Concise Paper

When to Stop Testing for Large Software Systems with Changing Code

Siddhartha R. Dalal and Allen A. McIntosh

Abstract—**Developers of large software systems must decide how long software should be tested before releasing it. A common and usually unwarranted assumption is that the code remains frozen during testing. We present a stochastic and economic framework to deal with systems that change as they are tested. The changes can occur because of the delivery of software as it is developed, the way software is tested, the addition of fixes, and so on.**

Specifically, we report the details of a real time trial of a large software system that had a substantial amount of code added during testing. We describe the methodology, give all of the relevant details, and discuss the results obtained. We pay particular attention to graphical methods that are easy to understand, and that provide effective summaries of the testing process. Some of the plots found useful by the software testers include the Net Benefit Plot, which gives a running chart of the benefit; the Stopping Plot, which estimates the amount of additional time needed for testing; and diagnostic plots. To encourage other researchers to try out different models, all of the relevant data are provided.

Index Terms—**Optimal stopping rule based on cost, graphical methods, software metrics, statistical inference, software reliability model, software fault detection**

I. Introduction

Software testing is a necessary but expensive process, consuming one-third to one-half of the cost of a typical development project. Testing a large software system costs thousands of dollars per day. Overzealous testing can lead to a product that is overpriced and late to market, whereas fixing a fault in a released system is usually an order of magnitude more expensive than fixing the fault in the testing lab. Thus, the question of how much to test is an important economic question. Similar economic formulations of when to stop testing have been discussed by [2]–[5], [7], [18].

This concise paper provides a case study and discusses various implementation issues to guide practitioners. Specifically, we present details of a large telecommunications software system that we call System A. Our emphasis is on graphical methods to produce effective summaries and to help in decision making.

II. System A

System A is a large telecommunications software system, consisting of approximately 7 000 000 noncommentary source lines (NCSL). Here we focus on the system test of one release of System. The release contained approximately 400 000 new or changed noncommentary source lines (NCNCSL). Some of these added new features, and the remainder fixed existing faults.

In order to provide timely delivery of the release to customers, the final phases of development overlap the start of system test. As a consequence, the system being tested is a moving target. The source code used for system test was updated every night throughout the test period. (This is not quite as bad as it sounds. All of the code for new features was in place by the end of the first third of the test period.)

Every day, testers were asked to record the amount of time that they spent testing the current release. We have used this metric to model the testing process rather than the metric of execution time often discussed in the literature [15], for two reasons. First, most of the execution time is spent on regression testing, which does not typically detect a large number of faults. Second, the testing environment is distributed, and accumulating execution times over different types of processors is meaningless.

When a fault was found, it was not fixed immediately unless it rendered all further testing impossible. Instead, a report on the fault was sent to the development organization, which repaired the fault whenever practical. The report included the date on which the fault was found. Our fault data are based on these reports.

In Table I, we give all of the relevant data for System A, namely, the staff time spent testing, the number of faults found, and the additional NCNCSL under test for each calendar day. The initial testing of System A was by subsystem, and the numbers of NCNCSL have been adjusted to reflect this. When we did not have daily NCNCSL data, we used linear interpolation.

III. Basic "When to Stop Testing" Model

In order to decide when to stop testing, we need an economic and a stochastic model for the testing process. First, we summarize the model in [3]–[5], where the code is assumed to be fixed.

The stochastic model assumes that there are N faults in the software (N unknown), and that the times to find faults are observable and are i.i.d. exponential with rate μ. Models of this type have been proposed and used in the past with minor variations [9]–[15].

The economic model defines the cost of testing to be

$$NC(t, K(t), N) = ft - cK(t), \tag{1}$$

where $K(t)$ is the number of faults observed to time t, and f is the cost of operating the testing lab per unit time. The constant $c = a - b$, where a is the cost of fixing a fault when found while testing, and b is the cost of fixing a fault when found in the field. Thus, c is the net cost of fixing a fault after, rather than before, release.

