mention it here as a second problem. As described in Section 1 we assume in software reliability theory that in some sense software testing is a random process. It is sometimes wrongly understood, that for random testing all inputs should have equal probabilities of selection (uniform distribution). This is not true. We speak of random testing, if inputs are selected in a non-deterministic way with

JM	pi	alpha (i)	alpha (ii)
$\hat{N}$	533 ± 27	3480±2082	1646±324
$\hat{\phi}$	0.0316±0.0035	0.0042±0.0028	0.0012±0.0003
$\hat{\lambda}(\tau)$	3.35	11.89	1.16

L	pi	alpha (i)	alpha (ii)
$\hat{N}$	525±26	2381±887	1454±235
$\hat{\alpha}$	0.0226±0.0024	0.0064±0.0028	0.0014±0.0003
$\hat{\epsilon}$	0	0	0
$\hat{\lambda}(\tau)$	3.20	11.12	1.09

TABLE 3. Parameter estimations and standard deviations for the JM model and the L model.

occurrence probabilities that coincide with the input distribution in the operational profile. Random testing takes place in practice only rarely. Besides an healthy aversion of management against any time-absorbing and inefficient looking methods, random testing can be difficult to put into practice, because the process cannot be automated (problem 1) or because no information about the input distribution in the operational profile is available. If one is testing by verifying the functionality of different subsegments one by one (which means that systematically from time to time new parts of the software are inspected) or if one is only using extraordinary inputs (with small occurrence probabilities), software reliability theory cannot be expected to give reasonable answers. During the alpha test at PMS this was the case. Therefore the alpha-test is not the most suitable test-period for applying software theory. More appropriate are the pi-test and perhaps the beta-test. To make software reliability theory applicable we strongly recommend to spend at least a part of the available time and personnel to random testing. As input distribution one could ultimately choose the uniform distribution if all other information about the use in the field fails. If no proper random testing is possible but the number of tests is huge, one could decide to perform the specified tests in a random order (pseudo random testing). This will overcome a part of the problem. One should have in mind that if the tested inputs are not representative of the operational profile, translating the software reliability test results to practice can be very difficult.

(3) Hardware and software are closely related. In principle, we are investigating the behaviour of the software. Sometimes, however, only after closer examination of a system failure it is found out that it was not caused by a bug in the software but by one in the hardware. Hardware

FIGURE 3. Estimates and 95% confidence bands for the mean value function of the Jelinski-Moranda and Littlewood models for the alpha-test data using approach (i).

used during testing is often not exactly identical to the hardware used in the field. We do not want to take these hardware bugs into account. Another consequence of this strong relationship between software and hardware is that software test plans sometimes have to be changed because of hardware problems. This may lead to a decrease in test intensity (or better test efficiency), which is hard to measure precisely. It can also lead to a conflict with respect to the randomness of testing. Certain parts of the software are temporarily not inspected because of bugs in the hardware that interacts with this software. One should hence count only real software bugs and keep good track of when hardware is misbehaving. Try to quantify the test intensity (in testing hours per day) as accurate as possible.

(4) Minor faults show an occurrence behaviour which is completely different from that of major ones. The number of minor (aesthetical) faults detected tends to grow linearly in time, as occurrence of those kind of faults heavily depends on the available time to analyse and solve them.

That is, the intensity of minor faults will be strongly influenced by the intensity of major faults, but not vice versa. Applying software reliability theory will therefore only make sense for major problems.

(5) *Available time, computer facilities and personnel are limited.* Finally, in practical applications everything turns on money: deadlines have to be kept, budgets have to be observed. This premise induces software managers to take actions and decisions which are in contradiction with the original software reliability theory assumptions. Such decisions are: (i) due to delays and deadlines, development and testing of parts of the software takes place simultaneously, (ii) alpha-test starts with too many bugs in the software, (iii) priorities of bugs are changed to speed up their solution. The second point is illustrated by the large number of bugs detected during alpha-test at PMS. This causes an enormous overhead (i.e. writing problem reports, updating database, weekly evaluation of new bugs, writing problem report answers, etcetera). The number of bugs found each day is therefore limited by the available manpower for testing and not by the quality of the software. Hence, in our opinion thorough testing takes place too late in the development phase to let the alpha-test be appropriate for application of the software reliability theory.

We have only mentioned five of the main problems to illustrate the complications arising when applying software reliability theory in practical situations. As stated earlier these problems are not typical for the software development at Philips, but are due to the complex process of software development itself, and to the many human factors involved.

