A Formal Model of the Software Test Process

João W. Cangussu, *Student Member*, *IEEE*, Raymond A. DeCarlo, *Fellow*, *IEEE*, and Aditya P. Mathur, *Fellow*, *IEEE*

Abstract—A novel approach to model the system test phase of the software life cycle is presented. This approach is based on concepts and techniques from control theory and is useful in computing the effort required to reduce the number of errors and the schedule slippage under a changing process environment. Results from these computations are used, and possibly revised, at specific checkpoints in a feedback-control structure to meet the schedule and quality objectives. Two case studies were conducted to study the behavior of the proposed model. One study reported here uses data from a commercial project. The outcome from these two studies suggests that the proposed model might well be the first significant milestone along the road to a formal and practical theory of software process control.

Index Terms—Feedback control, process control, software test process, software testing, modeling, state variable.

1 Introduction

Research in software process modeling dates back to the early 1970s. A detailed account of its evolution is given by Cugola and Ghezzi [1]. In this account, the features of PROSYT, a second generation process-centered software engineering environment, are grouped by Ghezzi into three main areas: 1) process modeling, 2) process enactment, and 3) system architecture. Under process enactment, Ghezzi states: "To better control process execution, PROSYT allows process managers to specify a *deviation handling* and a *consistency checking* policy. Such policies state the level of enforcement adopted . . . and the actions that have to be performed when the invariants are violated as a result of deviation, respectively." The problem of "managing unforeseen situations," also referred to as "tolerating deviations," is formulated and solutions are proposed by Cugola [2]. Cugola has also proposed policies to handle various types of deviations. Though useful in practice, the policies do not assist in the computation of quantitative values of corrections that are often needed when deviations occur in process variables that can be, and often are, measured in numerical terms; project schedule and product quality are examples of such process variables.

We are aware of formal and rigorous procedures applied to the software process [3], [4], [5], such as Statistical Process Control [6]. However, in practice, these and other less formal but rigorous approaches fall far short of the formalisms for process control that exist in other engineering disciplines. For example, control theory is rich in formalisms that are practical and in regular use in chemical process control. Temperature control, aircraft wing control, and continuous gravimetric control are only a few examples of a myriad of control applications in the industry that rely on control algorithms firmly grounded in theory. One of the issues control theory deals with in a formal manner is the reduction of deviations from one or more set points characteristic of the process under control, e.g., temperature in a boiler control system.

Research reported herein is perhaps the first step toward a formal theory of software process control. The long term objective of this research is to borrow, adapt, and modify, when needed, from the richness of control theory; especially from the theory of state variables, which has proven useful in modeling engineering, biological, and social processes [7], [8], [9]. For the short term, we focus our efforts on one significant phase of the Software Development Process (SDP), namely, the Software Test Process (STP). Though control of other phases of the SDP is often as important to an organization as the control of the test phase, the following two reasons motivated us to select the STP: 1) STP lends itself well to the characterization of input, output, and internal process variables and 2) there is a significant amount of data available from past and ongoing projects that is a key to the conduct of case studies to investigate the applicability of our model and approach. This second reason helps us to solve the difficult problem of identifying and estimating the key parameters to be included in any model of the STP.

Within the STP, we decided to focus on the system test phase due to its easier formulation and modeling. Although there are variations on how to conduct the system test phase and how it impacts the other phases of the SDP, we assume the test and debug cycles alternate or occur concurrently *not* based on a predetermined schedule but as and when the need arises. Specifically, a software product $\mathcal{P}$ could be in one of three states: *Test*, *debug*, and *End*. When in *Test*, $\mathcal{P}$ is executed until a failure is detected. At this point, if a debug-mode condition is true, then $\mathcal{P}$ enters the *debug* state; otherwise, it remains in *Test*. The debug-mode condition could be, for example, "The number of failures detected exceeds a threshold." As another example, it could be "A failure that will prevent succeeding tests from running as

• *J. Cangussu and A.P. Mathur are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907-1398. E-mail: {cangussu, apm}@cs.purdue.edu.*
• *R.A. DeCarlo is with the Deptartment of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285. E-mail: decarlo@ecn.purdue.edu.*

*Manuscript received 9 Apr. 2001; accepted 5 Dec. 2001.
Recommended for acceptance by M. Broy.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 113955.*

planned." $\mathcal{P}$ remains in the *Debug* state until errors are found and fixed, when it returns to the *Test* state. However, while $\mathcal{P}$ is in *Debug*, another avatar of it could remain in the *Test* state. This is when we say that testing and debugging are taking place concurrently. This test-debug cycle can be handled effectively by our model; other scenarios are possible with some care in collecting the needed data.

