Quality Engineering, 19:339–351, 2007

ISSN: 0898-2112 print/1532-4222 online
DOI: 10.1080/08982110701489231

Taylor & Francis
Taylor & Francis Group

Two Case Studies in Applying Statistical Models in Software Development

Ed Brandt and Rob Henzen
Refis System Reliability Engineering, Bilthoven, The Netherlands

Isaac Corro Ramos and Alessandro Di Bucchianico
Department of Mathematics, Eindhoven University of Technology, Eindhoven, The Netherlands

ABSTRACT We present two case studies in software testing to illustrate two different goals of software reliability analyses. The first case study is about administrative software of an insurance company, while the second one is on an issue of national interest in the Netherlands: safety of software control of a closable dam as part of a sea flood protection system. We pay attention to practical and methodological aspects that often are ignored or not correctly treated in software reliability analyses. In particular, we provide pointers to recent statistics literature that are relevant for software reliability.

KEYWORDS goodness-of-fit test, GOS models, ML estimate, model selection matrix, NHPP models, software reliability, software testing, trend test

1. INTRODUCTION

Test activities play an important role in developing and maintaining information systems. Although the way these test activities are being performed has changed dramatically through the past decades (for instance as a result of the introduction of structured testing methods like TMap® and ISEB), the final goal of testing has more or less remained the same over the years. In general two elements can be distinguished. The first element is that one tries to establish to what extent the system under test meets the required quality demands. In this case testing is thus meant to *measure quality*. Here test results are compared to the required level of quality of the system, expressed in acceptance criteria and/or requirements.

The second element consists of detecting and solving defects in the system. Here testing is used to *improve quality*. If more defects are found and resolved, then fewer defects will remain in the system, which results in a more reliable system.

Both aspects of testing make use of the defects found during test. In reporting the final results of testing the number of executed test cases, detected errors and resolved defects are key performance indicators.

In general however nothing is said about the number of defects that were *not* found during test. These defects however are equally if not more important than the defects that were found. The reason for this being that the evaluation of the product risk at implementing or shipping a software

Address correspondence to A. Di Bucchianico, Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600MB Eindoven, The Netherlands. E-mail: a.d.bucchianico@tue.nl

product not so much depends on the number of defects found during test, but mainly on the number of defects that still remain in the product.

This article is organized as follows. In Section 2 we discuss statistical models for software reliability. Our discussion includes both general modeling issues and specific statistical issues such as using model selection, appropriate goodness-of-fit tests and numerical issues when computing estimates. In Sections 3 and 4 we describe the insurance company case study and the removable dam case study, respectively. We show both our original analyses and a re-assessment in view of new developments in statistics. Section 5 contains general conclusions from this article.

2. STATISTICAL MODELS FOR SOFTWARE RELIABILITY

At first sight it seems unlikely to give a reliable estimate of the number of defects remaining in a system after test. However, practical experience since the early 1970's shows that the rates at which defects are detected follow certain patterns (see e.g., Ohba (1984)). Together with the key assumption that the test cases are representative for the use of the software system in production, statistics can be used to produce estimates of software reliability.

In the past decades several statistical models have been developed to estimate the remaining number of defects in a system using defect information generated during testing. These models originally were developed and applied in the field of hardware reliability and the testing of EPROMS. During the years the existing models were adjusted to make them suitable for use in other fields like (administrative) software information systems. The monographs Lyu (1996), Musa (2006), and Pham (2006) are rich sources of information on this subject.

Although all models differ in their respective conditions concerning the development and test process, and the process of defect resolving, they are all based on the same principle: all models assume the reliability[1] of a system to increase when defects are found and resolved. In mathematical reliability models, one tries to model this reliability growth. Once a suitable model is found, the corresponding reliability curves can be used to extrapolate the results to the (nearby) future, thus enabling us to predict the growth in reliability of the system when testing is continued and to estimate the remaining defects in the system.

2.1. Notation and General Background

We first establish some notation. We adopt the general notation that N is the initial number of defects in the software and that $T_1,\ T_2, \ldots T_n$ are defect detection times (failure times), so $T_1,\ T_2 - T_1, \ldots$ $T_n - T_{n\text{-}1}$ are times between detection of defects. In this case we speak of exact data or ungrouped data. Defect reports often only contain weekly summaries of defects. The observations are then defect counts m_i in intervals (T_{i-1}, T_i) (and hence, $\sum_{i=1}^{n} m_i$ is the total number of defects). In such cases we speak of grouped data or interval data. It is often convenient to define $T_0 = 0$ (often implicitly assumed in many articles), so that the times between errors can be written in terms of the cumulative times as $X_1 = T_1 - T_0, X_2 = T_2 - T_1, \ldots, X_n = T_n - T_{n-1}$. Models arise when one attaches probability distributions to these quantities. It is important to note that both $T_1, T_2, \ldots, T_n$ and $X_1, X_2, \ldots, X_n$ violate the assumptions of a random sample. In other words, the random variables involved are usually not identically distributed and may not be independent.

If we assume perfect immediate repair of defects (which is justified if we take testing effort rather than calendar time as time), then like in hardware reliability systems with minimal repair we must interpret the times $T_1, T_2, \ldots, T_n$ as event times of a point process. A point process $(N(t))_{t\geq 0}$ is a collection of random variables with non-negative integer values such that $N(t)$ is the number of events up to time t, so $N(t) = \#\{i \mid T_i \leq t\}$. A usual assumption is that at most one defect is detected at a time point. The case of failure counts as discussed above perfectly fits in if we distinguish between (possibly hidden) detection events described by $(N(t))_{t\geq 0}$ and the available observations (which may only be available in summarized form through failure counts). The distinction between time-between-failure models and failure count models as is sometimes used in some literature is thus confusing and not necessary. Of course, for a

[1]Reliability is defined as the probability that an item will perform a required function without failure under stated conditions for a stated period of time.

given model it is important to distinguish between time-between-failure data and failure count data, because they have different likelihoods and do not carry the same amount of information.

