December 2004

The Development, Testing, and Release of Software Systems in the Internet Age: A Generalized Analytical Model

Rahul Roy
Indian Institute of Management

Amitava Bagchi
Indian Institute of Management

THE DEVELOPMENT, TESTING, AND RELEASE OF SOFTWARE SYSTEMS IN THE INTERNET AGE: A GENERALIZED ANALYTICAL MODEL

Rahul Roy and Amitava Bagchi
Indian Institute of Management Calcutta
Calcutta, India
rahul@iimcal.ac.in **bagchi@iimcal.ac.in**

Abstract

A major issue in the production of software by a software company is the estimation of the total expenditure likely to be incurred in developing, testing, and debugging a new package or product. If the development cost and development schedule are assumed known, then the major cost factors are the testing cost, the risk cost for the errors that remain in the software at the end of testing, and the opportunity cost. The control parameters are the times at which testing begins and ends, and the time at which the package is released in the market (or the product is supplied to the customer). By adjusting the values of these parameters, the total expenditure can be minimized. Internet technology makes it possible to provide software patches, and this encourages early release. Here we examine the major cost factors and derive a canonical expression for the minimum total expenditure. We show analytically that when the minimum is achieved (1) testing will continue beyond the time of release and (2) the number of software errors in the package when testing ends will be a constant (i.e., the package will have a guaranteed reliability). We apply the model to a few special scenarios of interest and derive their properties. It is shown that the incorporation, as a separate item, of the cost incurred to fix the errors discovered during testing has only a marginal effect on the canonical expression derived earlier.

Keywords: Software reliability, software patching, risk cost, opportunity cost

Introduction

A software company designs and builds software packages and sells them in the market. Sometimes it also develops special-purpose software products for select customers. In either case, the lifetime of a software package (or product) consists of three phases, which can partly overlap in time:

- Phase I: Development: Systems analysis, design, coding and unit testing; this job is performed by the development team
- Phase II: Testing: Integration and system testing; this job is performed by the testing team
- Phase III: Field Use: Release of the package in the market (or the delivery of the product to a customer) and its use in the field

A package typically consists of a number of modules. It is the responsibility of the development team to ensure that the individual modules themselves are free of internal errors (bugs). Complete harmony between the modules cannot be achieved in the development phase, and when the modules are knit together to form a package, software errors caused by inter-module interactions and incompatibilities creep into the package. The addition of new modules increases the number of such bugs at a faster and faster rate. Testing can start as soon as enough modules are ready, which can occur before development ends. The testing team tries to discover and eradicate as many bugs as possible prior to the release of the package in the market. In the ideal situation, all bugs would be totally eliminated during testing, but in practice such total elimination is not possible. A high cost is incurred for every bug that is discovered in the field, so the cost to the company goes up if testing ends too early. If testing ends too late, not only

does the cost of testing rise, the late entry into the software market also increases the opportunity cost. This makes the choice of the date of release a critical decision for the software company.

The price that must be paid for inadequate testing can be high. A study conducted by the National Institute for Standards and Technology of the U.S. Department of Commerce has found that software bugs cost the U.S. economy an estimated $59.5 billion annually (RTI 2002). Furthermore, the study has also found that although all errors cannot be removed, more than a third of the cost, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure that enables earlier and more effective identification and removal of software defects. Thus, while early release is desirable since it reduces the opportunity cost, testing should not end too soon. In most studies on software development there is an implicit assumption that testing ends at the time of release, but as has been pointed out recently by Jiang and Sarkar (2003), that need not be the case.

The economics of software development, testing, and release has been an important area of study for many years. Early work in the field, described in the pioneering and influential study by Boehm (1981), is based largely on empirical observation. Theoretical research has produced two related classes of software reliability models. Both classes assume that statistical data is available on occurrences of failures, and make use of the theory of maximum likelihood estimation to estimate the values of parameters. The first class consists of models (Jelinski and Moranda 1972; Schick and Wolverton 1978) in which 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. The second class of models consists of non-homogenous Poisson process (NHPP) models (Goel and Okumoto 1979; Musa 1999, Sectio 8.1), in which the variable of interest is the *expected* number of errors detected viewed as a continuous function of time. It is assumed that the number of errors detected (and fixed) in a small interval of time is proportional to the number of errors that still remain in the software at that time. The goal is to find a function that correctly computes the expected number of errors in the software at any given instant of time.

Both the above classes of models try to determine how long to test from a purely engineering perspective, and typically ignore business related costs. In subsequent research (Dalal and Mallows 1988; Ehlrich et al. 1993; Pham and Zhang 1999; Zhang and Pham 1998; Zheng 2002), testing and other costs are introduced, and the optimal release time is derived by minimizing the total cost. Dalal and Mallows (1988), for example, use an economic reward model to arrive at an optimal rule for test completion. Ehlrich et al. (1993) convert this into a cost model, and introduce the notions of cost of failure recovery in the field and the loss to the customer on account of field failure. Zhang and Pham (1998) and Pham and Zhang (1999) express the total cost of testing as a function of the testing time from the onset of testing to the release of the software. The total cost includes the costs of testing both before and during the warranty period, of removing the errors that get discovered, and of software failure in the field.

Jiang and Sarkar (2003) have recently shown that software patching can reduce the total cost of software development. They permit a software package or product to be released before the optimal release time as determined using traditional reliability models. Instead of terminating the testing of the software at its time of release, testing is continued for some more time, and fixes for all bugs found by users and by the testing team are made available as software patches at the company website. Their model incorporates the costs of testing, of fixing bugs found by users during the period of joint testing, of fixing bugs found later in the field, and the opportunity cost. They do not supply an analytical solution, but illustrate with the help of a numerical example that their model is capable of achieving a significantly lower total cost than earlier models that force the completion of testing to coincide with the release of the software. Their model is likely to prove attractive to software producers, who are eager to exploit the tremendous technical opportunities made available by the Internet, but at the same time are unwilling to commit large resources at the initial stages of a project in view of the inherent business risks (Iansiti and McCormack 1997). The model encourages joint testing by developers and users and thereby allows software producers to understand the market more quickly. In two related studies, Ji and Mookerjee (2002) and Ji et al. (2003) provide integrated planning models for a situation where development and testing take place concurrently rather than sequentially, and where higher software quality is traded off against longer project duration. They apply optimal control theory to determine the relative proportions of funding (or programming effort) that should be funneled into development and testing at any instant of time.