Finally, we summarise our conclusions. The software reliability models and methods investigated so far can be applied to the data available at PMS, but the results are not very promising. The small number of field problems indicate that the released software is highly reliable. Because of the nature of the testing process and the quality of the test-data, it is not possible at this stage to use the results to provide the customer with strong reliability guarantees. The test process environment should be adapted (see the recommendations in the beginning of this section) in order to apply software reliability theory with more success. Of course there are also other (more simple, static) approaches to software reliability. We think of static models, Humphrey's regression method (Humphrey & Polson (1992)), etcetera. Of course, also software reliability models and methods should be further improved. We are, as Grady & Caswell (1987), strongly convinced that a thorough quantitative analysis of the software and of the software test process will lead to an increase of both efficiency and quality.

References

1. O.O. Aalen (1978), Non-parametric inference for a family of counting processes, *Annals of Statistics* **6,** pp. 701–726.

2. F. Akiyama (1971), An example of software system debugging, *Information Processing 71,* North-Holland, New-York, pp. 353–359.
3. P.K. Andersen, O. Borgan (1985), Counting processes models for life history data: a review, *Scandinavian Journal of Statististics* **12,** pp. 97–158.
4. P.K. Andersen, O. Borgan, R.D. Gill, N. Keiding (1993), *Statistical Methods Based on Counting Processes,* Springer-Verlag, New York.
5. H.E. Ascher, H. Feingold (1982), Repairable systems reliability: future research topics, *Reliability in Electrical and Electronic Components and Systems,* North-Holland, Amsterdam, pp. 81–88.
6. L. Barendregt, M.C. van Pul (1994), Consistency of maximum likelihood and other estimators for the parameters of the Littlewood model in software reliability. To appear in *Statistica Neerlandica, early 1995.*
7. R.E. Barlow, F. Proschan (1975), *Statistical Theory of Reliability and Life Testing Probability Models,* Holt, Rinehart and Winston, New York.
8. S.L. Basin (1973), *Estimation of Software Error Rates via Capture-Recapture Sampling: A Critical Review,* Science Applications Report, Paolo Alto, CA.
9. P.J. Bickel, D. Freedman (1981), Some asymptotic theory for the bootstrap, *Annals of Statistics* **9,** pp. 1196–1217.
10. p. Billingsley (1968), *Convergence of Probability Measures,* Wiley, New York.
11. O. Borgan (1984), Maximum likelihood estimation in parametric counting process models, with applications to censored failure time data, *Scandinavian Journal of Statistics* **1,** pp. 1–16. (corrigendum, *ibid.,* 275.)
12. P. Bremaud (1977), Processes ponctuals et martingales: Résultats récents sur la modélisation et filtrage, *Advanced Applied Probability* **9,** pp. 362–416.
13. B. Carnahan, J.O. Wilkes (1973), *Digital Computing and Numerical Methods,* Wiley, New York.
14. H. Cramér (1946), *Mathematical Methods of Statistics,* Princeton University Press, Princeton.
15. A.R. Feuer, E.B. Fowlkes (1979), Some results from an empirical study of computer software, *Proceedings Fourth International Conference on Software Engineering,* Munich, Germany, pp. 351–355.
16. R. Fletcher, C.M. Reeves (1964), Function minimisation by conjugate gradients, *Computer Journal* **7,** pp. 149–154.
17. T. Gilb (1979), *Software Metrics,* Winthrop, Cambridge, p. 28.
18. R.D. Gill, S. Johansen (1990), A survey of product-integration with a view towards application in survival analysis, *Annals of Statistics* **18,** pp. 1501–1555.
19. A.L. Goel, Okumoto, K. (1979), Time dependent error-detection rate model for software reliability and other performance measures, $\mathbb{E}EE$ *Transactions on Reliability* **28,** pp. 206–211.
20. R.B. Grady, D.L. Caswell (1987), *Software Metrics: Establishing a*