It is important to note that the concept of failure rate in software reliability models is the failure rate of the associated point process $(N(t))_{t\geq 0}$ rather than the failure rate concept of a random variable as is used in hardware reliability in the case of non-repairable systems (see Thompson (1981, 1988) for details).

In order to define explicit models, one can either model the joint distribution of $X_1, X_2, \ldots, X_n$ or the joint distribution of $(N(t))_{t\geq 0}$. In the software reliability literature these are often called type I and type II models respectively, after Singpurwalla and Wilson (1994).

2.2. GOS (General Order Statistics) Models

The basic idea behind this class is to assume a common distribution for the time to detect faults and to interpret the observation times $T_1, T_2, \ldots, T_n$ as the order statistics of a sample, i.e., as the observations ordered from small to large. The well-known Jelinski-Moranda model is the special case of a sample from an exponential distribution, while the Shick-Wolverton model occurs if we take the Weibull distribution. As far as parameter estimation is concerned, it is important to realize that apart from the parameter(s) from the underlying distribution there is an additional parameter N, the initial number of defects. This makes parameter estimation non-trivial from a numerical point of view (see e.g., Joe and Reid (1985)). In order to obtain confidence intervals for parameters, asymptotic normality of ML estimators in terms of Fisher information is often suggested in the software reliability literature. This is not correct as the regularity conditions for asymptotic normality are violated because N is also a parameter. We refer to Joe (1989) for details and a (complicated) correct way of obtaining asymptotic confidence intervals. For a Bayesian analysis, we refer to Raftery (1987).

2.3. NHPP (Nonhomogeneous Poisson Process) Models

A popular class of models arises when one chooses Poisson distributions to model the number of defects observed in intervals through the formula $P(N(t) - N(s) = k) = e^{-(\Lambda(t)-\Lambda(s))}(\Lambda(t) - \Lambda(s))^k/k!$ with the additional assumption of independence of number of defects in disjoint time intervals. The function $\Lambda(t)$ denotes the expected cumulative number of defects at time t. The choice $\Lambda(t) = a(1 - e^{-bt})$ leads to the Goel-Okumoto model, while the choice $\Lambda(t) = a(1 - (1 + bt)e^{-bt})$ leads to the S-shaped Yamada model (there are more S-shaped models in this class, using a slightly different form of $\Lambda(t)$ or by adding an additional parameter). The Musa basic execution time model and the Schneidewind models are basically the same as the Goel-Okumoto model, because they only differ in the way the time t is interpreted and whether all observed times are used to estimate model parameters. A relation with the GOS models was given by Langberg and Singpurwalla (1985), who showed that if one treats N in GOS models as a hyperparameter (i.e., attach a probability distribution to it), then one obtains well-known models in the NHPP class like the Goel-Okumoto model. Parameters for NHPP models are usually estimated by maximizing the likelihood equations which can be found in several text books a personal favorite is Rigdon and Basu (2000). Although ML estimators possess optimal asymptotic properties in general (minimal variance unbiased estimators), one should carefully consider whether to investigate the original parameters of the models involved or study derived parameters like reliability and failure intensity. For the Goel-Okumoto model, Jeske and Pham (2001) show that the original ML estimators of the parameters are inconsistent but the ML estimator of the failure intensity is consistent.

Note that the likelihood equations may not possess solutions (see Yin and Trivedi (1999) for a graphical illustration). Simple algebraic conditions for existence of ML parameter estimates are available for both ungrouped data (Hossain and Dahiya, 1993; Knafl and Morgan, 1996) and grouped data (Knafl, 1992; Hossain and Dahiya, 1993; Rongguan and Heliang, 2002). It is often ignored that direct optimization of the likelihood function is numerically troublesome because of the different magnitudes of the parameters involved (see Yin and Trivedi, 1999 for examples) and because likelihoods may be extremely small numbers. The estimation procedures described in Knafl (1992) and Knafl and Morgan (1996) reduce the optimization procedures to simple root finding of monotone functions and

thus are to be preferred. Modern techniques such as MCMC (Markov Chain Monte Carlo) to perform Bayesian analyses for NHPP models are treated in detail in Kuo and Yang (1996).

In order to obtain confidence intervals for parameters, asymptotic normality of ML estimators in terms of Fisher information is often used in the software reliability literature. As for GOS models discussed in the previous paragraph, this is not correct. We again refer to Joe (1989) for details and a (complicated) correct way of obtaining asymptotic confidence intervals.

2.4. Model Selection

Before one tries to fit any reliability growth model, one should determine whether the data indicate reliability growth. This may be accomplished using appropriate plots or more formally with trend tests. A simple plot is plotting T_i/i for ungrouped data (these are the running averages of the time-between-errors) or running averages of m_i/X_i in case of grouped data (using the notation of Subsection 2.1). Popular choices for trend tests include the Laplace and the MIL-HDBK 189 tests. Both tests translate the null hypothesis of no reliability growth as data coming from an ordinary, homogeneous Poisson process. Within this context, the Laplace test is known to be optimal against the NHPP model with $\Lambda(t) = b^{-1}\exp(a + bt)$ while the MIL-HDBK 189 test is optimal against the power law NHPP model with $\Lambda(t) = (t/\theta)^{\beta}$. For a comparison of the performance of these and other tests, we refer to Bain et al. (1985) and Cohen and Sackrowitz (1993).

Unlike in hardware reliability, there are no basic scientific laws that lead to models. The well-known statement of George Box 'All models are wrong, but some are useful,' is very much to the point in software reliability. However, one can find lists of assumptions and data requirements for software reliability models in the literature (see e.g., Ohba (1984), Goel (1985), Chapter 3 of Lyu (1996), Chapter 4 of Xie et al. (2004) and Chapter 6 of Pham (2006)). Sometimes assumptions can be translated into differential equations, the solution of which yields a specific model. Unfortunately these lists of assumptions suffer from several drawbacks. For example, there is no universal agreement in the literature on the list of assumptions for certain well-known models, assumptions are ill-defined or may be hard to verify in practice. Moreover, some assumptions or data requirements are not valid. An example is the data requirement mentioned in some papers that certain GOS or NHPP models only work for failure count data or only for ungrouped data. This is simply false since likelihood equations exist for all GOS and NHPP models for both types of data.