In this paper we propose an integrated model of system development and testing that takes software patching into account. It generalizes the work of Jiang and Sarkar by allowing testing to begin before development ends. It also provides analytical results and thereby strengthens the theoretical foundation of the approach. In our proposed NHPP model, the expected number of errors detected per unit of time is not constant. Once the development phase is over, it decreases steadily up to the end of a project. The paper is organized as follows. The next section describes the model within the context of the current scene of software development and derives some general theoretical results. Specialized scenarios of interest are then discussed. A small numerical example is provided to illustrate that the formulae introduced give correct solutions. The number of bugs discovered by testing and the cost of fixing them is examined more closely. The concluding section summarizes the paper and enumerates some open problems.

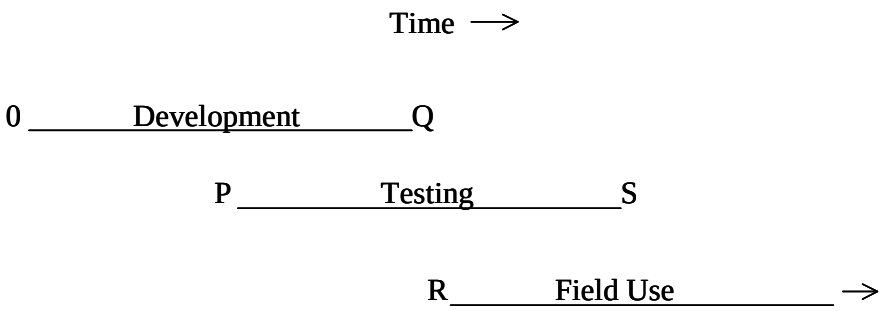

Figure 1. Milestones in the Lifetime of a Package or Product

The General Model

As has been explained, the life of a package (or product) has three phases: the development phase, the testing phase, and field use. Thus the lifetime of a package can be characterized in terms of the following five events or milestones (see Figure 1).

(1) The time at which the development phase starts; let us say this occurs at time $t = 0$.
(2) The time $t = P$ at which the testing phase starts.
(3) The time $t = Q$ at which the development phase ends.
(4) The time $t = R$ at which field use starts; this is the time at which the package is released in the market (or, in the case of a product, supplied to a customer).
(5) The time $t = S$ at which the testing phase ends.

In the most general scenario, $0 \le P \le Q \le R \le S$. This allows testing to begin before development ends. It also allows the software to be released before testing is completed. Let $n(t)$ be the number of bugs in the package when it has been run for time $t \ge 0$. For each run of the package, this number depends in a complex way on the input data that is supplied. So for each $t > 0$, $n(t)$ is best viewed as an integer-valued random variable with a probability distribution (Jalote 1997, Section 9.6), which means that $\{ n(t) \mid t \ge 0 \}$ defines a stochastic process. To simplify the analysis, we follow the lead of Goel and Okumoto (1979) and Musa (1999, Section 8.1), and identify $n(t)$ with the *expected value* of the probability distribution at time t. This makes the model deterministic and allows $n(t)$ to take fractional values. Then $n(t)$ is governed by the following system of differential equations (for brevity we sometimes write n instead of $n(t)$ below):

$$\frac{dn}{dt} = \begin{cases} f(t,n) & 0 \le t < P \quad (1) \\ f(t,n) - \lambda n & P \le t < Q \quad (2) \\ -\lambda n & Q \le t < R \quad (3) \\ -(\lambda + \rho)n & R \le t < S \quad (4) \end{cases}$$

The meanings of the symbols are explained below. We have assumed in (1) through (4) that $0 < P < Q < R < S$, i.e., the above five events are distinct. It is possible for P to equal Q and/or R to equal S in specific cases. Some such scenarios are discussed later. We restrict ourselves to situations in which the above system of differential equations has solutions in closed form. The initial condition is $n(0) = 0$. Note that (3) and (4) are first order linear differential equations with constant coefficients (Ross 1989). When $f(t,n)$ is a constant or a linear function of n, (1) and (2) also become first order linear differential equations.

The first equation says that initially the number of bugs grows with time as some function $f(t,n)$ of t and $n(t)$. In the simplest case, $f(t,n) = \mu$, where μ is a positive constant, implying that the number of bugs is linear in the length of the package as measured by the number of instructions, under the simplistic assumption that this length is linearly proportional to t. But more generally we might expect $n(t)$ to grow with t at a rate that is faster than linear, because the addition of new modules to the package causes inter-module interactions and dependencies to grow in a nonlinear manner. We require $f(t,n)$ to be continuous in both t and $n(t)$; $n(t)$ is continuous at all time instants, but dn/dt can be discontinuous at the time instants P, Q, and R.

At time $t = P$, integration and system testing begin. During the period $P \leq t < Q$, the development team is still at work, but since some of the modules are ready, testing has already commenced. This combined effect is reflected in (2). Bugs keep getting generated at the rate of $f(t,n)$ per unit of time, but bugs also get removed at the rate of $\lambda n(t)$ per unit of time. The parameter λ is called the *error detection rate*; it is a positive real number such that $\lambda n(t) < f(t,n)$. We are assuming here that (1) bugs are independent and identically distributed, and have an identical effect on software failure; (2) the number of bugs that get discovered (and removed) per unit of time by the testing team is proportional to the number of bugs already present in the package at that time; (3) a bug gets removed as soon as it is discovered. Most prior work in this field also makes the simplifying assumption that the process of removal of bugs does not create new bugs. We have also done so above, but this condition can be relaxed to a limited extent as explained below.

