Design Reliability - Estimation through Statistical Analysis of Bug Discovery Data

Yossi Malka, Avi Ziv
IBM Research Lab in Haifa
MATAM
Haifa 31905, Israel
email: {yossi, aziv@vnet.ibm.com}

Abstract

Statistical analysis of bug discovery data is used in the software industry to check the quality of the testing process and estimate the reliability of the tested program. In this paper, we show that the same techniques are applicable to hardware design verification. We performed a study on two implementations of state-of-the-art PowerPC processors that shows that these techniques can provide quality information on the progress of verification and good predictions of the number of bugs left in the design and the future MTTF.

1 Introduction

Functional verification comprises a large portion of the effort in designing a processor [1]. The investment in expert time and computer resources is huge, and so is the cost of delivering faulty products [3]. In current industrial practice, random architectural test program generators and simulation methods are used to implement large portions of the verification plans [1, 2, 11]. Verification is done by comparing the behavior of the tested design to its expected behavior, as defined by the design's architecture and specification. Each time the behavior of a tested design diverges from the expected behavior, a failure occurs. When a failure occurs, the bug that caused the failure is tracked and, in most cases, the fault is removed or bypassed. Each time a bug is removed from the tested design, there are less elements that can cause failures of the design, and therefore, the mean time to the next failure (MTTF) increases or, in other words, the reliability of the design grows.

One of the major challenges and concerns that verification people are facing is how to measure the quality of the verification process and determine the reliability of the tested design. Verification people are often asked questions like: "When can a certain level of reliability be expected?" or "What are the effects of a new test generation tool on the verification process?"

Accurate predictions of the reliability of a design are extremely important when determining dates for tape-out or general release. The huge costs of a tape-out and the complexity of the hardware bring-up process that follows it leads to a small number of tape-outs. It is, therefore, important to tie tape-outs with reliability milestones, such as the expected time when the processor is able to boot its operating system. This requires estimation of the current reliability of the design. Moreover, since a tape-out needs to be planned months in advance, it is important to accurately predict when the desired reliability is going to be reached in order to avoid a premature tape-out.

During the verification process, a lot of information regarding the bugs in the tested design and the verification process is collected. This information includes data on when the bug was discovered, how it was discovered, and its severity. The information also includes the amount of simulation, measured in cycles, that was executed. This information is used to evaluate the reliability of the tested design, examine the verification process, and to make decisions regarding future plans, such as starting the next testing phase and tape-outs. Since the amount of data that is gathered is huge, usually only statistics on the information are used.

Currently, intuitive decisions are taken based on the shape of the curve of the number of bugs found versus the amount of simulation. Statistical analysis of the data can provide more concrete observations. It can be used to detect trends that occurred in the past, determine the current state of the tested design, and predict its future reliability.

The software reliability engineering (SRE) community considers statistical bug analysis a major part of software reliability. Statistical bug analysis of software is used in many organizations, such as AT&T [12], Tandem [15], and JPL [9]. The analysis is used to measure the reliability of the software in a wide variety of systems, such as operating systems, switching devices, and spacecraft controllers. Statistical bug analysis is used in these systems to measure the reliability of the software system, predict its future reliability, and optimal release date and to assist in design and testing plans.

The design, coding, and testing of software is closely related to design, implementation, and verification of hardware designs. In both hardware and software, the design process starts with high level design of the whole system, which is then broken into smaller units. Each unit is implemented and tested by itself, and then the whole system is constructed out of the units. Testing of software systems is also done in a similar way to hardware design verification. Input patterns are generated and the behavior of the module is compared to its expected behavior, which is defined by the specification and architecture. Finally, in both hardware and software, bugs are detected and removed from the

35th Design Automation Conference ®
Copyright ©1998 ACM
1-58113-049-x-98/0006/$3.50 DAC98 - 06/98 San Francisco, CA USA

Design	Design Type	Duration of Testing [Month]	Number of bugs found	Simulation cycles [$\times 10^6$]
Processor A	Full processor	8.5	719	68052
Processor B	Execution units	8.5	236	2383

Table 1: Source bug discovery data summary

design during testing, causing reliability growth in the module. Therefore, the vast experience and knowledge gathered in statistical analysis of software bugs can be used (at least as the starting point) for statistical bug analysis during verification of hardware modules.

