December 2006

Time Delay in Rectification of Faults in Software Projects

Roy Rahul
Indian Institute of Management Calcutta

Amitava Bagchi
Indian Institute of Management Calcutta

TIME DELAY IN RECTIFICATION OF FAULTS IN SOFTWARE PROJECTS

Design Science

Rahul Roy
Indian Institute of Management Calcutta
Joka, Diamond Harbour Road
Calcutta 700104, India
rahul@iimcal.ac.in

Amitava Bagchi
Indian Institute of Management Calcutta
Joka, Diamond Harbour Road
Calcutta 700104, India
bagchi@iimcal.ac.in

Abstract

Software reliability models, such as the Basic (i.e., Exponential) Model and the Logarithmic Poisson Model, make the idealizing assumption that when a failure occurs during a program run, the corresponding fault in the program code is corrected without any loss of time. In practice, it takes time to rectify a fault. This is perhaps one reason why, when the cumulative number of faults is computed using such a model and plotted against time, the fit with observed failure data is often not very close. In this paper, we show how the average delay to rectify a fault can be incorporated as a parameter in the Basic Model, changing the defining differential equation to a differential-difference equation. When this is solved, the time delay for which the fit with observed data is closest can be found. The delay need not be constant during the course of testing, but can change slowly with time, giving a yet closer fit. The pattern of variation during testing of the delay with time can be related both to the learning acquired by the testing team and to the difficulty level of the faults that remain to be discovered in the package. This is likely to prove useful to managers of software projects in the deployment of staff.

Keywords: Software reliability, system testing, Basic Model, Logarithmic Poisson Model

Introduction

A software company developing a package for release in the market needs to decide when to stop testing it for faults. Software reliability models (Pham 2003, 2000) can help the company in this task by enabling it to estimate the expected cumulative number of failures at any given time during testing. It can then infer how many faults remain in the package at the time of release. But because the models make many idealizing assumptions, the fit with observed data is often not very close. One of the assumptions frequently made is the following:

When a software failure occurs, the corresponding fault ('bug') in the package is removed without any loss of time.

In practice, after a run-time failure is reported, the testing team always takes some time to rectify the offending fault in the program code. But currently available software reliability techniques do not tell us how to determine the average time taken to rectify a fault from the failure data obtained by testers. In this paper we show how the average delay to rectify a fault can be incorporated as a parameter in a software reliability model. In the case of the Basic (or Goel-Okumoto Exponential) Model (Goel and Okumoto 1979; Musa et al. 1987), this changes the defining differential equation to a *differential-difference equation* (Bell and Cooke 1963), which can then be solved and the delay for which the fit with observed data is closest can be determined experimentally. The model can be generalized by allowing the delay to vary slowly with time during system testing. Empirical tests show that a variable delay results in a better fit with observed data. The pattern of variation of the delay with time can be related both to the learning acquired by the testing team in the course of system testing and to the difficulty level of the faults that remain to be discovered in the package.

Our objectives in this paper are as follows:

- We show how to introduce the time delay to rectify a fault as a parameter in the Basic Model, and how to solve the resulting differential-difference equation analytically. For the case when the delay varies with time, we suggest a simple enhancement of the method that yields good approximate solutions.
- We supply experimental results on available failure data confirming that our methods give a very close fit.
- We provide an explanation for the observed nature of variation of the delay with time in terms of the learning acquired by the testing team and the level of difficulty of the remaining faults in the package. Managers of software projects can make use of these observations in the deployment of staff.
- We also show how the Logarithmic Poisson Model (Musa et al. 1987) can help us to distinguish in a simple way between the cases when there is significant delay and when there is no delay in the rectification of faults.