Development ends at time $t = Q$. After Q, no new bugs get added to the package. At time $t = R$, the package is released to the public. If $S > R$, testing continues beyond release, and bug fixes are made available from time to time as software patches. Patching is a low cost activity so its cost can be ignored. After R, the error detection rate increases to $(\lambda + \rho)$. The parameter ρ is the rate of discovery of bugs by the community of users of the package and is a positive real number. We expect ρ to be less than λ since users are unlikely to be as efficient in discovering bugs as the testing team. Since the expected number of errors that get detected per unit of time is not the same throughout the duration of the project, but is time dependent, this is an NHPP model. In summary, then, our assumptions are very similar to those made by Goel and Okumoto and later by Jiang and Sarkar. The only major departure our model makes from earlier conventions is that testing is permitted to begin prior to the end of development.

Figure 2 shows how $n(t)$ varies with t. The number of bugs rises sharply from zero in the time interval $0 \leq t \leq P$. In the next interval, $P \leq t \leq Q$, $n(t)$ continues to increase since $\lambda n < f(t,n)$, but the rise is less sharp because the testing team has started removing bugs from the package. New bugs are not added after time instant Q, and as a result $n(t)$ decays exponentially from Q to R. After R, the decay is faster because users also help in the discovery of bugs. In practice, P must lie above a threshold value $P_{\min}$ since integration and system testing can start only when a minimum number of modules are ready.

The expression for the total expenditure E incurred by the software company in producing the package has the form

$$E = E_d + v(S-P) + \frac{w\rho}{\lambda+\rho}[n(R)-n(S)] + wn(S) + g(R) \qquad (5)$$

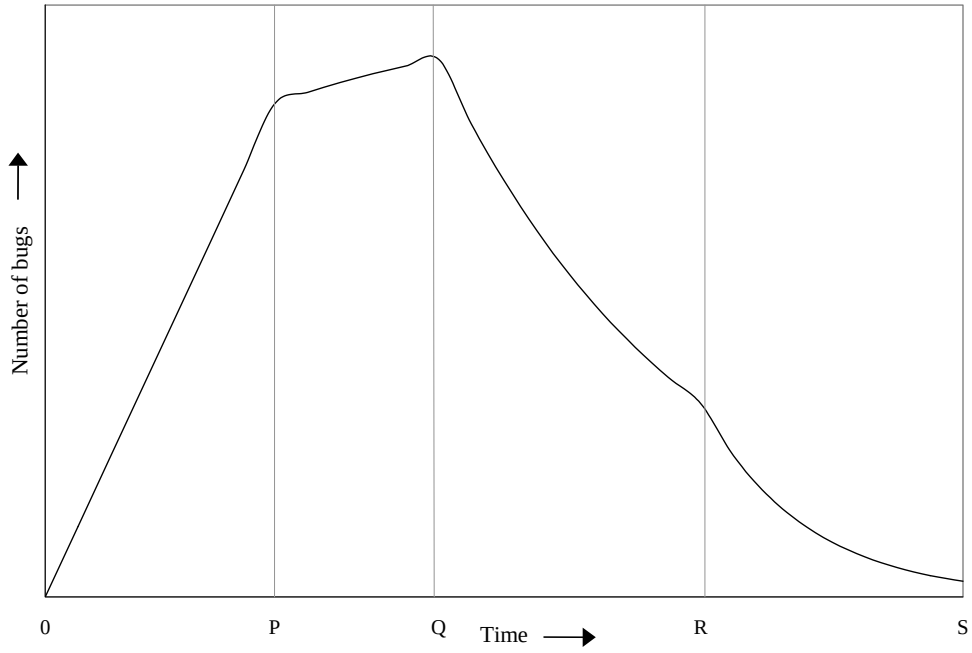

Figure 2. Variation of the Number of Bugs with Time

The symbols in (5) have the following meaning:

E_d The total cost of operation of the development team from the start of the project up to time instant Q; this cost is assumed to be constant and prespecified.

v The constant cost of testing per unit of time; this includes the salary cost of the members of the testing team, other administrative costs, the cost of machine time if any, and the cost of fixing bugs found during testing.

w The constant cost incurred by the software company when a software bug is found in the field by a customer, including the cost of fixing the bug and patching the software, together with the business loss suffered by the software company as a result of erosion of reputation and clientele.

$g(t)$ The opportunity cost incurred by the software company because of the late release of the package in the market; we assume that for $t \geq Q$, $g(t)$ is a rapidly increasing, continuous, differentiable function of t, and that dg/dt is strictly positive and increasing in t.

The values of the parameters and the functional form of $g(t)$ depend on the specific software package and the environment in which the package is being run, and must be estimated from software testing data, business considerations, and environmental conditions. Our objective is to minimize E over all allowable values of P, R, and S. We take Q to be a prespecified positive constant, and assume that the total expenditure E_d incurred in development is also prespecified. However, since $f(t,n)$, which determines the rate of increase of bugs during the period of development, can vary with both t and $n(t)$, the expenditure on development per unit of time need not be constant.

We see from (5) that the expression for E has five components. The first term is the constant cost of development. The second term is the testing cost. This is proportional to the length of time $(S - P)$ for which the testing team is in operation. The third term is the total cost of the bugs found in the field by client users in the period R to S. The number of bugs discovered and rectified in this period is $n(R) - n(S)$, of which a fraction $\rho/(\lambda + r)$ is found by users. The remaining fraction $\lambda/(\lambda + r)$ is found by the testers, and the cost of rectifying these bugs is already included in the second term. The fourth term is the risk cost for the $n(S)$ bugs that remain in the package when testing is completed at S. We expect these to be subsequently discovered in the field, but some might never be found. The last term is the opportunity cost, which is determined by the release time R of the package in the market.