In this paper we describe the results of a study we did on the applicability of statistical bug analysis techniques used in software reliability engineering. We used bug discovery data obtained for two advanced PowerPC processors, and applied to it two types of statistical bug analysis that are commonly used is software reliability engineering. The techniques that we are using are *trend analysis* for reliability growth, and *modeling.* These techniques can provide answers to questions like:

- How does the introduction of a new test generator affect the reliability growth?
- What is the mean time to the next failure?
- When can a certain level of reliability be expected? That is, when will the probability of a failure in X time units be less than a given ϵ?

The answer to the first question can be provided using trend analysis of the reliability growth. For example, if after introduction of a new test generator, the reliability growth slowed down, we can then assume that the new generator is more efficient in detecting bugs, since it can produce more failures. Answering the second and third questions involve prediction of the future reliability of the tested design. The second question requires short term prediction about the time of the next failure, while the third question requires long term prediction.

The study reported in the paper was performed on two different designs. The first design, called Processor A in the paper, is a super-scalar PowerPC processor. The second design, called Processor B, includes all the execution units of a different super-scalar PowerPC processor. For both designs, we used bug discovery data that was collected over a period of more than 8 months. During this period, the designs were heavily tested (over 68 billion simulation cycles for Processor A and over 2 billion for Processor B), and several hundred bugs were found. The data used for the analysis, for both designs, was the number of passing simulation cycles executed and the number of bugs found every day. The data includes only bugs found using random simulation, and ignores bugs found using other verification techniques, such as formal verification and inspections. Table 1 summarizes the source bug discovery data that was used for the analysis.

The results of the analysis enable us to detect the affects of the introduction of a major function in one processor on its reliability growth, and to predict accurately the number of bugs discovered and the MTTF in the other processor three months in advance. Overall, our results show that these techniques can be very useful in monitoring the verification process and estimating its future progress.

Note that in this paper we concentrate on the analysis results, not on a description of the methods. A complete description of the methods used in this paper can be found in many text books, such as [9] and [13].

2 Trend Analysis

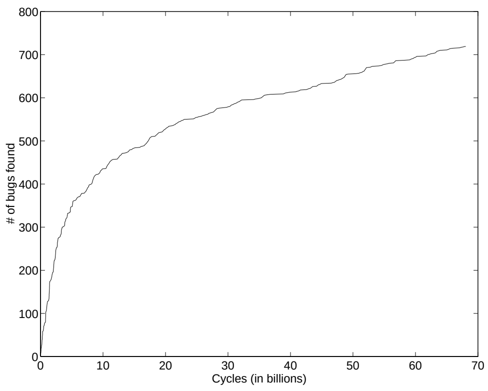
The task of the verification process is to increase the reliability of the tested design or, in other words, to decrease the bug intensity of the design. This is done by detecting and removing bugs that cause failures from the design. Therefore, the first indication we would like to get from the statistical analysis is whether there is a reliability growth in the tested design. This type of information can be provided by trend analysis.

Reliability growth means that the bug discovery intensity in the design is decreasing, and therefore the time between bug discoveries is increasing. This trend can be detected by looking at the cumulative number of bugs as a function of the testing time. Since the bug intensity is a derivative of the cumulative number of bugs, decreasing bug intensity means that the function of the cumulative number of bugs vs. simulation time should be concave.