Under somewhat more general assumptions, Dalal and Mallows found the exact optimal stopping rule by assuming that N is Poisson (λ), and that λ is Gamma (α, β). The structure of the exact rule, which is based on stochastic dynamic programming, is rather complex [4], [5]. However, for large N, which is necessarily the case for large systems, the optimal stopping rule is: stop as soon as $f(e^{\mu t} - 1)/(\mu c) \geq K(t)$. Besides the economic guarantee, it gives a guarantee on the number of remaining faults, namely that it has a Poisson distribution with mean $f/(\mu c)$. Thus, instead of determining the ratio f/c from economic considerations, we can choose it so that there are probabilistic guarantees on the number of remaining faults. Some practitioners may find that this probabilistic guarantee on the number of remaining faults is more relevant in their application. (See [2] for a more detailed discussion.) Finally, by using reasoning similar to that used in deriving equation (4.5) of [4], it can be shown that the current estimate of the additional time required for testing, δt, is given by $\delta t = \mu^{-1} \log\{c\mu K(t)/(f(e^{\mu t} - 1))\}$.

Manuscript received October 21, 1992; revised January 1994. Recommended by Littlewood.

The authors are with the Information Sciences and Technologies Laboratory, Bellcore, Morristown, NJ 07960–6438.

IEEE Log Number 9216605.

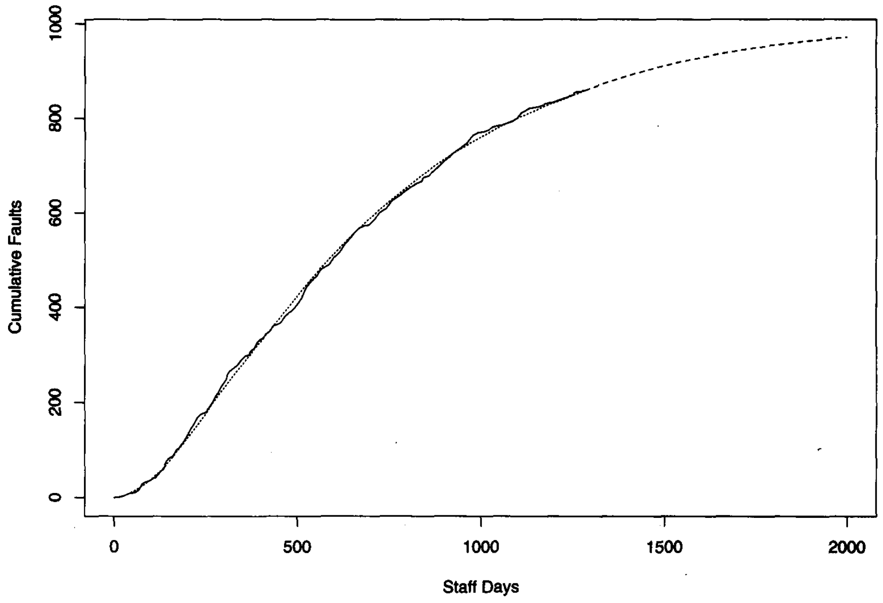


Fig. 1. Model fit.

IV. Enhanced "When to Stop Testing" Model: Code Changes

Because code changes to System A that occur during testing may add more faults, and because only the counts, rather than actual fault times, are observed, the basic model needs to be changed.

Let the testing process be observed at time point t_i with N_{i+1} faults in the system, where N_{i+1} is Poisson with mean $\lambda_{i+1}, i = 0, \cdots, h$, and $t_0 = 0$. In each interval (t_{i-1}, t_i), only M_i, the number of faults found in the interval, is observed. As in the basic model, we assume that the the times to find faults are i.i.d. exponential with rate μ. At time t_i, the system source code may be changed by an amount C_i; it remains unchanged elsewhere. The changed code adds faults; we assume that this is Poisson with mean proportional to C_i, say, θC_i. Because all code should undergo at least some testing, we assume that there are no source code changes beyond time t_F, $F \leq h - 1$. Then the number of faults found in the ith interval is Poisson with expectation $\lambda_i\,(1-e^{-\mu(t_i-t_{i-1})}), i = 1, \cdots, h$. The expected number of faults in the system at time t_i (after possible modification) is the expected number of faults in the system at time t_{i-1}, adjusted by the expected number found in the interval (t_{i-1}, t_i) plus the faults introduced by the changes made at time t_i:

$$\lambda_{i+1} = \lambda_i e^{-\mu(t_i-t_{i-1})} + C_i\theta \quad i = 1, \cdots, h \tag{2}$$

This model is a three-parameter death process with immigration at t_i, in which θ represents the rate at which faults occur in new and changed code, μ represents the rate at which faults are found, and λ_1 represents the number of faults in the code at the start of system test. The parameter λ_1 allows us to differentiate between the new and changed code added for the current release and the older code, which is better tested.