Systematic approaches to use model assumptions and data requirements for initial model selection have not received much attention in the literature, Kharchenko et al. (2002) being an exception. Therefore we have been developing a matrix-based procedure to support the choice of the models to work with. A simple version of this matrix can be found in Table 1, containing only a small selection of the existing models. Since we wish to select applicable models rather than rule out non-applicable, we avoided restrictive requirements in our matrix by using negated forms of restrictions. As said in the introduction of Section 2 a key assumption is that the test cases are representative for the use of the software system. However, there are attempts to overcome this assumption (see e.g., Zhang et al. (2002)). The same holds true for the perfect repair assumptions (see Xie et al. (2007)).

To select models, one first has to select relevant assumptions and weights to incorporate the available information on the testing project at hand. If all requirements and selected assumptions of a model are satisfied, then the score for this model is 100%. In all other cases the score of the model is defined using the relative importance (weight) of the applicable characteristics of the model. The weights in our matrix are subjective choices; further research is needed to obtain objective criteria. We stress, however, that the selection matrix is a soft tool providing help to practitioners that do not know the assumptions behind the many existing models.

After initial model selection and parameter estimation have been performed, final model selection should be done through model validity checks. For this one should use appropriate plots like the u-plot and TTT (Total Time on Test) plots as well as formal goodness-of-fit tests. As argued in Subsection 2.1, software reliability models typically do not yield independent and identically distributed observations. So the widespread practice of applying standard goodness-of-fit tests like the Kolmogorov test

TABLE 1 **Selection Matrix: Columns Indicate Models. An 'x' Indicates Whether a Data Requirement or Assumption is Allowed**

	Weight	Geometric	Jelinski-Moranda	Littlewood-Verrall	Musa-Okumoto	Goel-Okumoto	Shick-Wolverton	Schneidewind	Yamada S-shaped	Duane (power law)
Data Requirements and Assumptions										
Data may be exact failure times (ungrouped data)	2	x	x	x	x	x	x	x	x	x
Data may be grouped failure times (interval count data)	2	x	x	x	x	x	x	x	x	x
Testing intervals may be of different length	3	x	x	x	x	x	x	x	x	x
Failures need not occur equally likely	2	x		x	x	x		x	x	x
Detection of defects may be dependent of each other	2			x	x	x		x	x	x
Failures need not be of the same severity	1			x	x	x		x	x	x
Detection rate depends on time (testing effort)	3			x	x	x		x	x	x
Detection rate depends on number of remaining defects	3	x	x				x			
Failures need not be repaired instantaneously	3		x			x	x	x	x	x
Imperfect repair of defects allowed	2									
Infinite number of defects allowed	2				x					x

to times between failures is inappropriate in the case of NHPP models. An additional step is needed to transform the observations (like in the case of u-plots) or use that conditional on the last observation, the other observations are distributed as the order statistics of a sample from a distribution proportional to the cumulative intensity Λ (see e.g., Rigdon and Basu (2000)). New unconditional goodness-of-fit tests that have more power than traditional conditional tests are becoming available for several subclasses of NHPP models, see e.g. Bhattacharjee et al. (2004) and Zhao and Wang (2005). However, results for grouped data are not available.

3. CASE STUDY 1: ADMINISTRATIVE SOFTWARE AT AN INSURANCE COMPANY

As an example of the usefulness of measuring software reliability (in addition to other metrics) we now discuss a case in which a major insurance company in the Netherlands encountered serious difficulties concerning the duration of acceptance testing of one of its administrative systems. In particular, the time needed to test a new release of the system tended to increase with every release and the span of time needed for the acceptance test almost exceeded the available time between two system releases. Needless to say this situation presented a serious problem as the timely implementation of necessary system enhancements was threatened. In this particular case the main question to be answered was whether (and if so: how) the test process could be shortened with none or minimal effect on the quality of the test and thus on the software delivered after test.

3.1. Context

The software was built by an external vendor, runs on an AS-400 platform and consists of a batch and an online part. Several hundreds of concurrent users depend on this system on a daily basis. The system is kept up to date by releasing a new version of the software every two months. Testing is performed according to the standard way of working at the insurance company and consists of a Functional Acceptance Test (FAT) and a User Acceptance Test (UAT). In both tests the adopted test strategy demanded not only test cases to be executed for new or adjusted parts of the system, but also for the parts that where not modified at all, i.e., an extensive regression test is always part of both the FAT and the UAT.

3.2. Approach

In order to get insight in the quality of the software after FAT and UAT it was suggested to perform a reliability analysis on the software based on the test results of one or (preferably) more releases. In choosing suitable releases one of the difficulties encountered was the partial unavailability and/or incompleteness of test results, and a lack of information about the effort made in testing. Finally it was decided to perform the analysis only for the FAT-part of a late 2003 release, because only for this release nearly all necessary information was available at the time. For this test a total number of 493 defects was recorded, divided into eight different categories (see Table 2).