The software development and testing process described here is quite general and inclusive. Most software development schemes in current use can be viewed as special cases. For example, the condition ($P = Q$ and $R = S$) corresponds to the strictly sequential scheme in which testing follows development and precedes field use. On the other hand, the condition $P = 0$ implies that testing begins right from the start and corresponds to the scheme generally in use in the development of Web applications. It can be assumed that testing is based on some form of reliability growth test (Musa 1999, Section 6.2). In the absence of any clear evidence that testing strategy, whether top-down, bottom-up or any other, makes any significant difference to the total cost of testing, we do not try to relate the cost of testing to the testing strategy.

It might seem from (5) that the best strategy for the software producer would be to release the package as soon as the development phase is over (i.e., to force $R = Q$). Then the testing effort will be shared between the producer and the users for as long as possible. This, however, is not the case. The relative values of v and w play a role in the determination of the best choice of R. A larger value of the ratio w/v tends to push R further to the right of Q (see Tables 1 through 3 later in this paper). Nor is it desirable to stop testing too early, right on the heels of the release time R. This will result in a large number of undetected bugs in the package, which will push up the risk cost.

We might also ask whether it is realistic to view w as a constant. The cost of fixing a bug found in the field would be less to the software producer when the testing team is still in operation than after it has been disbanded. So we might expect w to take two different values w_1 and w_2 where $w_1 < w_2$, $w = w_1$ for $R \leq t \leq S$, and $w = w_2$ for $t > S$. The values of w_1 and w_2 might themselves depend on R, increasing when R decreases. This gives us the following equation which generalizes (5) slightly

$$E = E_d + v(S - P) + \frac{w_1 \rho}{\lambda + \rho}[n(R) - n(S)] + w_2 n(S) + g(R) \qquad (5a)$$

Equations (5) and (5a) are very similar in form. Moreover, the nature of variation of w_1 and w_2 with R is not precisely known. So we take w to be a constant and continue to use (5) below.

It has been assumed when formulating (2) through (4) that no new bugs are generated when a bug that has been detected is fixed. This assumption can be significantly relaxed in a very simple way. Consider equation (3). If we make the reasonable assumption that when bugs are fixed, the number of new bugs generated in a small interval of time is proportional to the number of bugs that have been fixed, then (3) becomes $\frac{dn}{dt} = -\lambda n + \lambda'.(\lambda n)$, where λ' is a constant, $0 \leq \lambda' < 1$. The case $\lambda' \geq 1$ can be disregarded, as that corresponds to an unrealistic situation in which the number of bugs does not diminish with testing. This differential equation has the same form as (3); only the effective value of λ is changed to (λ - $\lambda'\lambda$). The analysis gets more complicated if the number of bugs generated is a nonlinear function of the number of bugs that have been fixed.

Two other issues call for comment. One relates to the nature of the time variable *t*. Does *t* refer to calendar (*i.e.*, ordinary) time or to execution time? According to Musa (1999, Section 8.3.1), better predictions are obtained using software reliability models when *t* is interpreted as execution time rather than as calendar time. While this has no direct effect on the mathematical formulation of the problem given in (1) through (4), it would be of relevance when the theoretical results are validated using field data. Another issue concerns the nature of variation of *dn/dt* with *n*(*t*) in (2) through (4). We have assumed here that *dn/dt* depends linearly on *n*(*t*) during testing. This is called the *basic model*. There is an alternative model known as the *logarithmic Poisson model* in which the dependence is assumed to be exponential. This model is not discussed here. Again, according to Musa (1999, Section 8.4), experimental evidence shows that both these models yield good predictions of the number of bugs, and are generally superior in this respect to all other software reliability models that have been proposed until today.

In our formulation, *E* is the total expenditure incurred by the software producer on the package. We can also look at the problem from the point of view of the community of users of the package. Ignoring the time cost of money, the cost that a user incurs has three principal components: (1) the potential loss suffered in revenue as a result of the late release of the package (or the late delivery of the product), which has a form similar to that of the opportunity cost in (5); (2) the annual maintenance cost of the package payable after the warranty period; and (3) the money value of the harassment, loss of reputation, and loss of clientele caused by the discovery of bugs in the field. The package becomes available for use at time *R*, but all users do not buy the package at the same time, nor are their parameter values the same. It is, therefore, not easy to compute the expected total cost incurred by the users of the package. Supposing it can be done, the amount can be added to *E*, and the sum can be minimized by varying *P*, *R*, and *S*. This would minimize the expected cost incurred by the entire software community for the package. The values of *P*, *R* and *S* that would be obtained by this procedure would reflect the users' concerns as well as the software producer's, and would differ from those obtained by minimizing *E* alone. The minimum cost when subtracted from the total income earned by the software producer and the users from the sale and use of the package can serve as a measure of the maximum welfare that can accrue to the community from the package. However, this interesting but difficult exercise is not attempted here. We confine our efforts to the minimization of *E*.

E is clearly a continuous function of *P*, *R*, and *S*. Moreover, *E* is a sum of terms each of which is differentiable with respect to *P*, *R*, and *S*. To find the minimum value $E_{\min}$ of *E* we differentiate *E* partially with respect to *P*, *R*, and *S* in turn, and then set the partial derivatives to zero. We get three equations in three unknowns, which when solved give us the values $P = P_0$, $R = R_0$, and $S = S_0$ at which, as we show below, the minimum project expenditure is realized. Clearly, we will always get $0 < P_0 \leq Q < R_0 \leq S_0$. Equations (1) through (4) can be solved independently of each other up to a constant of integration. To find the latter, we make use of the initial condition $n(0) = 0$ and the continuity of *n*(*t*) at *P*, *Q*, and *R*. Solving (3) and (4) we get (see Ross 1989)

$$n(t) = \begin{cases} Me^{-\lambda t} & Q \leq t \leq R \quad (6) \\ Ne^{-(\lambda+\rho)t} & R \leq t \leq S \quad (7) \end{cases}$$

where *M* and *N* are independent of *t*. Whatever be the functional form of *f*(*t*,*n*), *n*(*t*) depends only on *P* and not on *Q*, *R*, and *S* in the interval $P \leq t < Q$. Since *n*(*t*) is continuous at *Q*, it follows that *M* is a function of *P* and *Q* but not of *R* and *S*. Similarly, *N* is a function of *P*, *Q*, and *R* but not of *S*. Because *n*(*t*) is continuous at *R*, we must have $Me^{-\lambda R} = Ne^{-(\lambda+\rho)R}$, which implies that $N = Me^{\rho R}$. So we have

$$\frac{\partial n(R)}{\partial R} = -\lambda n(R), \quad \frac{\partial n(S)}{\partial R} = \rho n(S), \quad \frac{\partial n(S)}{\partial S} = -(\lambda+\rho)n(S).$$