Literature Survey

Software development, testing, and release have been important areas of study for many years. Early work in the field, described in Boehm (1981), is based largely on empirical observation. Theoretical research has produced a variety of software reliability models that try to estimate the number of faults remaining in a program at any instant during testing by applying the techniques of probabilistic analysis to statistical data on occurrences of program failures (Pham 2003, 2000). Parameter values are usually estimated using Maximum Likelihood Estimation (MLE). In the Failure Rate Models (Jelinski and Moranda 1972; Schick and Wolverton 1978), the variable of interest is the *actual* number of errors detected in the software, viewed as a discrete function of time. The initial number of errors is estimated from the given data, and the final value at the end of testing serves as a measure of the adequacy of testing. In the Non-Homogeneous Poisson Process (NHPP) Models, the variable of interest is the *expected* number of errors in the package viewed as a continuous function of time. According to Musa (1999, Sec 8.1), two models give the best estimates of software reliability. These are the Basic (or Exponential) Model (Goel and Okumoto 1979) and the Logarithmic Poisson Model (Musa et al. 1987). The former assumes that the number of faults detected during testing in a unit interval of time depends linearly on the total number of faults present in the software at that time; the latter assumes the dependence is exponential rather than linear.

The above classes of models try to determine ‘how long to test’ from an engineering perspective and typically ignore business-related costs. In subsequent work, opportunity and risk costs are introduced and the optimal release time is derived by minimizing the total cost. Dalal and Mallows (1988) take opportunity cost into account and employ a statistical approach to determine the most suitable time for the completion of testing. Ehlrich et al. (1993) apply a very similar method to find out when the testing of an actual network management system should be stopped. In Zhang and Pham (1998) and Pham and Zhang (1999), the total cost includes the costs of testing before and during the warranty period, of removing the errors that are found, and of software failure in the field.

In recent times, Jiang and Sarkar (2003) have suggested from empirical observation that software patching can help reduce the total cost of software development. This idea has been formalized and generalized by Roy and Bagchi (2004), who have examined the changes in the process of software development, testing, and release that have resulted from the adoption of Internet technology. Attempts have been made by Ji and Mukherjee (2002) and Ji et al. (2003) to apply the principles of optimal control to determine the best way to distribute the total available funding between the development and testing of software. Stikkel (2006) has tried to relate, using empirical data, the growth in the number of program failures during testing to the effort expended by the testing team in man-hours.

Problem Description and Analysis

The Basic Model with Constant Delay

In an execution of a large software package, the occurrence of failure can be viewed as a random event. Uncertainties regarding the faults that remain embedded in the program, the input data that is supplied, and other operating conditions make it infeasible to develop a dependable deterministic theory of software reliability (Jalote 1997, Chapter 9). A variety of probabilistic models are currently available to explain how the number of faults remaining in a package decreases with time during the course of system testing. Once the defining differential equation is solved, the predictions of the model can be verified by comparison with observed data. The best known such model is the Basic Model.

Let time $t = 0$ correspond to the instant when system testing begins, and time $t = T$ to the instant when system testing ends. An important problem facing software developers is to determine an appropriate value of T. Let $n(t)$ be the expected value of the number of faults remaining in the package at time t. We allow $n(t)$ to take fractional values. The Basic Model assumes that the rate of decrease of $n(t)$ with t is proportional to $n(t)$, i.e., the number of failures that occur in a short interval of time is proportional to the total number of faults that remain in the package at that time. It is also assumed that a fault is rectified as soon as it is detected. Thus, the defining differential equation is

$$\frac{dn(t)}{dt} = -\lambda n(t), \quad t \ge 0 \qquad (1)$$

where $n(0) = N_0$. The parameter λ is called the *error detection rate*; it is a small positive constant. Parameter N_0 is also a positive constant; it gives the total number of faults in the package at $t = 0$. The solution to (1) is

$$n(t) = N_0 e^{-\lambda t}, \quad t \ge 0 \qquad (2)$$

It would be more realistic to assume that it takes some time to rectify a fault. Suppose there is an average delay of Δ time units in the rectification of a fault, where Δ is a parameter, $0 \le \Delta << T$. Then (1) becomes

$$\frac{dn(t,\Delta)}{dt} = \begin{cases} 0, & 0 \le t \le \Delta \\ -\lambda n(t-\Delta), & t > \Delta \end{cases} \qquad (3)$$