For obvious reasons the defects in the categories 3, 6, 7 and N/A were not taken into account in the reliability analysis. As for the available information about the testing effort, the data recorded gave rise to some confusion. It appeared that two different records had been kept, both representing testing hours. One record contained tested hours per tester per week, the other contained tested hours per tester per day. However, although both records covered the same period in time, comparing the total amount of testing hours in both records revealed a difference of 90 hours. Unfortunately no explanation could be found for this difference. Moreover, at some occasions no hours were written in the daily log for a

TABLE 2 Defect Categories of FAT Test

Category	Source description	# Defects
1	Defect in new software	229
2	Design defect	20
3	Test Environment defect (hardware, configuration etc)	26
4	New defect in existing software	12
5	Existing defect in existing software	18
6	Erroneously recorded as a defect	100
7	Already recorded	53
8	Unknown	1
N/A	Category not recorded	31

TABLE 3 **Failures Counts and Testing Hours of FAT Test**

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

certain day, while defects were registered in the defect log on the very same day. To overcome this problem an extra assumption had to be made about the testing hours for these days based on the average number of testing hours per day over the entire testing period, the result of which is Table 3.

Graphical inspection of running averages of reported defects per test hour showed a slight reliability growth after week 1. Since the test intervals had different lengths, standard trend tests could not be applied. Finally suitable reliability models had to be selected. At the time our original analysis took place the selection matrix in Table 1 was not yet available and our ad-hoc selection method yielded the Goel-Okumoto model only.

After re-analysis with our matrix, we decided to add the Yamada S-shaped model.

3.3. Results

The original analysis was performed using the Casre software, developed by Allen Nikora of the Jet Propulsion Laboratory. In spite of the extensive manual, many details concerning implementation of procedures are not available. Moreover, Casre inhibits model estimation if certain strict rules concerning trends are not met. Therefore for our re-analysis we also used a package of our own written in R (free open source statistical software that is rapidly becoming the standard in the statistics community). After doing a check on the existence of ML estimates, we computed parameter estimates and plotted fitted models (see Figure 1). It is clear that the Goel-Okumoto model predicts many more remaining errors than the Yamada S-shaped model.

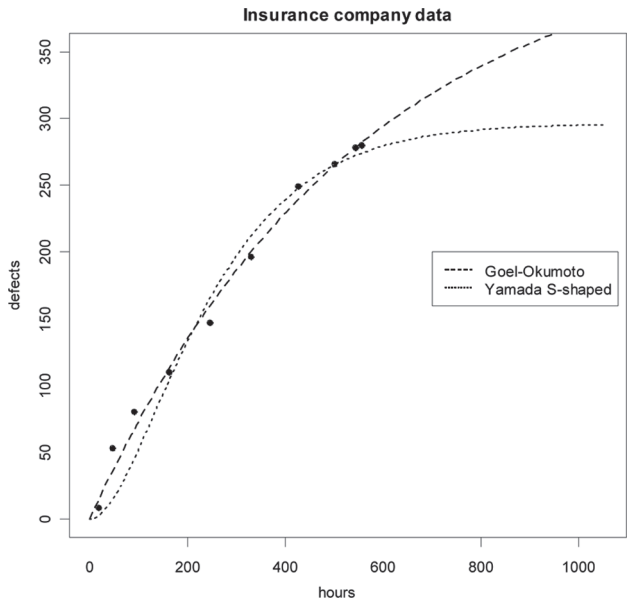

FIGURE 1 **Fitted models for insurance data from Table 3.**

Goodness-of-fit tests are hard to use here because of the limited number of observations. Assuming that the Yamada S-shaped model fits and extrapolates well and that testing is continued for a period of ten more weeks, with testing hours per week on average equal to the testing hours per week in the past period, we expect to find approximately twenty more defects. We also expect that the number of defects found per testing hour will drop rapidly and thus further testing becomes less and less efficient. However, these conclusions would change considerably if we use the Goel-Okumoto model.

3.4. Other Metrics

Let us now go back to our initial question about the feasibility of reducing the duration of the test process with none or minimal effect on the quality of the test. Obviously we did not answer this question by the reliability analysis described in the previous subsection. What we do have however, is a benchmark which can be used to check whether the requirement of no loss in software quality (or reliability) is met after measures have been taken to shorten the test process. In order to suggest suitable accommodations of the test process a few other metrics were used, two of which we will now briefly discuss.

In order to gain insight into the efficiency of the test process data were gathered concerning the rate

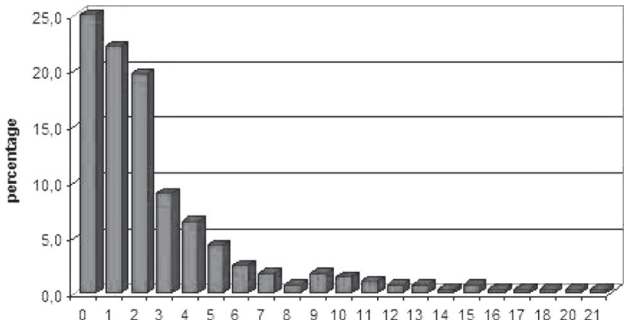

FIGURE 2 **Percentage of defects submitted for retest as function of days between submission and detection.**

of defect removal. Relevant data for this metric are the dates on which defects are found, the dates on which defects are resolved and released for retest, and finally the dates on which the defects are reported as solved (after retest). Fortunately all three of these dates were available for all the defects reported during the FAT. A Pareto-type analysis of these data is summarized in Figures 2 and 3, which show a rapid debugging process. Although these figures do not reveal the percentage of new defects being introduced, further detailed investigation proved that no direct actions were required to improve the process of defect resolving and retesting.

Finally a closer look was taken at the effectiveness of the defect detection process. To this end the defect removal efficiency (DRE) was calculated. The DRE is defined as the ratio between the number of defects found in a certain test phase and the number of defects detected in the successive phase (which can be production).

The number of defects detected in the FAT and UAT are 293 and 250, respectively. Hence, $\text{DRE} = 293/(293+250) * 100\% = 54\%$. Compared to the standard value of 80% commonly used in practice (see e.g., Musa (2006)), this value is rather low. For reasons of comparison the value of the DRE for several other releases were also calculated (see Table 4).