A referee has pointed out that this model is similar to one developed independently by van Pul [16], [17]. His sampling scheme is different, and he allows a different fault rate in the initial system only in a restricted version of his model. He develops various interesting asymptotic properties of maximum likelihood estimators. These could be useful in studying the properties of the model considered here.

As before, let $K(t)$ be the number of faults cumulative to time t, and write $\kappa(t)$ for the expected value of $K(t)$. Then $\kappa(t) = \sum_{j\leq i-1}\lambda_j e^{-\mu(t_j-t_{j-1})} + \lambda_i e^{-\mu(t-t_{i-1})}$ for $t_{i-1} \leq t \leq t_i$ and $\kappa(t) = \sum_{j\leq h}\lambda_j e^{-\mu(t_j-t_{j-1})} + \lambda_{h+1} e^{-\mu(t-t_h)}$ for $t_h \leq t$. Estimates of the parameters can be found by maximizing the likelihood function $L(\mu, \lambda_1, \theta; m_i, t_i) = \prod(\lambda_i p_i)^{m_i} e^{-\lambda_i p_i}$, where $p_i = 1 - e^{-\mu(t_i-t_{i-1})}$ and m_i is the number of faults observed in the interval (t_{i-1}, t_i). The computer code that we used to fit this model allowed a more general form of λ_i, so we used derivative-free maximization code [8].

We now give several results related to when to stop testing similar to the ones for the basic model. For these results, we use reasoning similar that used in Section IV of [4]. The stopping criterion at any time t_i, where $i > F$ is to stop if the following is true:

$$\begin{aligned}&\hat\lambda_{F+1}\hat\mu e^{-\hat\mu(t_i-t_F)} \leq f/c \quad \text{or, equivalently,}\\ &\hat\lambda_i\hat\mu e^{-\hat\mu(t_i-t_{i-1})} \leq f/c.\end{aligned} \tag{3}$$

The latter formula can also be used to suspend testing temporarily until the next code delivery, because the testing in the meantime is not economically justifiable. For example, suppose that there is no new delivery at time t_i. Then suspend testing if $\hat\lambda_i\hat\mu e^{-\hat\mu(t_i-t_{i-1})} \leq f/c$, and resume testing upon receiving the next delivery.

The number of faults remaining in the existing code at any given time t, $t_{i-1} \leq t \leq t_i$, is a Poisson random variable with mean $\lambda_i e^{-\mu(t-t_{i-1})}$, and the number of remaining faults at the end of testing is a Poisson random variable with mean $f/(\mu c)$. Thus, as in the basic model, the value of f/c may be chosen based on probabilistic rather than economic considerations in order to obtain probabilistic guarantees on the number of remaining faults. (See [2] for a more detailed discussion.) To predict the stopping time after observing the process up to time $t \leq t_F$, one needs to have precise knowledge of future changes in the code between t and t_F. If we have this knowledge, the time remaining until the optimal stopping criterion is met is given by the following:

$$(1/\hat\mu)\log(c\hat\mu\hat\lambda_{F+1}/f) + (t_F - t). \tag{4}$$

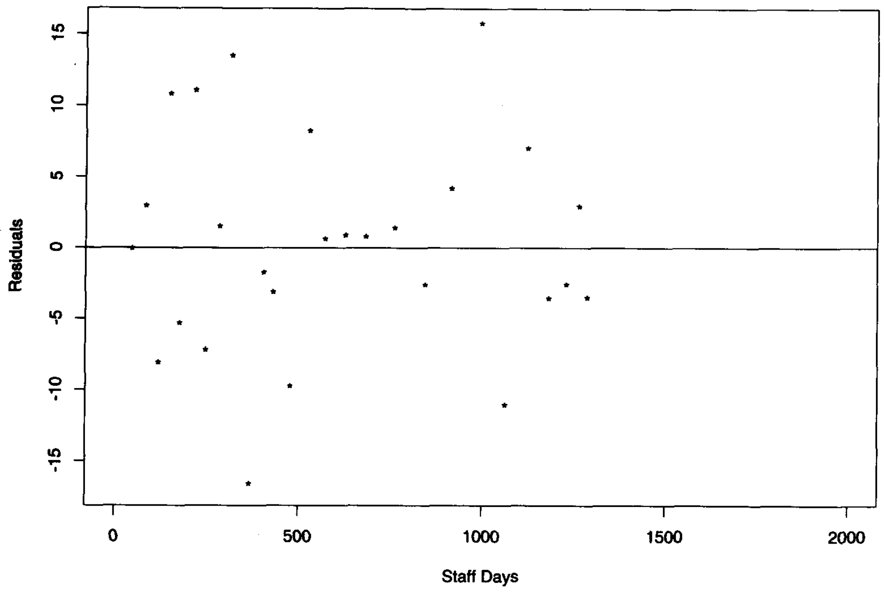


Fig. 2. Residuals vs. staff time.