Here, too, $n(0,\Delta) = N_0$, whatever be the value of Δ. (3) is a *differential-difference equation* and says that the rate at which n changes at time t is proportional to the value that n had Δ time units earlier. We can rewrite (3) in the form

$$\frac{dn(t,\Delta)}{dt} = \begin{cases} 0 & 0 \le t \le \Delta \\ -\lambda N_0 & \Delta < t \le 2\Delta \\ -\lambda n(t-\Delta) & t > 2\Delta \end{cases} \qquad (4)$$

From (4) we get
$$n(t,\Delta) = \begin{cases} N_0 & 0 \le t \le \Delta \\ N_0[1-\lambda(t-\Delta)] & \Delta \le t \le 2\Delta \end{cases} \qquad (5)$$

Proceeding in this way, making use of (4) and generalizing, we get

Theorem 1: For $0 \le t \le T$ and $r\Delta < t \le (r+1)\Delta$, $r \ge 0$,
$$n(t,\Delta) = N_0 \sum_{s=0}^{r} (-1)^s \frac{\lambda^s}{s!} (t - s\Delta)^s \qquad (6)$$

Proof: By induction on t, *i.e.*, on r. A similar differential-difference equation is solved in (Bellman and Cooke 1963, p 43). □

Theorem 1 expresses $n(t, \Delta)$ as a finite series. Although this series does not have a convenient closed form, $n(t, \Delta)$ can be readily computed by machine given t and Δ when N_0 and λ are known. The expression for $n(t, \Delta)$ reduces to (2) when $\Delta \to 0$. This allows us to express $n(t)$ as $n(t, 0)$. Note that $n(t, \Delta)$ is always non-increasing in t. We will name this reliability model the CDB (Constant Delay Basic) Model.

The Basic Model with Variable Delay

In the CDB Model, Δ is the average delay in rectifying a fault over the period of system testing. Thus Δ is viewed as a constant during the time interval $[0,T]$. In reality, Δ varies with time, as a result of the changes in the types and difficulty levels of the faults that still remain in the package, and also the greater familiarity with the package acquired by the members of the testing team, making them quicker at locating and correcting faults. One way to generalize the CDB Model is to break up the interval of length T into, say, three sub-intervals $[0, t_1]$, $(t_1, t_1 + t_2]$ and $(t_1 + t_2, T]$, where $t_1 + t_2 + t_3 = T$. Let us suppose that the average value of Δ is Δ_1 in the interval $[0, t_1]$, Δ_2 in the interval $(t_1, t_1 + t_2]$, and Δ_3 in the interval $(t_1 + t_2, T]$, where $0 \le \Delta_1 << t_1$, $0 \le \Delta_2 << t_2$, and $0 \le \Delta_3 << t_3$. When Δ is no longer a constant, it is difficult to solve equation (3) analytically. The following approximate technique yields good solutions in this case:

Use equation (6) to compute $n(t, \Delta)$ in the time interval $0 \leq t \leq T$, setting $\Delta = \Delta_1$ when $0 \leq t \leq t_1$, $\Delta = \Delta_2$ when $t_1 < t \leq t_1 + t_2$, and $\Delta = \Delta_3$ when $t_1 + t_2 < t \leq T$. Owing to the abrupt changes in the value of Δ at t_1 and $t_1 + t_2$, this computation can sometimes force a step *increase* in the value of $n(t, \Delta)$ at those two instants. Since in practice n must be non-increasing in t, when computing $n(t, \Delta)$, take the minimum of the currently computed value from equation (6) and the already computed value at the immediately preceding instant.

We name this model the TDB (Three Delay Basic) Model. A much closer fit with observed data is obtained when three different values of Δ are used instead of just one. When the number of different Δ values is increased further, say to five, the fit becomes even closer. This idea can be generalized by allowing Δ to become a *slowly varying* function of time t. Then, when $n(t, \Delta)$ is computed, the value of Δ at time t is used in equation (6); it should be ensured that $n(t, \Delta)$ remains non-increasing in t as explained above. We give the name VDB (Variable Delay Basic) Model to the generalized version of the TDB Model.