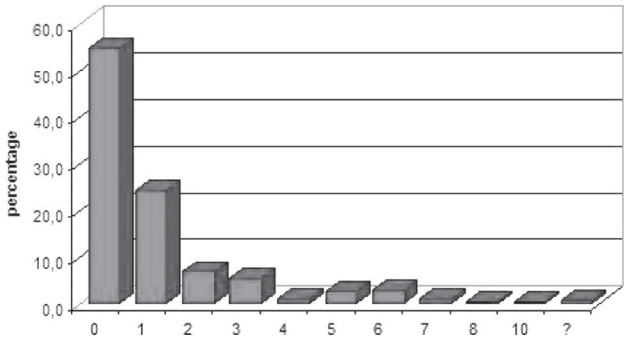

FIGURE 3 **Percentage of defects reported as solved as function of repair time in days.**

TABLE 4 **Defect Removal Efficiency for Several System Releases.**

Release	# Defects FAT	# Defects UAT	DRE (%)
X0	293	250	54
X1	56	316	15
X2	64	101	39
X3	18	4	82
X4	77	157	33
X5	115	68	63

It turns out that the low DRE value found in the release we investigated is no coincidence. Further inspection of the way the FAT and the UAT were performed learned that this partially resulted from the fact that many test scripts and test cases were used in both the FAT and the UAT (or similar tests were performed). The only difference was that different aspects of the expected output were to be examined. This obviously leads to a lot of unnecessary extra work.

3.5. Conclusions of Case Study 1

Going back once again to the original question about the duration of the test process, the obvious conclusion is of course that a considerable profit can be made in terms of shortening this duration by either combining the FAT and the UAT, or checking all relevant output on all relevant aspects in the FAT. Furthermore no accommodation of the defect removal and retest process is necessary.

Finally the reliability analysis performed shows that continuing testing in the FAT will certainly produce more defects, but this is a relatively small portion of the defects already found. Thus no more testing is done than needed, but at the same time, the testing period as used is justified, because otherwise one would have missed too many defects.

4. CASE STUDY 2: A CLOSABLE DAM OPERATING SYSTEM

In the Netherlands, the Directorate for Public Works and Water management (DPWW) is responsible for realizing and maintaining a wide variety of constructions like bridges, dams, dikes and sluices.

As is the case in many other fields in engineering the reliability of these constructions more and more depends on the software employed. Over the years DPWW has gathered extensive experience in measuring and calculating the reliability of the hardware involved. However, knowledge about evaluating software reliability is far less developed within the directorate.

The analysis in this section was mainly conducted to investigate whether a reliability analysis on operating software for objects like those mentioned above is at all possible with regard to the current way of building, testing, and recording test data of process software, and to evaluate the extent to which the results of such analyses would be able to satisfy the need of DPWW to gain knowledge about software reliability of the aforementioned constructions. DPWW therefore provided a case with all available relevant information about a specific construction project which was considered to be representative for the objects managed by DPWW.

4.1. Context

The system to be evaluated consisted of the operating software of a large closable dam built and maintained under responsibility of DPWW. The software was built by a vendor on assignment from the main contractor of DPWW. Test activities consisted of two separate tests. The Factory Acceptance Test (FAT) took place at the location of the vendor and was conducted by the vendor under supervision of the main contractor. The Site Acceptance Test (SAT) was conducted by the main contractor and supervised by several delegates of DPWW. As suggested by the name the SAT took place at the object itself. The analysis of the software is based on the datasets generated by one FAT execution and two executions of the SAT (being initial test and retest). The two executions of the SAT had the same testing goals (opening and closing of the dam), but due to different environmental conditions (wind speed, water height, etc.) the test cases were not the same.

4.2. Approach

The data provided by DPWW consisted of a list of 92 defects of both the FAT and the two SAT tests. For each defect a number of attributes was defined, like detection date, correction date, type, description, and test type. Moreover for every defect the severity and priority (both ranging from 1 to 3) were recorded. In Figure 4 an overview is given of the distribution of the defects over the defined defect types.

Based on the similarity of test cases in FAT and SAT it was decided to combine the defects of both test types as input for the analysis. Moreover, because the only subject of our analysis was the operating software of the construction, only defects of the type "software' were taken into account.

Further inspection of the defects listed revealed that one defect was reported twice. In two cases a defect was classified under 'design' while this actually should have been 'software'. Finally a number of defects classified as 'data' actually turned out to be the result of an incorrect parameter value being inserted. Based on the notion that parameter settings are essential for the software to run correctly it was decided to include these defects in the final list. As a result 38 defects remained to be used in the reliability analysis. Finally it was decided to pay special attention to defects with severity 1, since this type of defect causes 'show stoppers,' i.e., failure of the primary functions of the object.

Although the original list contained the detection date of the defects, no records were kept concerning number of hours spent on testing. Based on interviews with the project manager it was therefore decided to assign each test interval (an interval being defined as one day) a test effort of eight hours. Combined with the DPWW's estimate of the number of times the object would perform its primary function per year it was deduced that 1 test interval corresponds with 219 operational days.

Trend tests confirmed reliability growth throughout, so it was decided to use all 38 selected defects in our reliability analysis (see also Figure 5).

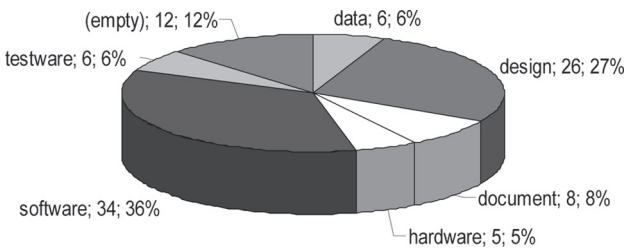

FIGURE 4 **Defects of closable dam operating system software per defect type.**

TABLE 5 **Defects and Hours of Closable Dam FAT and SAT Combined**

Weeks	Defects	Days	Severity
1	4	219	1
1	2	219	2
1	2	219	3
2	2	219	1
2	0	219	2
2	0	219	3
3	1	219	1
3	2	219	2
3	1	219	3
4	1	219	1
4	3	219	2
4	0	219	3
5	0	219	1
5	0	219	2
5	1	219	3
6	1	219	1
6	2	219	2
6	1	219	3
7	1	219	1
7	4	219	2
7	0	219	3
8	1	219	1
8	3	219	2
8	0	219	3
9	0	219	1
9	1	219	2
9	0	219	3
10	0	219	1
10	0	219	2
10	1	219	3
11	1	219	1
11	1	219	2
11	0	219	3
12	0	219	1
12	0	219	2
12	0	219	3
13	0	219	1
13	1	219	2
13	0	219	3
14	0	219	1
14	0	219	2
14	1	219	3