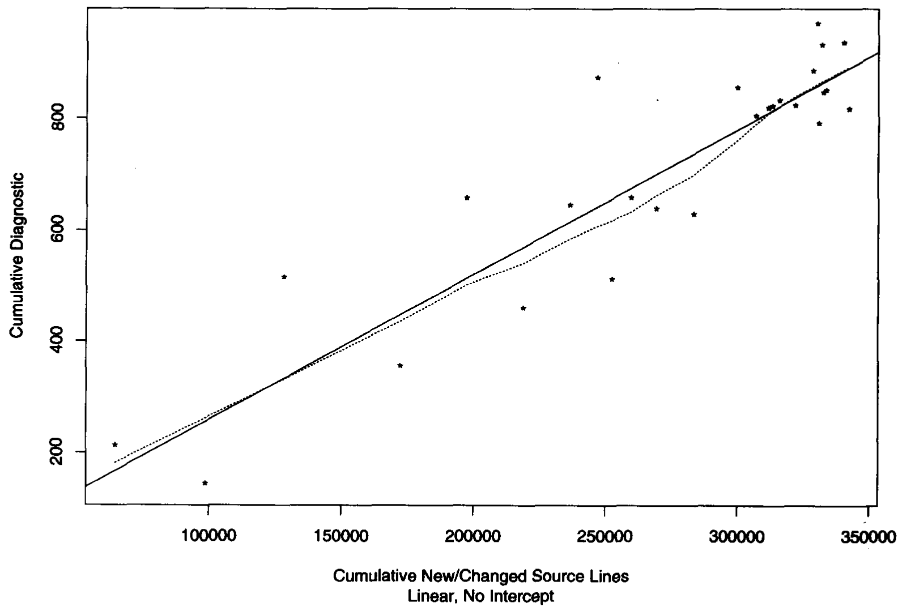


Fig. 3. Diagnostic plot.

The model may be used to estimate the net benefit of testing for a few more days in the future. Suppose that it is desired to estimate this at some future time $t \geq t_h$. Then the estimated net benefit according to the model is as follows:

$$c\hat{\kappa}(t) - ft = c\hat{\lambda}_h(1 - e^{-\hat{\mu}(t-t_h)}) - ft. \quad (5)$$

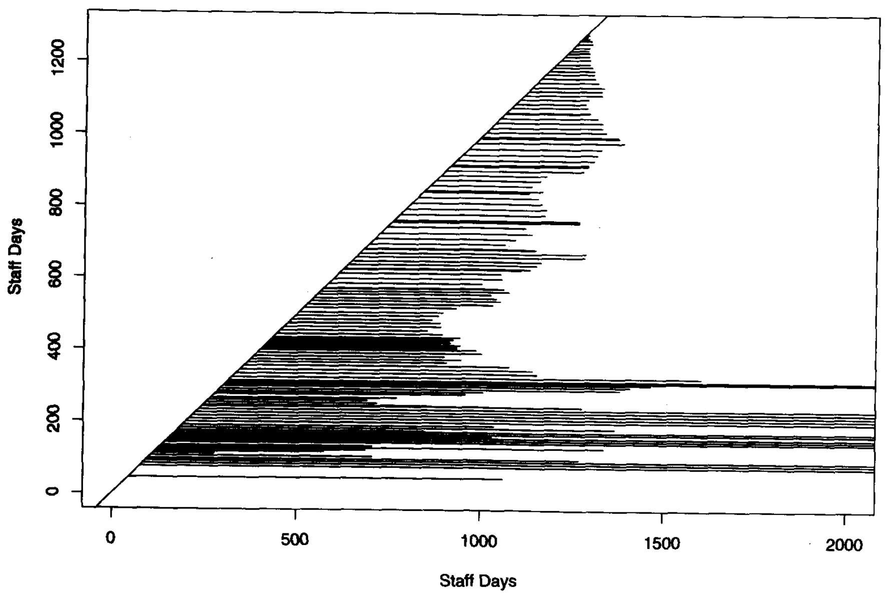
V. Plots and Discussion

For using the methodology outlined in this concise paper for deciding when to stop testing, we have devised several plots. Here we describe only a few plots in the context of System A; many more plots are described in [6].