Partial differentiation of (5) with respect to R and S yields

$$\frac{v}{\lambda} = wn(S_0) \qquad (8)$$

$$\frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0} = \frac{w\rho}{\lambda+\rho}\left[n(R_0) - n(S_0)\right] \qquad (9)$$

A third condition can be obtained by differentiating E partially with respect to P. But that is not shown here since the nature of dependence of E on P is affected by the exact functional form of $f(t,n)$. (8) and (9) lead immediately to the following two theorems:

Theorem 1: $$E_{\min} = E_d + \frac{v}{\lambda} + v(S_0 - P_0) + \frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0} + g(R_0)$$

Theorem 2: (a) $R_0 < S_0$;
(b) $n(S_0)$, the number of bugs at the end of testing, is a constant and equals $v/(\lambda w)$.

These results are quite general. The first theorem can be obtained by substituting (8) and (9) into (5). It says that provided the assumptions are all satisfied, the expression for the minimum expenditure on a software project has the form shown. It is interesting to note that this expression makes no reference to the number of bugs at all, and depends only on P_0, R_0, and S_0. If these three values can be estimated, then with the help of Theorem 1, an approximate value of $E_{\min}$ can be obtained. The first part of the second theorem states that the values of R_0 and S_0 are not equal. This follows from (9) because by assumption the derivative of $g(t)$ is always greater than 0, so the number of bugs at R_0 must exceed the number of bugs at S_0. Thus if the project expenditure is to be minimized, the time of release of the software must precede the end of testing. The second part of Theorem 2, which follows directly from (8), states that when the expenditure is minimized, the number of bugs at the end of testing is a constant—it does not depend on the exact form of $f(t,n)$, or on the exact position of P_0 in the interval $[P_{\min},Q]$ (i.e., the exact time at which testing starts), or on the duration $(S_0 - P_0)$ of testing. Thus the package at the time of release has a certain guaranteed reliability.

To show that our procedure ensures that E has been minimized we proceed as follows. Let us suppose P is fixed, so that there are only two variables R and S. It is then easily checked that

$$\alpha = \frac{\partial^2 E}{\partial R^2} = \frac{w\rho\lambda}{\lambda+\rho}\left[\lambda n(R) + \rho n(S)\right] + \frac{\partial^2 g}{\partial R^2}$$

$$\gamma = \frac{\partial^2 E}{\partial S^2} = w\lambda(\lambda+\rho)n(S)$$

$$\beta = \frac{\partial^2 E}{\partial R \partial S} = \frac{\partial^2 E}{\partial S \partial R} = -w\lambda\rho n(S)$$

The condition that must be satisfied to ensure a minimum (Rogers 1999, pp. 262-267) is that the two Hessian functions for E must both be positive, *i.e.*, $\alpha\gamma - \beta^2 > 0 \ and \ \alpha > 0$. Since by assumption $\frac{\partial^2 g}{\partial R^2} > 0$, and the remaining terms in α are all positive, the condition $\alpha > 0$ is clearly satisfied. It can be readily verified that the other condition also holds.

When P is also allowed to vary, this proof must be generalized to three variables. We then have three Hessian functions to take into account, and these must all be positive to ensure a minimum. But partial differentiation of E with respect to P appears to be possible only if we assume a specific functional form for $f(t,n)$. For example, in Scenario A below, $f(t,n)$ is a constant. Whether it is possible to prove that E achieves a minimum without making any restrictive assumptions about the functional form of $f(t,n)$ is still an open question.

Special Cases

In this section we examine four interesting special cases.

Scenario A: This scenario is similar to the general case discussed in the previous section, except that $f(t,n) = \mu$, where μ is a positive constant.

Scenario B: This is similar to Scenario A, except that $P = P_{min} < Q$. So here we are keeping P fixed at its minimum allowable value (i.e., testing is started at the earliest possible time as determined from practical considerations).

Scenario C: This is similar to Scenario A, except that $P = Q$. Thus testing begins only when development is completed, implying that line (2) of the differential equation no longer applies.

Scenario D: This is similar to Scenario C, except that $R = S$. Thus testing begins only after development ends and the package is released only after testing ends. This implies that lines (2) and (4) of the differential equation no longer apply.

For each scenario, we try to minimize the expenditure E by differentiating partially with respect to those parameters among P, R, and S that are variable, and then setting the expressions to 0 in each case. For example, for Scenario A, we differentiate with respect to each of the three parameters. Note that this can yield a value of P_0 that is less than P_{min}, making it unacceptable from practical considerations. For Scenarios B and C, we differentiate only with respect to R and S, since P has a fixed value. For Scenario D, we first put $S = R$ and then differentiate with respect to R alone. We use the name of the scenario as a superscript to distinguish between the symbols. Thus R_0^A means the value of R_0 for Scenario A, and so on. The following theorem then holds:

Theorem 3: (a) $E^A_{\min} \le E^B_{\min}$; (b) $E^A_{\min} \le E^C_{\min} \le E^D_{\min}$.

This theorem follows from the observation that Scenarios B, C and D are special cases of Scenario A, so the total expenditure for Scenario A must be smallest. Similarly, since Scenario D is a special case of Scenario C, it is clear that $E^C_{\min} \le E^D_{\min}$.

The next theorem says that the R and S values for the scenarios are related.