To select one or more suitable models a version of the matrix of Table 1 was used. Based on the total score in the selection matrix the following models were selected to perform a reliability analysis with:

- Yamada model
- Goel-Okumoto/Schneidewind model
- Generalized Poisson model

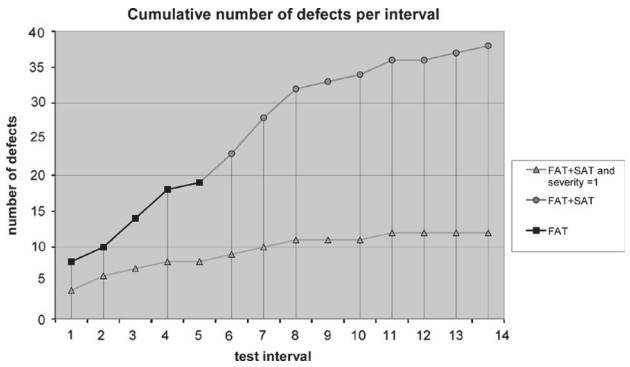

FIGURE 5 **Cumulative number of defects of FAT and SAT of closable dam operating system.**

Note that since we use all data and all test intervals have the same length, the Goel-Okumoto and Schneidewind models coincide.

4.3. Results

As mentioned before, special attention was given to defects with severity 1. Hence, all calculations were carried out twice; the first one with all defects as input data, the second with only defects of severity 1 as input. For each calculation a goodness-of-fit test was carried out in Casre to check the model results (see Table 6). If a specific calculation did not fit the 10% significance level (i.e., we fail to reject the null hypothesis that the model does not fit the data) this is indicated with an 'N' in the appropriate entry. Otherwise a 'Y' was entered.

As can be seen from Table 6 the Yamada S-shaped model failed the GOF test for both calculations. Hence, the outcomes of this model were ignored in the final results as shown in Figures 6–10. The same holds for the Generalized Poisson model when only defects with severity 1 are taken into account.

The Casre manual does not exactly state how the goodness-of-fit tests were performed. In view of our remarks in Subsection 2.4, we have doubts on the validity of these tests. Unfortunately, the

TABLE 6 **Goodness-of-Fit Tests with Significance Level 10% for Closable Dam Data**

	Yamada S-shaped	Goel-Okumoto	Generalized Poisson
Cumulative defects	N	Y	Y
Cumulative defects, severity = 1	N	Y	N

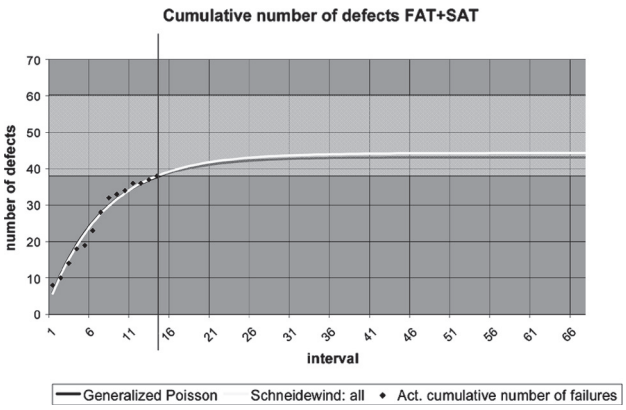

FIGURE 6 **Cumulative number of defects for closable dam data.**

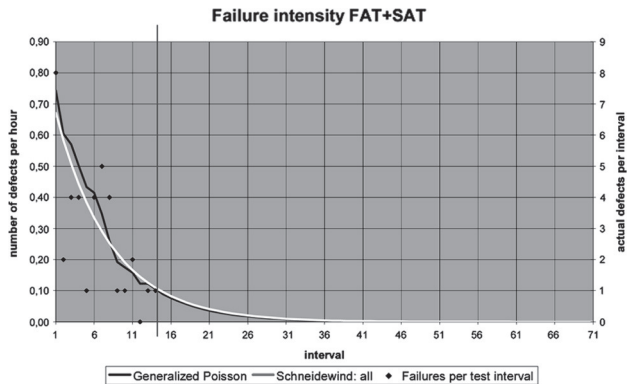

FIGURE 8 **Failure intensity of closable dam data.**

literature does not provide us yet with a better alternative for grouped data.

The light area in Figures 6 and 7 indicates a 95% prediction interval for the number of defects. As can be seen from Figure 6 the expected total number of defects to be found if testing were to be continued for another 30 test intervals (each test interval having a length of 8 hours) is 43 or 44. The first value being predicted by the Generalized Poisson model, the second from by the Schneidewind model. Thus in the next 30 test intervals we are likely to find 5 or 6 defects, the other ones already being detected in the first 14 test intervals.

If only defects with severity 1 are taken into account, the expected total number of defects found if testing were to be continued is 12, this being the exact amount of defects with severity 1 found in the first 14 test intervals!

Casre also supplies plots of the failure intensity like Figure 8. If all defects are taken into account, the failure intensity after a 15th (virtual) test interval will have dropped to less than 0.1 failures per hour (the actual value calculated by both models is 0.083). From that point on failure intensity is more or less halved every 5 test intervals. We recall here that this does not imply that we encounter 0.083 failures per hour. The failure intensity for NHPP models is a subtle mathematical quantity that equals the slope of the expected number of defects. For very small time spans, this is approximately equal to the number of defects per time unit.