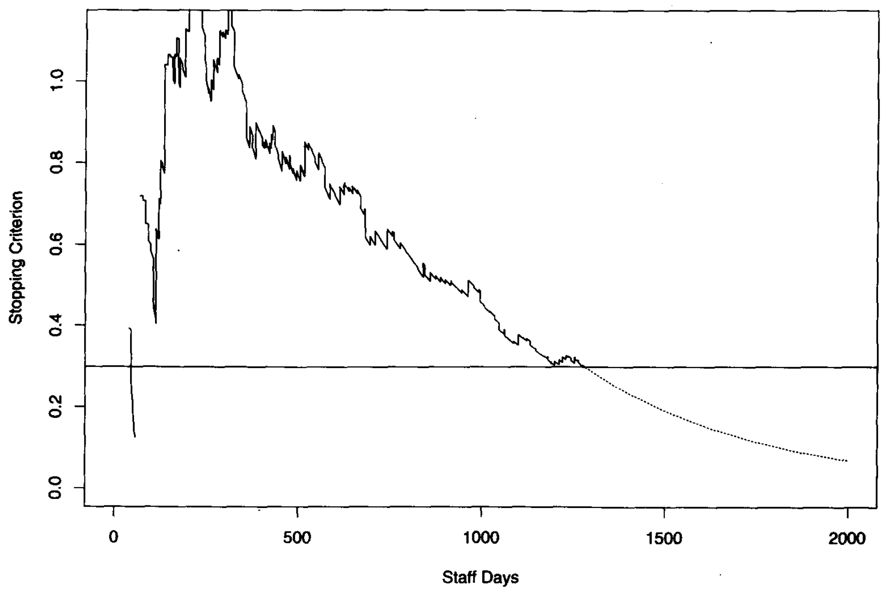
A. Model Diagnostics

First, we describe diagnostic plots for assessing the fit of our model. The first two plots are (or should be) a standard part of assessing the fit of any model. The third plot is specific to the model that we are using.

In Fig. 1, we plot $k(t)$ and $\hat{\kappa}(t)$ on the same graph. As can be seen, our model fits the data reasonably well. To explore the fit more carefully, Fig. 2 plots the residuals corresponding to new faults observed in (t_{i-1}, t_i), $(k(t_i) - k(t_{i-1}))$ $-(\hat{\kappa}(t_i) - \hat{\kappa}(t_{i-1}))$, as a function of time. There is no major trend, which suggests that the model is a reasonably good fit.



(a)



(b)

Fig. 4. (a) Estimated stopping time. (b) Progress toward stopping time.

Our third plot assesses the functional form of the C_i in the model. Equation (2) can be written more generally as $\lambda_{i+1} - \lambda_i e^{-\mu(t_i - t_{i-1})} = \theta g(C_i)$, with our model corresponding to the choice $g(x) = x$. To entertain various choices of g, we now devise a diagnostic plot. If we had estimates of the left-hand side of the above equation, we could plot them against $g(C_i)$ for the candidate g. If g is appropriate, then the plot should be approximately linear. For a given μ, an estimate of λ_i that does not depend on g is $\check{\lambda}_i = \{k(t_i) - k(t_{i-1})\}/\{1 - e^{-\mu(t_i - t_{i-1})}\}$. Since μ is unknown, we have replaced it with the maximum likelihood estimate $\hat{\mu}$. Our experience suggests that the linearity of the plot is not extremely sensitive to the use of an estimate $\hat{\mu}$ obtained from the model with g linear.

When the intervals (t_{i-1}, t_i) are short, as is the case here, the estimates $\check{\lambda}_i$ and their differences are highly variable. For this reason, we prefer to group the data weekly and plot the corresponding partial cumulative sums over time, $\sum_{j \le i}(\check{\lambda}_{j+1} - \check{\lambda}_j e^{-\hat{\mu}(t_j - t_{j-1})})$ against $\sum_{j \le i} g(C_j)$. This cumulative diagnostic plot is shown in Fig. 3. The solid line corresponds to $\sum_{j \le i} \hat{\theta} C_j$. The dotted line is a smooth curve obtained by using a nonparametric smoothing procedure [1]. The straight line and the smoothed line have only a small discrepancy. This