Delay in the Logarithmic Poisson Model

In this model, in place of equation (1) the following equation applies: $\dfrac{dn(t)}{dt} = -\lambda \exp(\sigma n(t)),\ t \geq 0 \qquad (7)$

where 'exp' denotes exponentiation, and the parameters λ and σ are small positive constants. Thus the rate of decrease of $n(t)$ is exponential rather than linear in $n(t)$. Substituting $q(t) = \exp(-\sigma n(t))$ we get $\dfrac{dq(t)}{dt} = \lambda\sigma, t \geq 0$.

Thus $q(t)$ when plotted against t is a straight line with positive slope, and its value increases with time. When there is a time delay of Δ in rectifying a fault, equation (7) changes to

$$\frac{dn(t,\Delta)}{dt} = \begin{cases} 0, & 0 \leq t \leq \Delta \\ -\lambda \exp(\sigma(t-\Delta)), & t > \Delta \end{cases} \qquad (8)$$

and making the substitution $q(t,\Delta) = \exp(-\sigma n(t,\Delta))$ we get

$$\frac{dq(t,\Delta)}{dt} = \begin{cases} 0, & 0 \leq t \leq \Delta \\ \dfrac{\lambda\sigma q(t,\Delta)}{q(t-\Delta,\Delta)}, & t > \Delta \end{cases} \qquad (9).$$

Since $q(t)$ is increasing in t, we expect $q(t, \Delta)$ to also increase with t. By an examination of equation (9) we conclude that, assuming a fixed Δ, q when plotted against t is no longer a straight line; while its slope is always positive, the slope is greater than $\lambda\sigma$ at small values of t and decreases asymptotically to $\lambda\sigma$ as t increases.

Experimental Results

When failure data for a package is available, we know the observed cumulative number of failures for the package at different time instants over the system testing phase. This is an integer series $OF(t_1)$, $OF(t_2)$, . . ., $OF(t_k)$, . . ., where OF stands for 'Observed Failures', and t_1, t_2, . . . , t_k, . . . , are the time instants of the observations. To check how closely a reliability model fits the observed data, the experimental steps are as follows (see Pham 2000, p 134):

Step 1: Estimate the values of the model parameters, such as N_0 and λ for the Basic Model, by the application of statistical techniques like Maximum Likelihood Estimation (MLE) on the given set of observed data (Pham 2000, pp 106-107).

Step 2: Using the model, compute the cumulative number of failures at the same time instants at which observations were made. These computed values form a series of real numbers $CF(t_1)$, $CF(t_2)$, . . ., $CF(t_k)$, . . ., where CF stands for 'Computed Failures'. (Note that $CF(t) = N_0 - n(t)$ for the Basic Model. Similarly, $CF(t) = N_0 - n(t, \Delta)$ for the CDT Model, and so on.)

Step 3: Compute the Sum of Squared Errors (SSE) for the model:

$$SSE = \{ CF(t_1) - OF(t_1) \}^2 + \{ CF(t_2) - OF(t_2) \}^2 + \ldots + \{ CF(t_k) - OF(t_k) \}^2 + \ldots$$

Among two models, the one that has a smaller SSE value fits the observed data more closely.

Example 1

Consider the failure data set given in (Pham 2000, Table 5.7, p 140) for a program that monitors a real-time control system and consists of about 200,000 lines of code in a high-level language. A record was maintained on a daily basis of the cumulative number of failures over a period of 111 days during the system testing phase. We selected this example for study because the failure data is fairly extensive. For this data set, Pham provides parameter values and experimental results for the Basic Model and some other related models. The *SSE* values we obtained show the same trend as Pham's for the models studied by him. In Table 1, we compare the Basic Model and the best three of the other models with the CDB, TDB, and VDB Models described above; for these three models, we used the parameter values $N_0 = 483.0$, $\lambda = 0.0311$, which gave better experimental results than the parameters for the Basic Model supplied by Pham, although Pham's values also gave good results. The issue of parameter estimation is discussed in a subsection below.