The test process was finished after the 14th test interval and after that the software was migrated to production and the object was made operational. Interviews with the project manager revealed that the object performed its primary function approximately once in every test interval of 8 hours. Finally we give the model results in terms of reliability, the corresponding graphical representations being shown in the Figures 9 and 10.

Figure 9 shows the reliability for the next 8 hours as calculated by the mentioned models. At the end of the test (i.e., after the 14th test interval) this reliability

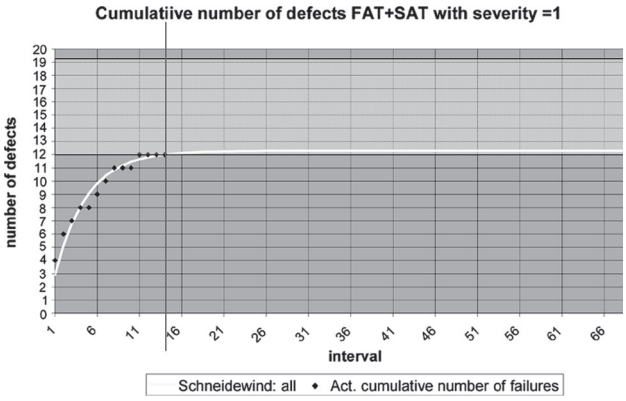

FIGURE 7 **Cumulative number of defects of severity 1 for closable dam data.**

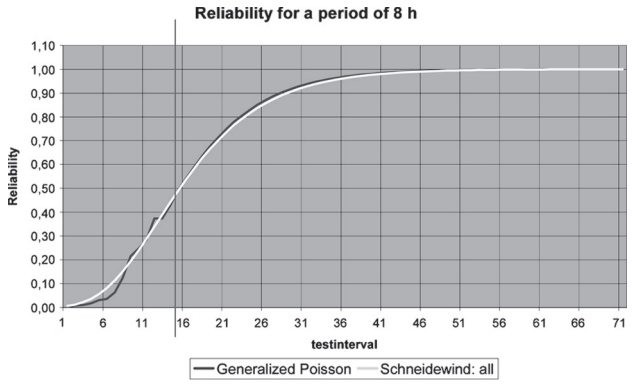

FIGURE 9 **Eight-hour reliability for closable dam data.**

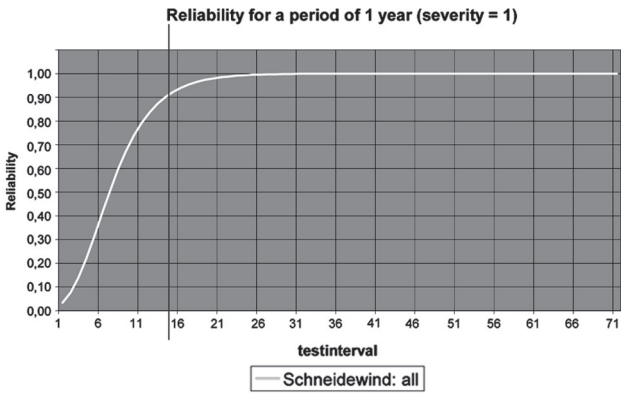

FIGURE 10 **Eight-hour reliability for severity 1 defects of closable dam data.**

was determined as 0.441, increasing to 0.490 after another (virtual) test interval. Unfortunately, the tools that we used did not provide confidence intervals for the quantities of interest.

Reliability is steadily growing if more and more test intervals are added, but this growth is rather slow, and to approach a value greater than 0.95 at least 29 more test intervals are needed. However, this analysis was carried as a post-hoc analysis, so continuing testing was not an option.

If we suppose one test interval to be equivalent to 219 days of operational use, as mentioned in Subsection 4.2, shows the increase in reliability for a period of one year, with only defects with severity 1 taken into account. After the 14th test interval the reliability of the operating software was calculated as 0.9014. In other words: at the time the software was released for production, the probability of the software to fail in the next year and thus cause a major overall failure of the object to perform its primary function is 0.0986.

An extra run was executed to determine the reliability of the software for a period of 15 years. This resulted in a reliability of 0.7499. But as no values for a desired level of reliability of the operating software were available, nothing can be said about this reliability being adequate or not.

4.4. Conclusions of Case Study 2

As stated in the introduction the analysis was performed mainly to decide whether a reliability assessment is feasible, regarding the available information about the test process and its results. Based on our experiences this is indeed the case, although a number of weak points should be removed to improve the quality of any reliability analysis. The lack of an adequate registration of hours tested for instance forced us to make assumptions which of course influence the outcomes of the analysis. Furthermore we would like to stress the fact that the model calculations for defects with severity 1 were performed using only a limited amount of data. This obviously is a rather small number to use in statistical calculations (which is reflected in wide confidence bounds) and the phenomenon that although it seems that testing revealed most errors (see Figures 6 and 7), reliability can still be considerably improved by further testing (see Figures 9 and 10). Consequently the results for this category should be viewed with a certain reserve.

5. CONCLUSIONS

We presented two case studies that illustrate different goals of software reliability analyses as well as practical data collection problems. We indicated some methodological issues that often are neglected. For some issues new developments in the statistical literature yields sound alternatives to not fully correct common procedures. Unfortunately, for many models the procedures to analyze grouped data are not satisfactorily available or worked out in detail. In particular, research is needed to develop goodness-of-fit tests and confidence intervals for parameters and derived quantities like reliability and failure intensity.

ABOUT THE AUTHORS

Ed Brandt has been working in information technology since 1982. His experience extends from software engineering projects to project management and consultancy in both national and international projects. He has specialised in software testing since 1996 and is the author of various articles and papers on this subject, such as the Test Process Scorecard. Together with Rob Henzen he founded Refis System Reliability Engineering in 2003. Refis is specialized in metrics for quality systems and reliability analysis on software.