TABLE I
DATA FOR SYSTEM A

Cum. Staff Days	Cum. Faults	Cum. NCNCSL	Cum. Staff Days	Cum. Faults	Cum. NCNCSL	Cum. Staff Days	Cum. Faults	Cum. NCNCSL
0	0	0	211.9	98	219248	417.3	312	271175
4.8	0	16012	217	105	221355	424.9	321	272457
6	0	16012	223.5	113	223462	434.2	326	273741
6	0	16012	227	113	225568	442.7	339	275025
14.3	7	32027	227	113	225568	451.4	346	276556
22.8	7	48042	227	113	225568	456.1	347	278087
32.1	7	58854	234.1	122	227675	456.1	347	278087
41.4	7	69669	241.6	129	229784	456.1	347	278087
51.2	11	80483	250.7	141	233557	460.8	351	279618
51.2	11	80483	259.8	155	237330	466	356	281149
51.2	11	80483	268.3	166	241103	472.3	359	283592
60.6	12	91295	268.3	166	241103	476.4	362	286036
70	13	102110	268.3	166	241103	480.9	367	288480
79.9	15	112925	277.2	178	244879	480.9	367	288480
91.3	20	120367	285.5	186	247946	480.9	367	288480
97	21	127812	294.2	190	251016	486.8	374	290923
97	21	127812	295.7	190	251016	495.8	376	293367
97	21	127812	298	190	254086	505.7	380	295811
97	21	127812	298	190	254086	516	392	298254
107.7	22	135257	298	190	254086	526.2	399	300698
119.1	28	142702	305.2	195	257155	527.3	401	300698
127.6	40	150147	312.3	201	260225	527.3	401	300698
135.1	44	152806	318.2	209	260705	535.8	405	303142
135.1	44	152806	328.9	224	261188	546.3	415	304063
135.1	44	152806	334.8	231	261669	556.1	425	305009
142.8	46	155464	334.8	231	261669	568.1	440	305956
148.9	48	158123	334.8	231	261669	577.2	457	306902
156.6	52	160781	342.7	243	262889	578.3	457	306902
163.9	52	167704	350.5	252	263629	578.3	457	306902
169.7	59	174626	356.3	259	264367	587.2	467	307849
170.1	59	174626	360.6	271	265107	595.5	473	308795
170.6	59	174626	365.7	277	265845	605.6	480	309742
174.7	63	181548	365.7	277	265845	613.9	491	310688
179.6	68	188473	365.7	277	265845	621.6	496	311635
185.5	71	194626	374.9	282	266585	621.6	496	311635
194	88	200782	386.5	290	267325	621.6	496	311635
200.3	93	206937	396.5	300	268607	623.4	496	311635
200.3	93	206937	408	310	269891	636.3	502	311750
200.3	93	206937	417.3	312	271175	649.7	517	311866
207.2	97	213093	417.3	312	271175	663.9	527	312467

Cum. Staff Days	Cum. Faults	Cum. NCNCSL	Cum. Staff Days	Cum. Faults	Cum. NCNCSL	Cum. Staff Days	Cum. Faults	Cum. NCNCSL
675.1	540	313069	938.3	710	330435	1184.6	832	332615
677.4	543	313069	952	720	330263	1198.3	834	332839
677.9	544	313069	965	729	330091	1210.3	836	333053
688.4	553	313671	967.7	729	330091	1221.1	839	333267
698.1	561	314273	968.6	731	330091	1230.5	842	333481
710.5	573	314783	981.3	740	329919	1231.6	842	333481
720.9	581	315294	997	749	329747	1231.6	842	333481
731.6	584	315805	1013.9	759	330036	1240.9	844	333695
732.7	585	315805	1030.1	776	330326	1249.5	845	333909
733.6	585	315805	1044	781	330616	1262.2	849	335920
746.7	586	316316	1047	782	330616	1271.3	851	337932
761	598	316827	1047	782	330616	1279.8	854	339943
776.5	612	318476	1059.7	783	330906	1281	854	339943
793.5	621	320125	1072.6	787	331196	1281	854	339943
807.2	636	321774	1085.7	793	331486	1287.4	855	341955
811.8	639	321774	1098.4	796	331577	1295.1	859	341967
812.5	639	321774	1112.4	797	331669	1304.8	860	341979
829	648	323423	1113.5	798	331669	1305.8	865	342073
844.4	658	325072	1114.1	798	331669	1313.3	867	342168
860.5	666	326179	1128	802	331760	1314.4	867	342168
876.7	674	327286	1139.1	805	331852	1314.4	867	342168
892	679	328393	1151.4	811	331944	1320	867	342262
895.5	686	328393	1163.2	823	332167	1325.3	867	342357
895.5	686	328393	1174.3	827	332391	1330.6	870	342357
910.8	690	329500	1174.3	827	332391	1334.2	870	342358
925.1	701	330608	1174.3	827	332391	1336.7	870	342358

gives us confidence that NCNCSL should enter linearly into our model.