Theorem 4: (a) $Q < R_0^B < R_0^C < R_0^D = S_0^D < S_0^C$; b) $Q < R_0^B < S_0^B < S_0^C$.

To see why the theorem is true, we start by solving (1) through (4) for Scenario A, and get

$$\left\{\begin{array}{ll} n(t) = \mu t, & 0 \le t \le P \\ n(t) = \mu(P - \frac{1}{\lambda})e^{-\lambda(t-P)} + \frac{\mu}{\lambda}, & P \le t \le Q \\ n(t) = \mu\left[(P - \frac{1}{\lambda})e^{\lambda P} + \frac{e^{\lambda Q}}{\lambda}\right]e^{-\lambda t}, & Q \le t \le R \\ n(t) = \mu\left[(P - \frac{1}{\lambda})e^{\lambda P} + \frac{e^{\lambda Q}}{\lambda}\right]e^{\rho R - (\rho+\lambda)t}, & R \le t \le S \end{array}\right\} \qquad (10)$$

We can obtain the values of $n(R)$ and $n(S)$ and substitute into (5) to derive an expression for E. Substituting in (8) and (9) we get

$$\frac{v}{\lambda} = w\mu\left[(P_0^A - \frac{1}{\lambda})e^{\lambda P_0^A} + \frac{e^{\lambda Q}}{\lambda}\right]e^{\rho R_0^A} - (\rho+\lambda)S_0^A \tag{11}$$

$$\frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^A} = \frac{w\mu\rho}{\lambda+\rho}\left[(P_0^A - \frac{1}{\lambda})e^{\lambda P_0^A} + \frac{e^{\lambda Q}}{\lambda}\right]\left[e^{-\lambda R_0^A} - e^{\rho R_0^A} - (\rho+\lambda)S_0^A\right] \tag{12}$$

Expressions similar to (11) and (12) can be derived for each of the other scenarios. Thus for Scenario B we have

$$\frac{v}{\lambda} = w\mu\left[(P_{\min} - \frac{1}{\lambda})e^{\lambda P_{\min}} + \frac{e^{\lambda Q}}{\lambda}\right]e^{\rho R_0^B} - (\rho+\lambda)S_0^B \tag{13}$$

$$\frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^B} = \frac{w\mu\rho}{\lambda+\rho}\left[(P_{\min} - \frac{1}{\lambda})e^{\lambda P_{\min}} + \frac{e^{\lambda Q}}{\lambda}\right]\left[e^{-\lambda R_0^B} - e^{\rho R_0^B} - (\rho+\lambda)S_0^B\right] \tag{14}$$

Similarly, for Scenario C we get

$$\frac{v}{\lambda} = w\mu Q e^{\lambda Q + \rho R_0^C} - (\rho+\lambda)S_0^C \tag{15}$$

$$\frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^C} = \frac{w\mu\rho Q}{\lambda+\rho}\left[e^{-\lambda(R_0^C - Q)} - e^{\lambda Q + \rho R_0^C} - (\rho+\lambda)S_0^C\right] \tag{16}$$

and for Scenario D we get

$$\frac{v}{\lambda} + \frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^D} = w\mu Q e^{\lambda Q - \lambda R_0^D} \tag{17}$$

We now show that $R_0^C < R_0^D$. It follows from (15) through (17) that

$$\left\{\begin{array}{l} \frac{v}{\lambda} = w\mu Q e^{\lambda Q - \lambda R_0^D} - \frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^D} \\ \frac{v}{\lambda} = w\mu Q e^{\lambda Q - \lambda R_0^C} - \frac{\lambda+\rho}{\lambda\rho}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^C} \end{array}\right\} \tag{18}$$

Suppose $R_0^C \geq R_0^D$. By assumption $\partial g/\partial R$ is increasing in R and $(\lambda + \rho)/\rho > 1$, so we have

$$\frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^D} < \frac{\lambda+\rho}{\lambda\rho}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^C}$$

$$w\mu Q e^{\lambda Q - \lambda R_0^D} \geq w\mu Q e^{\lambda Q - \lambda R_0^C}$$

But this implies (18) is not satisfied, a contradiction.

We make use of (13) and (14) to show that $R_0^B < R_0^C$. We find that

$$\frac{v}{\lambda} = w\mu\left[(P_{\min} - \frac{1}{\lambda})e^{\lambda P_{\min}} + \frac{e^{\lambda Q}}{\lambda}\right]e^{-\lambda R_0^B} - \frac{\lambda+\rho}{\lambda\rho}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^B}$$

We get the inequality by just comparing this with (18), noting that

$$(P_{\min} - \frac{1}{\lambda})e^{\lambda P_{\min}} + \frac{e^{\lambda Q}}{\lambda} < Qe^{\lambda Q}.$$

To show that $S_0^C > S_0^D = R_0^D$, we note that since $R_0^C < R_0^D$, we must have

$$\frac{v}{\lambda} + \frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^C} < \frac{v}{\lambda} + \frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R=R_0^D}$$

This implies by (15) through (17) that

$$\frac{1}{\lambda+\rho}\left[\lambda e^{\rho(R_0^C - S_0^C)} + \rho e^{\lambda(S_0^C - R_0^C)}\right] < e^{\lambda(S_0^C - S_0^D)}$$

Since the L.H.S. is always ≥ 1, S_0^C must exceed S_0^D.

To show that $S_0^B < S_0^C$, from (13) and (15) we get $\rho(R_0^C - R_0^B) < (\lambda+\rho)(S_0^C - S_0^B)$. Since the L.H.S. is positive, the R.H.S. must be positive also.