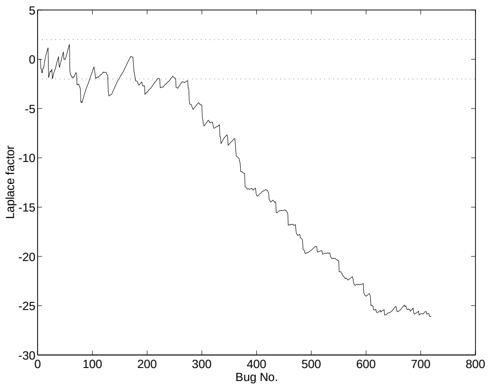
Taking a look at the raw data provides only a rough indication of reliability growth for two main reasons:

- The raw data of time between bugs is only an instantiation of a random process. Therefore, the cumulative number of bugs is not a smooth concave function, but a function with many bumps and extreme points.
- The raw data information can very easily hide local changes in the trend of the reliability growth or strong local variations can hide the overall trend of reliability growth.

There are several analytical tests for reliability growth that are designed to overcome these problems. The idea behind these tests, known as trend tests, is to test a null hypothesis H_0 versus an alternative H_1. Usually, H_0 corresponds to the assumption that the reliability is not changing, while H_1 corresponds to the assumption that reliability undergoes a monotonic trend. One of the most widely used trend tests is the Laplace test [4]. The Laplace test assumes that the bug discovery intensity is equal to $\lambda(t) = e^{a+bt}$. The test compares the null hypothesis, constant bug intensity ($b = 0$), versus the hypothesis that the bug discovery rate is decreasing, that is $b < 0$. The test procedure is to calculate the Laplace factors $u(\cdot)$. Simply stated, the test procedure compares the midpoint of the tested interval with the median bug discovery time. Negative values of Laplace factors indicate that more bugs were discovered in the first half of the inspected interval, and therefore there is a reliability growth. Positive values of Laplace factors indicate that the bug intensity is increasing. If the values are in the range -2 to 2, then the reliability is stable or, more precisely, the null hypothesis cannot be rejected.

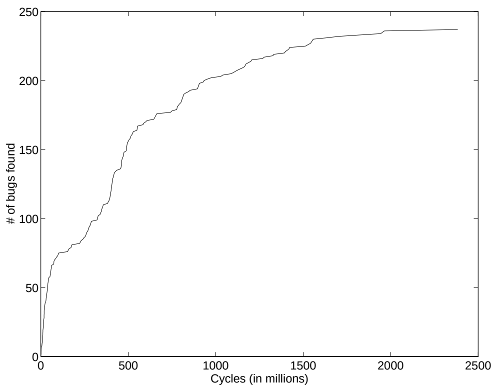


(a) Raw data

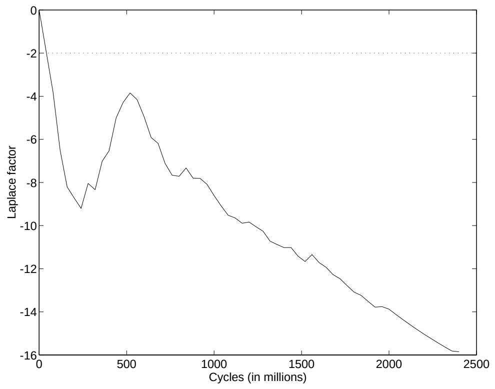


(b) Fault based trend analysis

Figure 1: Raw data and trend analysis for Processor A



(a) Raw data



(b) Cycle based Trend analysis

Figure 2: Raw data and trend analysis for Processor B

Laplace factors can be calculated both as a function of the testing time or the number of bugs found. For example, Figure 1b shows the Laplace factors versus the number of bugs found for Processor A, while Figure 2b shows the Laplace factors versus the number of simulation cycles for Processor B. Often, using one form of the test helps to show things that are not clear in the other form. For example, The first region in Figure 1b, where there is almost no reliability growth, is hard to detect in a cycle based plot, since the region covers only 3% of the total amount of simulation cycles.