B. When to Stop Testing

Now we turn to the plots related to when to stop testing. We use the hypothetical values of $f = 200$ and $c = 670$ Their ratio, $r = f/c = 0.3$, is correct for this project.

Fig. 4(a) plots an estimate of when stopping will occur at intermediate points by using (4). The diagonal line represents "now," and the horizontal lines represent how far into the future testing should continue. Specifically, for a given time s on the y-axis, it shows a line segment whose left end-point is s and whose right end-point is the predicted time t, at which stopping should occur based on the faults up to day s. The quality of the prediction can be seen in the stability of the prediction from day 800 onward.

Perhaps a better way of visualizing the stopping time is to use the second equivalent version of the left-hand side of (3). In Fig. 4(b), we plot the left-hand side of (3), $z_{t_i} = \hat{\lambda}_i\hat{\mu}e^{-\hat{\mu}(t_i-t_{i-1})}$, against t_i. Any point on the curve gives the value of z_t such that if z_t were the true value of $r = f/c$, then it would be correct to stop immediately (or at least until the next source code delivery occurs). On the graph, we plot a horizontal line at $z_t = 0.3$. Whenever the curve falls below this line, one should stop testing. In our case, stopping occurred according to this stopping rule. We also plot the predicted value of z_t (the dotted line in Fig. 4(b)) to see how long we need to test.

Notice also that there is a jump in the value of z_t when there is a delivery, because of the change in the number of faults in the code. Toward the end of the testing period, because source code is s, there is a steady decline in the value of z_t. One can project the curve and decide whether it is worth continuing. When one is not sure of the exact value of r, one can compare the curve with a number of different values. One would go on testing until at least the highest value is achieved and would not test beyond the lowest value. This tradeoff between different values of r is not easy to see in Fig. 4(a). Usually, when the optimal stopping time is reached, one would continue testing for a little longer to guard against a temporary dip due to noisy data.

Another plot, well liked even by the skeptics of statistical models, is the Net Benefit Plot. Specifically, for the time period in which $K(t)$ is observed, we plot $NB(t) = ck(t) - ft$ against t. For smoothing and extrapolating, we use the expected value $\kappa(t)$. Fig. 5 plots the both the observed Net Benefit (the solid line) and expected Net Benefit obtained from (5) (the dotted line) for System A. The dashed line is the extrapolated fitted curve.

C. Fault and Fault Density Prediction

The formula for the predicted number of faults at any time t is given in the paragraph following (3). A plot of this can also help in the stopping decision. If stopping occurs according to (3), then we can make a variety of other statements. For example, for System A, $f/(c\hat{\mu}) = 145$. The predicted number of faults at the end of the test period is Poisson distributed with mean 145. Dividing this quantity by the total NCNCSL, we obtain 4.2 per 10 000 NCNCSL as an estimated field fault density. Another quantity of interest is the estimate of θ, which is approximately 0.025. This tells us that there are about 25 faults per 10 000 NCNCSL entering system test. Thus,