Numerical Example

As a check on the derivations made in the two previous sections, we computed the values of R_0, S_0 and $E_{\min}$ for the three scenarios B, C, and D using the data supplied in Jiang and Sarkar (2003, Section 6). Some parameter values which are not included in that paper, such as those of μ, Q and $P_{\min}$ were chosen by us. The results are shown in Table 1. The computed $E_{\min}$ values do not include the development cost E_d. Since Jiang and Sarkar measure time from the end of the development phase, the opportunity cost was determined using the formula $g(t) = m(t - Q)^2$, where m is a constant and has the same value as in their paper. The parameter values satisfy all required conditions. For example, the maximum possible number of bugs is $\mu Q = 1000$. Since $\lambda = 0.10$, the condition $\mu > \lambda n(t)$ holds at all times. For scenarios C and D, the computed values of R_0, S_0, and $E_{\min}$ match the values given by Jiang and Sarkar quite closely. The R_0 and S_0 values are shown as decimal fractions but can be rounded off to the next higher integer. For Scenario B, we have taken $P_{\min} = 2.5$. By (8), for all three scenarios, $n(S_0) = 25$ (i.e., the number of bugs at the end of testing equals 25).

Tables 2 and 3 show the effect of changing the value of w without changing the value of v. A decrease in the value of w decreases the ratio w/v, bringing R closer to Q; it also decreases $E_{\min}$ since a part of the expenditure is proportional to w. Similarly, an increase in the value of w increases $E_{\min}$ and the ratio w/v, pushing R further from Q.

Table 1. Numerical Example

Common Parameters	$\lambda = 0.10$ $\mu = 200$ $v = 50$ $w = 20$ $Q = 5$ $m = 7$			
Scenario	Additional Parameters	E_{min}	R_0	S_0
D		6,465	23.62	23.62
C	$\rho = 0.04$	4,573	16.68	34.69
B	$\rho = 0.04$ $P_{min} = 2.5$	4,318	15.67	33.08

Table 2. Effect of Decreasing the Value of *w* Keeping *v* Fixed

Common Parameters	$\lambda = 0.10$ $\mu = 200$ $v = 50$ $w = 15$ $Q = 5$ $m = 7$			
Scenario	Additional Parameters	E_{min}	R_0	S_0
D		5,611	21.67	21.67
C	$\rho = 0.04$	3,911	15.11	32.18
B	$\rho = 0.04$ $P_{min} = 2.5$	3,776	14.16	30.60

Table 3. Effect of Increasing Value of *w* Keeping *v* Fixed

Common Parameters	$\lambda = 0.10$ $\mu = 200$ $v = 50$ $w = 25$ $Q = 5$ $m = 7$			
Scenario	Additional Parameters	E_{min}	R_0	S_0
D		7,183	25.18	25.18
C	$\rho = 0.04$	5,070	17.95	36.64
B	$\rho = 0.04$ $P_{min} = 2.5$	4,782	16.90	35.03

Number of Bugs

One cost that has been ignored in the foregoing analysis is the cost of fixing software bugs found during testing. This cost has been assumed to be small and has been subsumed in the testing cost of v per unit of time. If we want to take it into account separately, we need to determine how many bugs have been discovered in the testing phase. The total number of bugs discovered in the interval $Q \le t \le S$ is $n(Q) - n(S)$. Of these, a number $\frac{\rho}{\lambda + \rho}[n(R) - n(S)]$ have been found by the users and the remainder by the testers. The cost of fixing the bugs found by the users has already been taken into account and need not concern us here. The number of bugs discovered by the testers during development, *i.e.*, in the interval $P \le t \le Q$, is $\lambda \int_P^Q n(t)dt$, since by assumption the number of bugs discovered in any small interval of time is proportional to the number of bugs present in the software at that time. Thus the total number of bugs T_{bug} found by the testers is given by

$$T_{bug} = n(Q) - n(S) - \frac{\rho}{\lambda + \rho}\left[n(R) - n(S)\right] + \lambda \int_{P}^{Q} n(t)dt \qquad (19)$$

Let us suppose there is a constant cost $\beta > 0$ for fixing a bug found during testing. This cost is in addition to the testing cost of v per unit of time. We have assumed so far that the total cost of fixing the bugs found by testers is very small, i.e., $\beta T_{bug} \ll v(S - P)$. If this is not the case, the expression for the total expenditure E should be modified to

$$E = E_d + v(S - P) + \beta T_{bug} + \frac{w\rho}{\lambda + \rho}[n(R) - n(S)] + wn(S) + g(R) \qquad (20)$$

For Scenario A, we find from (10) and (19) that

$$T_{bug} = \mu Q - n(S) - \frac{\rho}{\lambda + \rho}\left[n(R) - n(S)\right] \qquad (21)$$

Incorporation of this expression in (20) and comparison with (5) reveals that the nature of dependence of the expenditure E on P, R, and S does not change much at all. This time we get

$$\frac{v}{\lambda} = (w - \beta)n(S_0)$$

$$\frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R = R_0} = \frac{(w - \beta)\rho}{\lambda + \rho}\left[n(R_0) - n(S_0)\right]$$

which yields

$$E_{\min} = E_d + \frac{v}{\lambda} + \beta\mu Q + v(S_0 - P_0) + \frac{1}{\lambda}\left[\frac{\partial g(R)}{\partial R}\right]_{R = R_0} + g(R_0) \qquad (22)$$

(22) is identical to the expression in Theorem 1 except for the presence of one additional constant term. But of course the values of P_0, R_0, and S_0 are not the same as before because the expression for the project expenditure has changed.

Conclusion

This paper presents a cost-based reliability model for software development and testing. The model is a generalization of the one described by Jiang and Sarkar (2003). In the formulation given in the description of the general model, an expression is derived for the total expenditure incurred by a software company in producing a new package. The expression takes all of the important cost, factors into account, including the testing cost, the risk cost and the opportunity cost. This expenditure is then minimized by adjusting the project milestones, such as the times at which testing begins, the package is released in the market, and testing ends. The time at which development ends is not regarded as a variable. It is assumed that the development schedule has already been finalized. A canonical expression for the minimum cost is given in Theorem 1. Theorem 2 says that to achieve a minimum value of expenditure, testing should continue beyond the time of release. It also says that the number of bugs remaining in the package at the time when testing ends is a constant. The section on special cases examines a few special scenarios which frequently arise in practice. The cost incurred by testers for fixing the bugs discovered during testing, which has been ignored in the general model, is computed and scrutinized in the numerical example section. It is found that the expression for the minimum total expenditure is only marginally affected when this additional cost is taken into account.