Company-Wide Program, Prentice-Hall, Englewood Cliffs, NJ.
21. M.H. HALSTEAD (1977), *Elements of Software Science,* Elsevier, New York.
22. I.S. HELLAND (1982), Central limit theorems for martingales with discrete or continuous time, *Scandinavian Journal of Statistics* **9,** pp. 79–94.
23. R. HELMERS (1991), On the edgeworth expansion and the bootstrap approximation for a studentised U-statistic, *Annals of Statistics* **19,** pp. 470–484.
24. W.S. HUMPHREY, N. POLSON (1992), *Projecting Software Defaults,* Unpublished paper.
25. I.A. IBRAGIMOV, R.Z. KHAS'MINSKII (1979), *Statistical Estimation - Asymptotic Theory,* Springer-Verlag, New York.
26. J. JACOD, A.N. SHIRYAEV (1987), *Limit Theorems for Stochastic Processes,* Springer-Verlag, Berlin.
27. Z. JELINSKI, P. MORANDA (1972), Software reliability research, *Statistical Computer Performance Evaluation,* pp. 465–484.
28. T.G. KURTZ (1983), Gaussian approximations for Markov chains and counting processes, *Bulletin of the International Statistical Institute* **50,** pp. 361–375.
29. J.F. LAWLESS (1982), *Statistical Models and Methods for Lifetime Data,* Wiley, New York.
30. E. LENGLART (1977), R'elation de domination entre deux proc'esses, *Annals Institute Henri Poincar'e* **13,** pp. 171–179.
31. B. LITTLEWOOD (1980), Theories of software reliability: How good are they and how can they be improved? *IEEE Transactions on Software Engineering* **6,** pp. 489–500.
32. D.K. LLOYD, M. LIPOW (1977), *Reliability: Management, Methods, and Mathematics,* 2nd ed., published by the authors, Redondo Beach, CA.
33. N.R. MANN, R.E. SCHAFER, N.D. SINGPURWALLAH (1974), *Methods for Statistical Analysis of Reliability and Life Data,* Wiley, New York.
34. T.J. MCCABE (1976), A complexity measure, *IEEE Transactions on Software Engineering* **2,** pp. 308–320.
35. H.D. MILLS (1972), *On the Statistical Validation of Computer Programs,* IBM Federal Systems Division, Report FSC-72-6015, Gaithersburg, MD.
36. G. MOEK (1983), *Software Reliability Models on Trial: Selection, Improved Estimation, and Practical Results,* Report MP 83059 U, National Aerospace Laboratory NLR, Amsterdam.
37. G. MOEK (1984), Comparison of some software reliability models for simulated and real failure data, *International Journal of Modelling & Simulation* **4,** pp. 29–41.
38. R.W. MOTLEY, W.D. BROOKS (1977), *Statistical Prediction of Programming Errors,* Rome Air Development Center, Technical Report RADC-TR-77-175, Rome, NY.
39. J.D. MUSA (1975), A theory of software reliability and its application, *IEEE Transactions on Software Engineering* **3,** pp. 312–327.

40. J.D. Musa, A. Iannino, K. Okumoto (1987), *Software Reliability: Measurement, Prediction, Application,* McGraw-Hill Book Company, New York.
41. P.M. Nagel, J.A. Skrivan (1982), *Software Reliability: Repetitive Run Experimentation and Modelling,* Boeing Computer Service Company, BCS-40366, Seattle, WA.
42. P.M. Nagel, F.W. Scholz, J.A. Skrivan (1984), *Software Reliability: Additional Investigations into Modelling with Replicated Experiments,* Boeing Computer Service Company, BCS-40446, Seattle, WA.
43. J.A. Nelder, R. Mead (1965), A simplex method for function minimisation, *Computer Journal* **7,** pp. 308–313.
44. D. Pollard (1989), *Convergence of Stochastic Processes,* Springer Series in Statistics, New York.
45. M.C. van Pul (1991), *Modelling Imperfect Repair and Software Growth,* Proceedings, Interface '91, Seattle.
46. M.C. van Pul (1992a), Asymptotic properties of statistical models in software reliability, *Scandinavian Journal of Statistics* **19,** pp. 235–253.
47. M.C. van Pul (1992b), Simulations on the Jelinski-Moranda model of software reliability; application of some parametric bootstrap methods, *Statistics and Computing* **2,** pp. 121–136.
48. M.C. van Pul (1993), *Statistical Analysis of Software Reliability Models* (PhD-Thesis, University of Utrecht), CWI Tracts **95,** Centre for Mathematics and Computer Science, Amsterdam.
49. C.R. Rao (1973), *Linear Statistical Interference and its Applications,* Wiley, New York.
50. R. Rebolledo (1980), Central limit theorems for local martingales, *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* **51,** pp. 269–286.
51. P. Rook (1990), *Software Reliability Handbook,* Elsevier Applied Science, London.
52. B. Rudner (1977), *Seeding/Tagging Estimation of Software Errors: Models and Estimates,* Rome Air Development Center, Technical Report RADC-TR-77-15, Rome, NY.
53. M.L. Shooman (1968), *Probabilistic Reliability: An Engineering Approach,* McGraw-Hill, New York.
54. K. (1981), On asymptotic accuracy of Efron's bootstrap, *Annals of Statistics* **9,** pp. 1187–1195.
55. T. Sunohara, A. Takano, K. Uehara, T. Ohkawa (1981), Program complexity measure for software development management, *Proceedings Fifth International Conference on Software Engineering,* San Diego, pp. 100–106.
56. T.A. Thayer (1976), *Software Reliability Study,* Rome Air Development Center, Technical Report RADC-TR-76-238, Rome, NY. Seattle, WA, pp. 106–109.
57. A.W. Vaart van der (1987), *Statistical Estimation in Large Parame-*

ter Spaces, Mathematical Centre Tracts 44, Centre for Mathematics and Computer Science, Amsterdam.