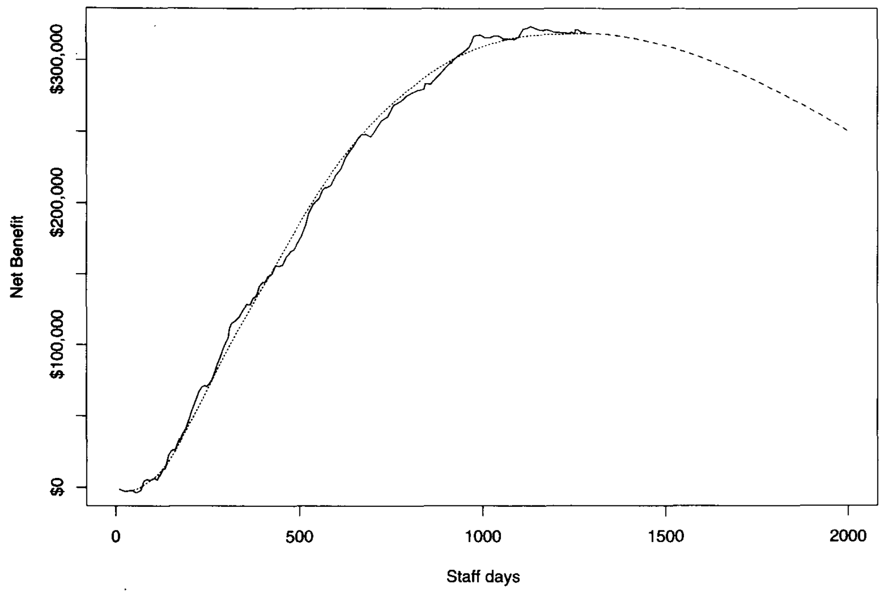


Fig. 5. Net benefit plot.

system testing was effective in that it removed 21 of every 25 faults. This raises another issue: 25 faults per 10 000 NCNCSL entering system test may be too high. It may be that something can be done upstream to reduce this number.

VI. Conclusion

In this concise paper, we have developed a framework for when to stop testing when there are additional code deliveries while testing and presented details of an actual trial. This trial has encouraged the use of reliability models and "when to stop testing" methodology in several large projects.

Acknowledgment

We thank the referees, an associate editor, Dr. Jon Kettenring and Dr. C. L. Mallows for their helpful comments.

References

[1] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *J. Am. Statist. Assoc.*, vol. 74, pp. 829–836, 1979.
[2] S. R. Dalal and C. L. Mallows, "Buying with exacting confidence," *Annals of Applied Probability*, vol. 2, pp. 752–765, Mar. 1992.
[3] ——, "Some graphical aids for deciding when to stop testing software," *IEEE J. Select. Areas Commun.*, Special Issue on Telecommunications Software Quality and Productivity, vol. 8, pp. 169–175, Feb. 1990.
[4] ——, "When should one stop testing software?" *J. Am. Statist. Assoc.*, vol. 83, pp. 872–879, 1988.
[5] ——, "When to stop testing software? Some exact and asymptotic results," in *Bayesian Analysis in Statistics and Econometrics*, P. Goel, Ed. New York: Springer-Verlag, 1990, pp. 267–276.
[6] S. R. Dalal and A. A. McIntosh, "Reliability modeling and when to stop testing for large software systems in the presence of code churn," Bellcore internal memorandum, 1992.
[7] W. Ehrlich, B. Prasanna, J. Stampfel, and J. Wu, "Determining the cost of a stop-test decision," *IEEE Software*, pp. 33–42, 1993.
[8] D. M. Gay, "Algorithm 611: Subroutines for unconstrained minimization using a model/trust-region approach," *ACM Trans. Mathematical Software*, vol. 9, no. 4, pp. 503–524, 1983.
[9] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. Reliability*, vol. R-28, pp. 206–211, 1979.
[10] Z. Jelinski and P. B. Moranda, "Software reliability research," in *Statistical Computer Performance Evaluation*, W. Frieberger, Ed. New York: Academic, 1972, pp. 465–484.
[11] W. Kremer, "Birth-death and bug counting," *IEEE Trans. Reliability*, vol. R-32, no. 1, pp. 37–47, Jan. 1983.
[12] N. Langberg and N. D. Singpurwalla, "A unification of some software reliability models," *SIAM J. Sci. Stat. Comput.*, vol. 36, no. 1, pp. 781–790, Jan. 1985.
[13] B. Littlewood, "Stochastic reliability-growth: A model for fault-removal in computer programs and hardware design," *IEEE Trans. Reliability*, vol. R-30, no. 4, pp. 313–320, Apr. 1981.
[14] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application.* New York: McGraw-Hill, 1987.
[15] J. D. Musa, "A theory of software reliability and its applications," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 312–327, 1975.
[16] M. C. J. van Pul, "Statistical Analysis of Software Reliability Models," Chapter 7, Ph.D. dissertation, Univ. of Utrecht, 1993.
[17] M. C. J. Van Pul, "Statistical models in software reliability," *Proc. 23rd Symp. Interface*, 1991, pp. 106–109.
[18] N. Singpurwalla, "Determining an optimal time interval for testing and debugging software," *IEEE Trans. Software Eng.*, vol. 17, pp. 313–319, Apr. 1991.