Table 1 shows that when an average delay of $\Delta = 5.75$ is used in equation (6), the *SSE* is less than half that for the Basic Model. There is a further sharp drop in the *SSE* when T is split into three time intervals and different delay

Table 1: SSE Values for Reliability Models (Example 1)

Model	Basic (Goel-Okumoto)	Inflection S-Shaped	Pham-Nordman	Pham-Zhang	CDB	TDB	VDB
Parameter Values	$N_0 = 497.282$ $\lambda = 0.0308$	As in (Pham 2000, p 141)			$N_0 = 483.0$ $\lambda = 0.0311$		
Time Intervals	-	-			-	$t_1 = 28$ $t_2 = 37$ $t_3 = 46$	As in Figure 2
Delay	-	-			5.75	$\Delta_1 = 4.7$ $\Delta_2 = 13.8$ $\Delta_3 = 11.3$	As in Figure 2
SSE	109672	32541	34282	32541	45529	13977	7131

values are assigned to the intervals; note that for this example, $T = t_1 + t_2 + t_3 = 111$. When Δ is allowed to vary slowly and continuously during the system testing phase, the *SSE* drops even further. Figure 1 shows how close the computed data values for the VDB Model are to the actual observations when the delay varies as shown in Figure 2. Our comments on the experimental results are as follows:

a) The Inflection S-Shaped, Pham-Nordman, and Pham-Zhang Models all give a much better fit with observed data than the Basic Model.
b) The CDB Model yields a much lower *SSE* value than the Basic Model, but is not as good as the above three models studied by Pham.
c) Both the TDB and VDB Models give much lower *SSE* values than any of the other models.

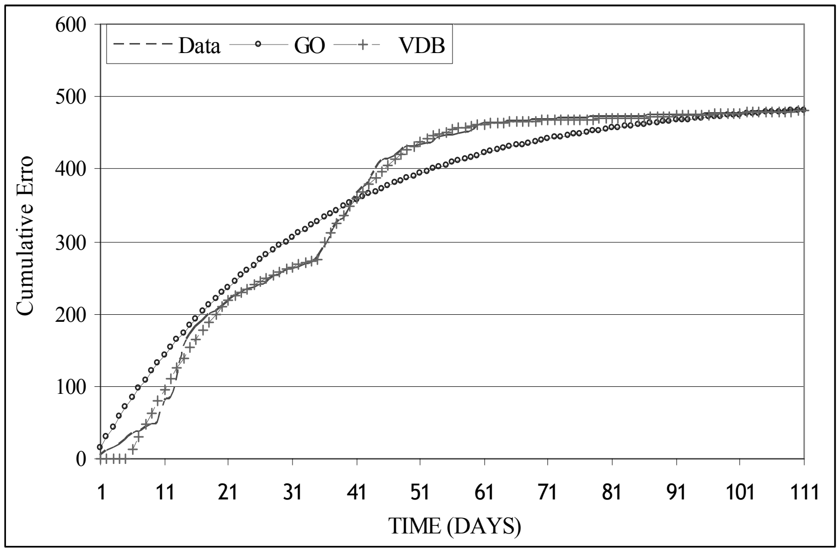

Figure 1: The Basic Model, the VDB Model, and Observed Data (Example 1)

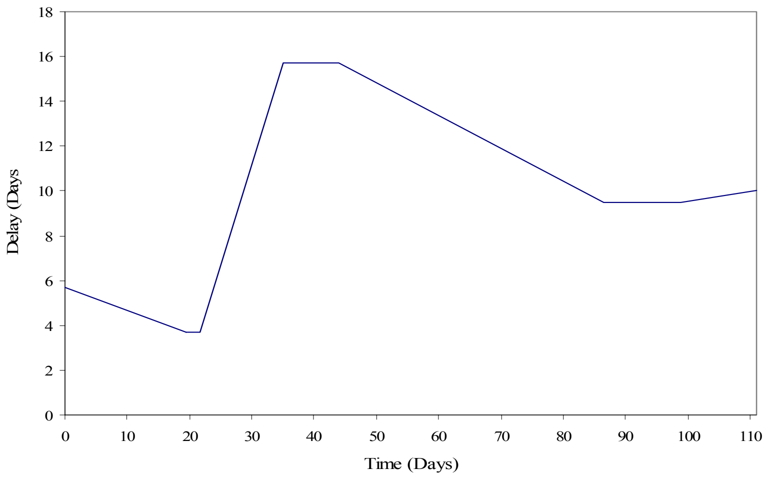

Figure 2: Variation of Delay Δ with Time during System Testing (Example 1)