Isaac Corro Ramos obtained his Master's degree in Mathematics from the University of Sevilla in 2001. Between 2002 and 2004 he worked as a programmer in Coritel-BPM (Accenture). In 2005 he started as a Ph.D. student at the Department of Mathematics and Computer Science of the Eindhoven University

of Technology working on the STRESS (Statistical Testing and Reliability Estimation of Software Systems) project under the guidance of professors Alessandro Di Bucchianico and Kees van Hee.

Alessandro Di Bucchianico is an associate professor of statistics at the Department of Mathematics and Computer Science of the Eindhoven University of Technology and coordinator of the industrial statistics programme at EURANDOM, a research institute on stochastics in Eindhoven. His research and teaching interest include industrial statistics and software reliability. Since 2004, he is actively involved in a software reliability programme at LaQuSo, the Research Laboratory of Quality Software at Eindhoven University of Technology.

Rob Henzen is working in information technology since 1990. He has a vast experience in programming, designing and testing software systems. In the course of the last few years he has delivered a number of presentations on software reliability. Within Refis he is working on research and development of metrics systems, and as test manager in several assignments.

REFERENCES

Bain, J. L., Engelhardt, M., Wright, F. T. (1985). Tests for an increasing trend in the intensity of a Poisson process. *J. Amer. Stat. Assoc.*, 80:419–422.

Bhattacharjee, M., Deshpande, J. V., Nimbalkar Naik, U. V. (2004). Unconditional tests of goodness of fit for the intensity of time-truncated nonhomogeneous Poisson processes. *Technometrics*, 46(3):330–338.

Cohen, A., Sackrowitz, H. B. (1993). Evaluating tests for increasing intensity of a Poisson process. *Technometrics*, 35(4):446–448.

Goel, A. L. (1985). Software reliability models: assumptions, limitations, and applicability. IEEE Trans. *Software Eng.*, 11(12):1411–1423.

Hossain, S. A., Dahiya, R. C. (1993). Estimating the parameters of a nonhomogeneous Poisson process model for software reliability. *IEEE Trans. Rel.*, 42(4):604–612.

Jeske, D. R., Pham, H. (2001). On the maximum likelihood estimates for the Goel-Okumoto software reliability model. *Amer. Statist.*, 55(3):219–222.

Joe, H. (1989). Statistical inference for General-Order-Statistics and Nonhomogeneous-Poisson-Process software reliability models. *IEEE Trans. Software Eng.*, 15(11):1485–1490.

Joe, H., Reid, N. (1985). Estimating the number of faults in a system. *J. Amer. Stat. Assoc.*, 80(389):222–226.

Kharchenko, V. S., Tarasyuk, O. M., Sklyar, V. V., Dubnitsky, V. Yu. (2002). The method of software reliability growth models choice using assumptions matrix. In *COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, pp. 541–546.

Knafl, G. J. (1992). Solving Maximum Likelihood equations for two-parameter software reliability models for using grouped data. In: *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 205–213. IEEE.

Knafl, G. J., Morgan, J. (1996). Solving ML equations for 2-parameter Poissonprocess models for ungrouped software-failure data. *IEEE Trans. Rel.*, 45(1):42–53.

Kuo, L., Yang, T. Y. (1996). Bayesian computation for nonhomogeneous Poisson processes in software reliability. *J. Amer. Statist. Assoc.*, 91(434):763–773.

Langberg, N., Singpurwalla, N. D. (1985). A unification of some software reliability models. *SIAM J. Sci. Statist. Comput.*, 6(3):781–790.

Lyu, M. R. ed. (1996). *Handbook of Software Reliability Engineering*. New York: McGraw-Hill and IEEE Computer Society.

Musa, J. D. (2006). *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, 2nd ed. Bloomington: Author House.

Ohba, M. (1984). Software reliability analysis models. *IBM J. Res. Develop.*, 28(4):428–443.

Pham, H. (2006). *System Software Reliability. Springer Series in Reliability Engineering*. London: Springer.

Raftery, A. E. (1987). Inference and prediction for a general order statistic model with unknown population size. *J. Amer. Statist. Assoc.*, 82(400):1163–1168.

Rigdon, S. E., Basu, A. P. (2000). *Statistical Methods for the Reliability of Repairable Systems*. New York: Wiley.

Rongguan, L., Heliang, F. (2002) Estimation of the parameters for Musa-Okumoto and inverse linear models in software reliability. *Chinese J. Appl. Prob.*, 18(4):425–430.

Singpurwalla, N. D., Wilson, S. P. (1994). Software reliability modeling. *International Statistical Review*, 62:289–317.

Thompson, W. A., Jr. (1981). On the foundations of reliability. *Technometrics*, 23(1):1–13.

Thompson, W. A., Jr. (1988). Point *Process Models with Applications to Safety and Reliability*. New York: Chapman and Hall.

Xie, M., Dai, Y.-S., Poh, K.-L. (2004). *Computing Systems Reliability. Models and Analysis*. New York: Kluwer.

Xie, M., Hu, Q. P., Wu, Y. P., Ng, S. H. (2007). A Study of the Modeling and Analysis of Software Fault-detection and Fault-correction Processes, *Qual. Reliab. Engng. Int.*, 23(4):459–470.

Yin, L., Trivedi, K. S. (1999). Confidence interval estimation of NHPP-based software reliability models. In: *Proc. 10th Int. Symp. Software Reliability Engineering (ISSRE 1999)*, pp. 6–11.

Zhang, X., Jeske, D. R., Pham, H. (2002). Calibrating software reliability models when the test environment does not match the user environment. *Appl. Stoch. Models Bus. Ind.* 18:87–99.

Zhao, J., Wang, J. (2005). A new goodness-of-fit test based on the Laplace statistic for a large class of NHPP models. *Comm. Statist. Simulation Comput.*, 34(3):725–736.