We have not attempted to integrate the interesting and original approach of Ji et al. (2003) into our analysis. The applicability of their control-theoretic formulation to real life situations that arise in the software industry is yet to be convincingly demonstrated. Development and testing are normally carried out by two independent teams, and frequent adjustments of the staff levels of the teams would be neither feasible nor desirable. Moreover, it is not clear whether an optimal operating point found with the help of control theory would be robust enough to withstand sudden disruptions in work schedule caused by, say, unforeseen turnover of technical staff. Such events might force frequent recomputations of the operating point. The approach described here is more general and flexible, in the sense that minor disruptions can, in all likelihood, be accommodated more easily.

The model presented in this paper makes many simplifying assumptions. Some of these can be relaxed and the model generalized as explained below.

- The fourth term in the expression for E in (5), namely $wn(S)$, gives the cost incurred on account of the bugs that remain in the package when testing ends. Some of these bugs will be discovered during the warranty period of the package, some later in its life, and some perhaps never at all. Thus it does not seem appropriate to assign the same cost to every bug that remains undiscovered at time S. For example, the cost to the software company of fixing a bug during the warranty period would presumably be higher than the cost of fixing a bug at a later time. The relative values would depend on the specifics of the warranty agreement.

- Suppose the software company wants to give a quality assurance guarantee to the customer. One way this can be modeled analytically is by allowing the value of w to increase with increase in the value of $n(S)$. This will tend to keep the value of $n(S_0)$ low when the total expenditure is minimized.

- Our model assumes that a bug once discovered is fixed completely and instantaneously, and without introducing any new bugs. None of these assumptions would be true in practice. We have already explained in the general model that, under certain conditions, the model is capable of accommodating the introduction of new bugs at the time of bug removal. It might also be possible to take the time to fix a bug into account (see Pham 2000, Section 6.2).

- The basic model described here is similar to the simple NHPP model of Goel and Okumoto (1979). Instead, we can make use of the S-shaped NHPP model described by Pham (2000, Section 5.3.2), in which bugs get discovered at a slow rate at the beginning. As testers become more experienced, the rate of bug discovery accelerates until saturation is reached. In addition, as bugs get discovered and fixed, other hidden bugs come to light.

References

Boehm, B. W. *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

Dalal, S. R., and Mallows, C. L. "When Should One Stop Testing Software?," *Journal of the American Statistical Association* (83), 1988, pp 872-879

Ehlrich, W., Prasanna, B., Stamfel, J., and Wu, J. "Determining the Cost of a Stop-Test Decision," *IEEE Software* (33), March 1993, pp. 33-42.

Goel, A. L., and Okumoto, K. "Time Dependent Error-Detection Rate Model for Software and Other Performance Measures," *IEEE Transaction on Reliability* (R-28:3), 1979, pp. 206-211.

Iansiti, M., and MacCormack,, A. "Developing Products on Internet Time," *Harvard Business Review*, September-October 1997, pp. 108-117.

Jalote, P. *An Integrated Approach to Software Engineering* (2nd ed.), Springer-Verlag, Heidelberg, Germany, 1997.

Jelinski, Z., and Moranda, P. R. "Software Reliability Research," in *Statistical Computer Performance Evaluation*, W. Freiberger (Ed.), Academic Press, New York, 1972, pp. 465-484.

Ji, Y., and Mookerjee, V. "Optimal Software Development and Debugging Policies: A Control Theoretic Approach," in *Proceedings of the 12th Annual Workshop on on Information Technology & Systems* (WITS 2002), A. Basu and S. Duta (Eds.), Barcelona, 2002, pp. 163-168.

Ji, Y., Mookerjee, V., and Sethi, S. "An Integrated Planning Model of System Development and Release," in *Proceedings of 13th Annual Workshop on Information Technolgy & Systems* (WITS 2003), D. Dey and R. Krishnan (Eds.), Seattle, WA, 2003, pp. 55-60.

Jiang, Z., and Sarkar, S. "Optimal Software Release Time with Patching Considered," in *Proceedings of the 13th Annual Workshop on Information Technology & Systems* (WITS 2003), D. Dey and R. Krishnan (Eds.), Seattle, WA, Conference, 2003, pp. 61-66.

Musa, J. D. *Software Reliability Engineering*, McGraw-Hill, New York, 1999.
Pham, H. *Software Reliability*, Springer, Singapore, 2000.
Pham, H., and Zhang, X. "A Software Cost Model with Warranty and Risk Costs," *IEEE Transactions on Computers* (48), 1999, pp. 71-75.
RTI. "The Economic Impacts of Inadequate Infrastructure for Testing: Final Report," RTI Project Number 7007.011, prepared by RTI, Health, Social, and Economics Research, Research Triangle Park, NC, for National Institute of and Technology, U.S. Department of Commerce, Technology Administration, May 2002 (available online at **http://www.nist.gov/director/prog-ofc/report02-3.pdf**).
Rogers Jr., H. *Multivariable Calculus with Vectors*, Prentice-Hall, Englewood Cliffs, NJ, 1999.
Ross, S. L. *Introduction to Ordinary Differential Equations* (4th ed.), John Wiley, New York, 1989.
Schick, G. J., and Wolverton, R. W. "An Analysis of Competing Software Reliability Models," *IEEE Transactions on Software Engineering* (SE-4), 1978, pp. 102-120.
Zhang, X., and Pham, H. "A Software Cost Model with Warranty Cost, Error Removal Times and Risk Costs," *IIE Transactions* (30), 1998, pp. 1135-1142.
Zheng, S. "Dynamic Release Policies for Software Systems with a Reliability Constraint," *IIE Transactions* (34), 2002, pp. 253-262.