Figure 1 shows the raw bug discovery data and the result of the Laplace test for reliability growth for Processor A. The raw data shows that there is an overall reliability growth, since the plot of the total number of bugs vs. the total amount of simulation cycles is concave. This look at the raw data provides only a rough indication of the reliability growth. A better indication can be obtained using trend analysis. The trend analysis plot shows Laplace factors as a function of the bug number. From the figure, we can see that, overall, there is a continuous reliability growth, but there are three regions of growth. In the first region, which exists until about the 270th bug (which took about 3% of the simulation cycles), there is a small decrease in the Laplace factor which indicates a small reliability growth. The second region, between bug 270 and 610, shows a much faster drop in the value of the Laplace factor, which indicates a faster reliability growth. In the last region, the Laplace factors drop slowly, which indicates a slower growth in reliability.

The first region in the trend, with slow reliability growth, can be explained by difficulties in the start-up of the verification process, and therefore it might not be a good representative for the overall reliability growth of the processor and should be ignored. The slowdown in the reliability growth rate in the third region can be explained by the level of reliability that was achieved by that time. This type of behavior is observed in all existing models for reliability growth.

Figure 2 shows the raw bug discovery data and the Laplace factors for Processor B. The raw data plot shows that, overall, the reliability of the processor grows with time, with some anomaly in the beginning. A look at the Laplace factors clearly shows that, in the area between 200 and 500 million cycles, there is an increase in the Laplace factors, which indicates a decrease in the reliability of the tested design. After that period, the reliability starts to grow again and continues to grow continuously until the end of the examined period. When we tried to find the reason for this behavior, we found that this time corresponds to the time when testing of a major function of the processor started. This caused an increase in the bug rate due to the bugs in the new function, and thus a decrease in the reliability.

3 Modeling and Predictions

Trend analysis can provide information about trends that occurred during the verification process, and answer important questions, such as: does the reliability of the tested design grow, and how do specific events affect this growth? Still, trend analysis cannot answer questions like: how many bugs are left in the tested design, and when will a desired level of reliability be achieved? To answer such questions, predictions of the future behavior of the tested design and the verification process are needed. Usually, predictions of the future behavior of a process are obtained by matching the history information of the process to some predefined model (or models) and using the properties of the model to describe predicted future behavior [13].

The basic approach of modeling is to fit past data to a model that describes the expected behavior of the data, and to use the model to predict future behavior. It has been shown that, in order to provide good predictions, the model should accurately describe the observed process or at least be close to it. Predictions obtained using models that do not fit the observed data can be misleading. Therefore, selecting the proper model is probably the most important part of modeling.

Because of the similarities in the processes of hardware and software design and testing, bug models that have been tried and found applicable in software are good candidates to be applicable in hardware as well. In our study, we tested the bug discovery data of the processors against several commonly used models in software reliability engineering. A short description of three of the models is given below. A complete description of the models can be found in their references or in books like [9] and [13]. The models that we used are:

- Goel Okumoto non-homogeneous Poisson process model (GO) [8]. This model is a finite model that assumes that the number of bugs observed by time t follow a non-homogeneous Poisson process with a bug rate that drops exponentially.
- Duane's model [6]. This model is an infinite Poisson model that assumes that the cumulative number of bugs grows exponentially with time.
- Musa Okumoto logarithmic Poisson model (MO) [14]. This is an infinite Poisson model that assumes that the bug rate drops exponentially with the expected number of bugs experienced.

As stated earlier, modeling fits the observed data into a predefined model and uses the model to predict future behavior. Fitting the observed data to the model is done by estimating the parameters of the model based on the observed data. There are several estimation techniques, each finding the best estimator based on different criteria. The most commonly used estimators are the maximum likelihood estimator (MLE), the least square error (LSE) and methods of moments. After the parameters of the model are estimated, the model can be used to predict future behavior of the tested design.

The predictions provided by different models, based on the same bug discovery data, can differ considerably. For example, Table 2 compares the predictions made by the three models for the number of bugs found and the MTTF for Processor A after 120 billion cycles of simulation. The table shows that there is a huge difference in the predictions made by the three models. Therefore, in order to get quality predictions, we need to find the model that fits the bug discovery behavior of the tested design in the best way, and thus has the best chance to provide the most accurate predictions. Unfortunately, no one model is clearly better than the other models, and can be used as the model for reliability predictions.