Our experiments show that the incorporation of a delay parameter in the Basic Model brings the computed failure curve much closer to observed data, and a delay that varies continuously and slowly with time gives a much closer fit than a constant delay. The shape of the delay curve in Figure 2 is of interest because the *SSE* is so low. The curve has seven straight line segments. In the first segment, Δ decreases from 5.7 to 3.7, and in the second segment, it remains constant for a short interval of time. In the third segment, it increases sharply from 3.7 to 15.7. It again remains constant for some time, decreases to 9.5, remains constant for some time, and then increases slightly to 10.0. The exact values taken by Δ will change when the model parameters are changed.

We offer the following interpretation for the experimentally determined shape of the delay curve. Initially Δ decreases as the testing team becomes familiar with the package. In the first two segments, the simpler faults are found. When only the harder faults remain, the delay increases sharply. In course of time, the testing team becomes quicker at rectifying the harder faults. At the end, only the very hard faults remain to be fixed and the delay again increases. Thus, the testing team undergoes two separate phases of learning, initially in the first segment and later in the fifth segment. Such explanations are likely to appeal to managers of software projects, who can make use of the phenomenon in the deployment of staff. Additional experiments on other data sets are needed to corroborate these conclusions.

One complicating factor has been ignored in the above discussion. If it is assumed that as soon as a failure is reported, the corresponding fault is rectified, then the same fault in the program code cannot cause more than one failure. But if it takes time to rectify a fault, then a fault that causes a failure can cause another failure before it is fixed. In this situation, the number of program failures can exceed the number of distinct faults discovered. Since equations (1) and (3) are formulated in terms of faults, but the observed data relates to failures, this causes a certain discrepancy between theory and observation, but it seems that the error introduced is not very large (Jalote 1997, p 464). To get around the problem, duplicate failures should not be counted when reporting observations, but with published data, it is not certain this was actually done. This issue requires further investigation (Musa 1999, pp 193-196).

Example 2

We now provide a contrasting example in which delay in fault rectification plays no role. Consider the data set given in Table P.5.2 of (Pham 2000, p 156), originally obtained from (Musa et al.. 1987), which shows the successive inter-failure execution times (in seconds) during system testing for a real-time command and control system. Musa reported that whenever a failure occurred, program execution was stopped and the corresponding fault was fixed before resuming execution. So the time to rectify faults does not enter into the reported data.

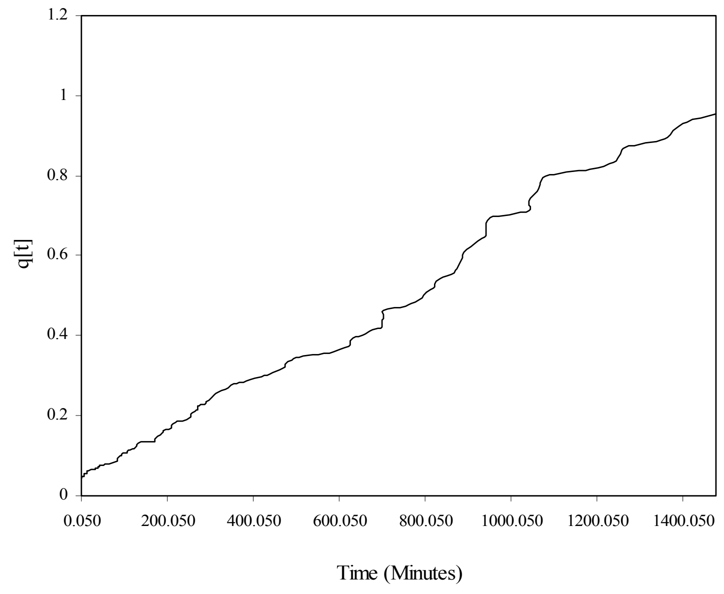

Figure 3: Plot of *q*(*t*) against *t* (Log Poisson Model, Example 2)