	Number of bugs found	MTTF [10^6 cycles]
Goel-Okumoto	810	750
Duane	886	355
Musa-Okumoto	756	4300

Table 2: Predictions for Processor A after 120 billions cycles of simulations

There are many possible ways to check how well a model fits the actual process. The first method is to compare the observed data with the expected behavior based on the model and the estimated parameters. Even when the observed data is used to estimate the parameters of the model, there could be a big differences between the behavior predicted by the model and the actual behavior when the actual behavior does not fit the model.

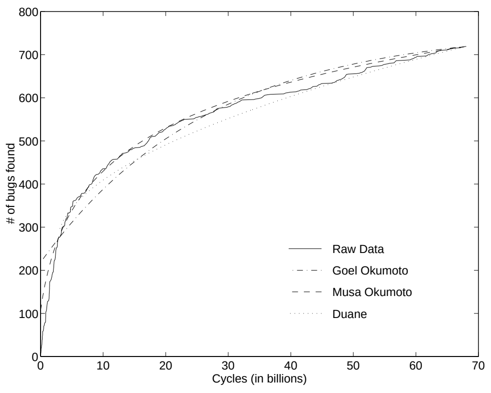
Figure 3a shows how the bug discovery data of Processor A is approximated by the three models described above. The figure shows that none of the models totally fit the bug discovery data, but some of them, specifically, the Musa Okumoto model, are fairly close to it.

While a plot that compares actual data with a given model (like Figure 3a) can provide an intuitive feeling of the fitness of the observed data, it cannot provide quantitative answers on the fitness level, or the level of certainty that the data fits the model. There are several statistical tests that can provide the answers to these questions, such as the *chi-square test* that compares sample data to a probability density function, and the *Kolmogorv-Smirnov test* that compares sample data to a cumulative distribution function [5]. The answers from the goodness-of-fit tests can be used to determine the quality of the fitness and the interval of confidence for estimated values.

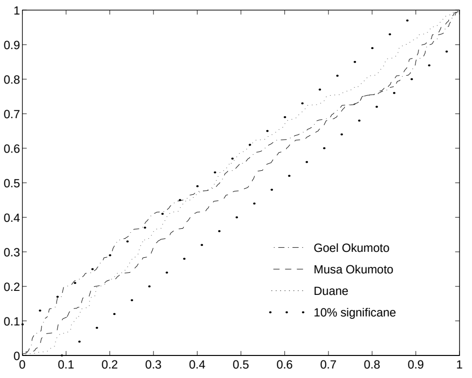
Figure 3b shows the results of the Kolmogorv-Smirnov fitness test for the three models for Processor A. In this test, the closer the curve of the model to the diagonal, the better the fitness is. A model whose curve is inside the 10% significance lines is considered to have a good fit. The figure shows that only the curve for the Musa Okumoto model is inside that region, and thus not only is it the most fitting model, but it is also the only one with a good fit.

Table 3 compares the results of the Kolmogorv-Smirnov fitness test for the three models for Processor B in two cases. The first case is when all the bug discovery data is considered, and the second case is when only the data after 500 million cycles is considered. The table illustrates the importance of selecting the correct data for analysis. The complete bug discovery data for Processor B contains two regions, as Figure 2b shows, one before the major feature was activated, and one after it. If we consider both regions together, then none of the models has a good fit to the data but, if only the second region is considered, the fitness of all the models improves, and 2 out of the 3 models have a good fitness.