This data set spans a large interval of time in seconds, so we converted the time unit to minutes, and used a time interval of 1481 minutes for our experiments. Since the data records 136 failures, the value of N_0 in the Basic Model should not be less than 136, but the smallest *SSE* of 52898 we found was obtained when we took $N_0 = 135.0$ and $\lambda = 0.00208$. The CDB Model gave the lowest *SSE* value for $\Delta = 0$, which verified Musa's claim that fault rectification times had been eliminated. But because the *SSE* was rather high, we also tried to fit the Logarithmic Poisson Model to the data. A very good fit was obtained when we chose $N_0 = 138.0$, $\lambda = 0.000453$ and $\sigma = 0.02264$. This gave an *SSE* of only 999. When $q(t)$ was plotted against t (*i.e.*, at the points of observation, see Figure 3), the straight line fitted by MS Excel gave an R^2 value in excess of 0.99, confirming that there was no delay in fault rectification. In case of Example 1, when the Logarithmic Poisson Model was applied to the data and $q(t)$ was plotted against t (see Figure 4), MS Excel gave a much smaller R^2 value, implying that there was positive delay in the error correction process; as is visually apparent, the plot of $q(t)$ against t is not a straight line in this case.

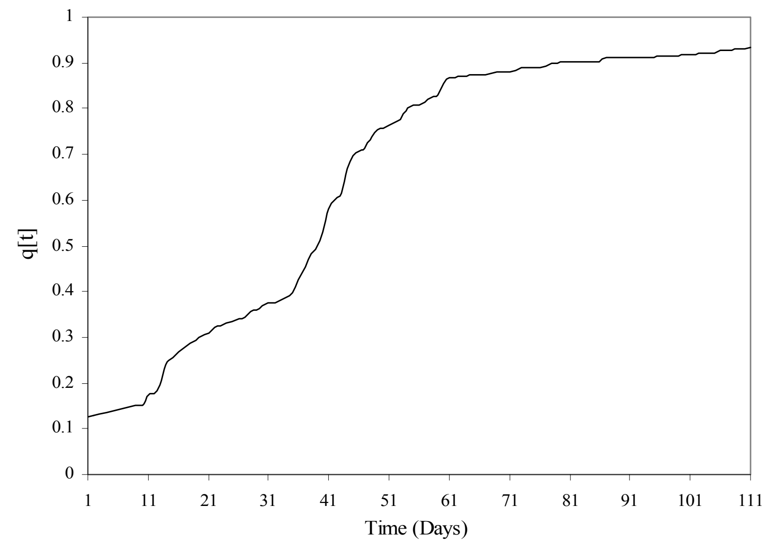

Figure 4: Plot of *q*(*t*) against *t* (Log Poisson Model, Example 1)

Parameter Estimation

To derive good estimates of the values of the parameters λ and N_0 that play such an important role in the Basic Model, we need to know the probability distribution of the incremental change $OF(t_{k+1})$ - $OF(t_k)$ in the observed values in terms of the incremental change $CF(t_{k+1})$ - $CF(t_k)$ in the corresponding computed values. There is some evidence to believe that this distribution is normal for large sample sizes (Pham 2000, p 106). Assuming that is so, we can fit the experimental data to the normal curve using the techniques of Maximum Likelihood Estimation (MLE), derive the statistical parameters of the normal distribution, and then compute λ and N_0 in their terms. When a third parameter Δ is introduced as in the CDB Model, the type of the probability distribution is not really known. One way to estimate the model parameters in this case is to start with the parameters for the Basic Model and undertake a systematic search of the neighbourhood to determine the point in the parameter space at which the fit of the model with the observed data is closest. An initial guess of an appropriate value (or values) of Δ is needed to start the process. This was the method we adopted in the examples for all the three models CDB, TDB, and VDB. We are currently trying to formalize and refine the method to make it easier for program testers to estimate the values of λ, $N_{0,}$ and Δ in practice.

Conclusion

Software reliability models such as the Basic Model generally make the simplifying but unrealistic assumption that when a failure occurs at run time in a package during the system testing phase, the corresponding fault is rectified instantaneously. This paper proposes a modification of the Basic Model in which the average time Δ to rectify a fault is incorporated as a parameter. It then becomes possible, by trying out different values of Δ experimentally, to find the value at which the fit with observed data is closest. By allowing a slow variation of Δ with time during system testing, an even better fit is realized.

This study is at a preliminary stage. Various issues require further investigation:

a) In our experimental work we adopted SSE as the measure for determining the closeness of fit of the experimental data with the values computed from the theoretical models. Are there any other measures that are convenient to use and give equally good results? If an alternative measure can be identified, then the results obtained from the SSE measure can be corroborated with its help.

b) To form a clearer idea about the role of Δ in equation (6), the mathematical properties of the finite series in the RHS of (6) needs to be studied more closely. For example, we might want to know the exact nature of variation of $n(t, \Delta)$ with t and with Δ. Owing to the complicated form of equation (6), it is not easy to tell how $n(t, \Delta)$ changes when t or Δ is varied.

c) Improved analytical techniques are needed for estimating the parameters N_0, λ and Δ for the CDB Model, and for determining the nature of variation of Δ with time for the TDB and VDB Models.

d) The CDB, TDB, and VDB Models have already been tested on other failure data sets taken from (Pham 2000) and (Stikkel 2006) with good results. But more extensive experimentation is desirable to corroborate the findings reported here.

References

1. Bellman R and Cooke K L, *Differential-Difference Equations*, Academic Press, 1963.
2. Boehm B W, *Software Engineering Economics*, Prentice Hall, 1981.
3. Dalal S R and Mallows C L, When Should One Stop Testing Software? *Journal of the American Statistical Association*, vol 83, 1988, pp 872-879.
4. Ehlrich W, Prasanna B, Stamfel J and Wu J, Determining the Cost of a Stop-Test Decision, *IEEE Software*, March, vol 33, 1993, pp 33-42.
5. Goel A L and Okumoto K, Time Dependent Error-Detection Rate Model for Software and Other Performance Measures, *IEEE Transaction on Reliability*, vol R-28(3), 1979, pp 206-211.
6. Jalote P, *An Integrated Approach to Software Engineering* (2nd Ed), Narosa, 1997.
7. Jelinski Z and Moranda P R, Software Reliability Research, in *Statistical Computer Performance Evaluation*, W Freiberger (ed.), Academic Press, New York, 1972.
8. Ji Y, Mookerjee V and Sethi S, An Integrated Planning Model of System Development and Release, *Proc WITS-2003*, 2003, pp 55-60.
9. Ji Y and Mookerjee V, Optimal Software Development and Debugging Policies: A Control Theoretic Approach, *Proc WITS-2002*, 2002, pp 163-168.
10. Jiang Z and Sarkar S, Optimal Software Release Time with Patching Considered, *Proc WITS-2003*, 2003, pp 61-66.
11. Musa J D, *Software Reliability Engineering*, McGraw-Hill., 1999.
12. Musa J D, Iannino A and Okumoto K, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1987.
13. Pham H, Software Reliability and Cost Models: Perspectives, Comparison, and Practice, *European Journal of Operational Research*, vol 149, 2003, pp 475-489.
14. Pham H, *Software Reliability*, Springer, Singapore, 2000.
15. Pham H and Zhang X, A Software Cost Model with Warranty and Risk Costs, *IEEE Transactions on Computers*, vol 48, 1999, pp 71-75.
16. Roy R and Bagchi A, The Development, Testing and Release of Software Systems in the Internet Age: A Generalized Analytical Model, *Proc ICIS-2004*, 25th International Conference on Information Systems, Washington DC, 2004, pp 399-412.
17. Schick G J and Wolverton R W, An Analysis of Competing Software Reliability Models, *IEEE Transactions on Software Engineering*, vol SE-4, 1978, pp 102-120.
18. Stikkel G., Dynamic model for the system testing process, *Information and Software Technology*, vol 48(7), July 2006, pp 578 – 585.
19. Zhang X and Pham H, A Software Cost Model with Warranty Cost, Error Removal Times and Risk Costs, *IIE Transactions*, vol 30, 1998, pp 1135-1142.