The Kolmogorv-Smirnov fitness test is not the only possible test of the fitness of models; there are many other tests that can be used to test the fitness. Since the goal of modeling is to provide predictions about future behavior, many such tests involve checking the quality of predictions made by the models. The most basic test is to compare the predicted median time to the next bug discovery, with the measured time. Other fitness tests include long-term prediction

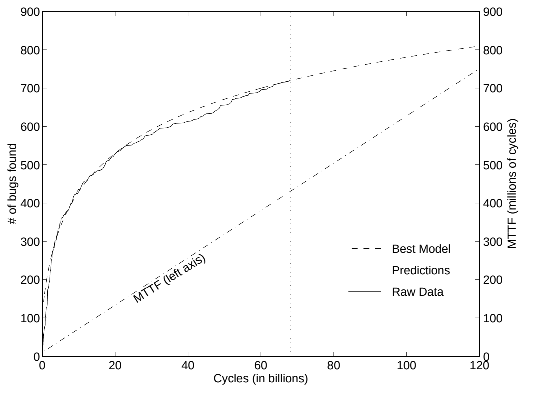


(a) Models Estimation

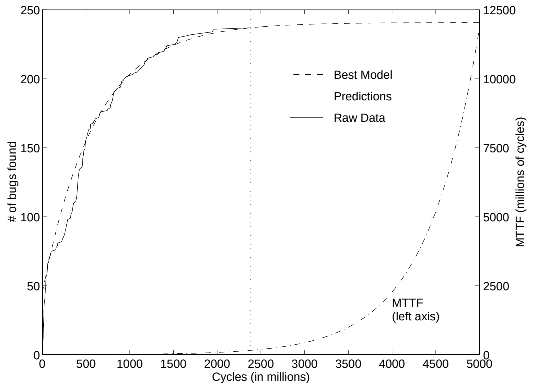


(b) Estimation fitness

Figure 3: Model estimation and estimation fitness for Processor A



(a) Processor A



(b) Processor B

Figure 4: Long term predictions

fitness and stability of the parameters of the model over time [9].

After all the desired fitness tests are made, the models can be ranked according to their performance in the tests, and the model with the highest ranking should be the one used for future predictions. There are several possible criteria according to which the ranking can be done, each giving different weights to different fitness tests. After the best model is chosen, provided that its absolute fitness is good enough, it can be used to predict the future behavior of the tested design as regards the design's reliability.

Figure 4 shows the predictions for MTTF (mean time to failure) and the number of bugs discovered for both processors, using the best fitting models. The best fitting model for Processor A is the Musa Okumoto model, and the best fitting model for Processor B is the Goel Okumoto model. The figure shows that there is a big difference between the predictions of future behavior for the two processors. For Processor A, the best fitting model is an infinite model, which predicts a continuous increase in the number of bugs found and linear growth in the MTTF. The prediction for Processor B, on the other hand, is for a finite number of bugs and an exponential growth in the MTTF.

We compared the total number of bugs and the MTTF for Processor A, as predicted by the Musa Okumoto model with the actual bug discovery data taken three months after the end of the study period. At the time of the comparison, a total of 105 billion cycles of simulation were executed and 794 bugs were found (that is 37 billion more cycles and 75 more bugs since the end of the study). The comparison results are summarized in Table 4. The table shows that there is a small difference of less than 7% in both the prediction of the MTTF and the number of bugs found since the prediction, despite the fact that the prediction was made 3 months in advance.

	All data	After 500M cycles
Goel-Okumoto	0.169	*0.104*
Duane	0.169	0.226
Musa-Okumoto	0.126	0.155
10% Significance	0.089	0.157

Table 3: Model estimation fitness for Processor B

4 Conclusions

In this paper we described the results of statistical analysis of bug discovery data which we performed on two state-of-the-art PowerPC processors. The analysis we performed is similar to the analysis that is done in software reliability engineering. The results of the analysis enable us to detect the

	Musa-Okumoto Prediction	Actual	Difference [%]
Total Number of bugs found	789.6	794	0.4
Number of bugs since prediction	70.6	75	4.0
MTTF [$\times 10^6$ cycles]	646	620	6.7

Table 4: Comparison of Musa Okumoto long term predictions and actual data for Processor A

effects of the introduction of a major feature in one processor on its reliability growth, and to predict accurately the number of bugs and the MTTF in the other processor.

The conclusion we came to from our study is that statistical analysis of bug discovery data, in general, and techniques used in software reliability engineering, in particular, can and should be used to analyze bug discovery data during hardware verification, and that such an analysis can provide useful information on the state of the verification process and the reliability of the tested design.

This type of analysis is very useful in determining optimal release dates for processors, since it is capable of providing an estimation of the post-release bug discovery rate. This information, combined with data on the cost of such bugs and the cost of a delayed release, can be used to determine the optimal time to release the processor.

The cost of the statistical analysis is very low. Only accurate data on the times in which bugs were found and the amount of testing done are needed. The analysis itself, although based on sophisticated statistical concepts, can be done using one of several commercially available tools, such as CASRE [10], and SMERFS [7].

There are several open questions regarding the use of statistical analysis of bug discovery data in hardware design verification. First, we need to find out which models are applicable to the analysis. In our study we found out that some models used in SRE are also applicable to hardware. The question is whether all SRE models are applicable to hardware design and whether there are new models that are applicable to hardware design, but not to software.

A second interesting issue is the relationship between testing environment and real environment. In many cases, the reliability prediction is required in terms of real operation. Since statistical analysis is done in a testing environment, it can provide predictions only in this environment. Unlike software testing, which is done in conditions similar to real operation of the system, hardware testing is done in a much more stressful environment. Therefore, one cycle in testing is equal to many cycles in real operation. It is interesting to investigate the relationship between testing cycles and real cycles, and how the testing environment affects this relationship.

References

[1] A. Aharon, D. Goodman, M. Levinger, Y Lichtenstein, Y. Malka, C. Metzger, M. Molcho, and G. Shurek. Test program generation for functional verification of PowerPC processors in IBM. In *Proceedings of the 32nd Design Automation Conference*, pages 279–285, June 1995.

[2] A.M. Ahi, G.D. Burroughs, A.B. Gore, S.W. LaMar, C.R. Linand, and A.L. Wieman. Design verification of the HP9000 series 700 PA-RISC workstations. *Hewlett-Packard Journal*, 14(8), August 1992.

[3] B. Beizer. The Pentium bug, an industry watershed. *Testing Techniques Newsletter, On-Line Edition*, September 1995.

[4] D.R. Cox and P.A.W. Lewis. *The Statistical Analysis of a Series of Events*. Chapman and Hall, 1978.

[5] M.H. DeGroot. *Probability and Statistics*. Series in Statistics. Addison Wesley, 1986.

[6] J.T. Duane. Learning curve approach to reliability monitoring. *IEEE Transaction on Aerospace*, 2:563–566, 1964.

[7] W.H. Farr and O.D. Smith. Statistical modeling and estimation of reliability functions for software (SMERFS) user's guide. TR-84-373, revision 1, NSWC, 1988.

[8] A.L. Goel and K. Okumoto. Time-dependent error-detection rate model for software and other performance measure. *IEEE Transactions on Software Engineering*, 11(12):285–306, December 1985.

[9] M.R. Lyu. *The Handbook of Software Reliability Engineering*. McGraw Hill, 1996.

[10] M.R. Lyu and A.P. Nikora. CASRE: A computer-aided software reliability estimation tool. In *Proceedings of the 5th International Workshop on Computer-Aided Software Engineering*, pages 264–275, July 1992.

[11] J. Monaco, D. Holloway, and R. Raina. Functional verification methodology for the PowerPC 604(TM) microprocessor. In *Proceedings of the 33rd Design Automation Conference*, pages 319–324, June 1996.

[12] J.D. Musa. Software-reliability-engineered testing. *Computer*, 29(11):61–68, November 1996.

[13] J.D. Musa, A. Iannino, and K Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, 1987.

[14] J.D. Musa and K. Okumoto. A logarithmic Poisson execution time model for software reliability measurement. In *Proceedings of the Seventh International Conference on Software Engineering*, pages 230–238, March 1984.

[15] A. Wood. Predicting software reliability. *Computer*, 29(11):69–77, November 1996.