The remainder of this paper is organized as follows: In Section 2, we state a process control problem that arises within the context of the STP. This problem is by no means the only control problem that one might need to deal with. It is, however, the focus of our attention in the remainder of this paper. In Section 3, we offer an introduction to feedback control from the point of view of a software engineer. The modeling approach and the motivation appear in Section 4, which also describes a model based on the use of state variables. The behavior of the model under extreme conditions is analyzed in Section 5. This analysis is the first step toward assessing the accuracy of the model. Estimation of various parameters of our model is discussed in Section 6. One case study to analyze the behavior of the model is presented in Section 7. A description of other modeling approaches is presented in Section 8. Finally, in Section 9, we present our conclusions and outline directions for future work in the area of process modeling using the theory of feedback control.

2 The STP Control Problem and Its Context

The key components of an SDP are: 1) specificatiom, 2) design, 3) coding, and 4) testing with multiple feedback paths from the completion of each phase back to earlier phases. We focus on the system test phase, specifically, on the control of the time and effort required to reduce the errors by a desired fraction. The unit and integration test phases are not accounted for in our model. Thus, upon the completion of coding and unit testing, our model incorporates any effort to test and debug. Such testing often occurs as the final phase prior to delievering the application to the customer for beta testing or actual use. Though the various phases of the STP can occur concurrently, we assume that such concurrency affects only the effort applied during the execution of a phase. Also, though inspections can be and often are performed after each phase of the SDP, we do not consider inspections as part of the STP.

2.1 The STP Control Problem

Consider an application $\mathcal{P}$ under test. We assume that the quality and schedule objectives were set at the start of the SDP and revised before the test process has started. For a scheduled target date or a set of checkpoints leading to a target date, the quality objective might be expressed in a variety of ways: 1) as the reliability of $\mathcal{P}$ to be achieved by the target date [10], 2) as the number of errors remaining at the end of the test phase, or 3) as the increase in code coverage [11]. Further, the quality objective could be more refined and specified for each checkpoint.

We assume that a test manager plans the execution of the test phase to meet the quality and schedule objectives. Such a plan involves several activities, including the constitution of the test team, selection of the test tools, identification of training needs, and scheduling of training sessions. Each of these activities involves estimation. For example, constituting the test team requires a determination of how many testers to use. The experience of each tester is another important factor to consider. It is the test team that carries out the testing activity and, hence, spends the effort that will hopefully help in meeting the objectives. The ever limited budget is usually a constraint to contend with. During the initial planning phase, a test manager needs to answer the question, "How much effort is needed to meet the schedule and quality objectives?" Experience of the test manager does help in answering this question. However, we approach this problem from a mathematical standpoint.

The question stated above is relevant at each checkpoint. We assume that the test manager has planned to conduct reviews at intermediate points between the start and the target date. The question might arise at various other points also, for example, when there is attrition in the test team. An accurate answer to the above question is important not only for continuous planning but also for process control and resource allocation. A question relevant for control is: "How much *additional* test effort is required at a given checkpoint if a schedule slippage of, say, 20 percent can be tolerated?" This question could be reformulated in many ways in terms of various process parameters. A few other related questions of interest to a test manager are:

1. Can testing be completed by the deadline and the quality objective realized?
2. How long will it take to correct the deviations in the test process that might arise due to an unexpectedly large number of reported errors, turnover of test engineers, change in code, etc.?
3. By how much should the size of the test team be increased if the deadline is to be advanced without any change in the error reduction objectives?

To answer these questions, we propose a model based on the application of the theory of feedback control using a state variable representation. Our model allows comparison of the output variables of the software test process with one or more "setpoint(s)." Such a comparison leads to the determination of how the process inputs and internal parameters ought to be regulated to achieve the desired objectives of the STP.

3 Feedback Control in the Context of the STP

When applied to the STP, the objective of feedback control is to assist a test manager in making decisions regarding the expansion of or reduction in workforce and the change of the quality of the test process. The control process is applied throughout the STP. Though it does not guarantee that the STP will be able to meet its schedule and quality objectives, it does provide information that helps the test manager determine whether or not the objectives will be met and, if not, what actions to take.

We assume that the schedule objective is specified as: *Complete the test process by a specified time* t_f. The quality objective is specified as: *Ensure that at most* r_f *errors remain in* $\mathcal{P}$ *at time* t_f. As noted earlier, the quality objective could be

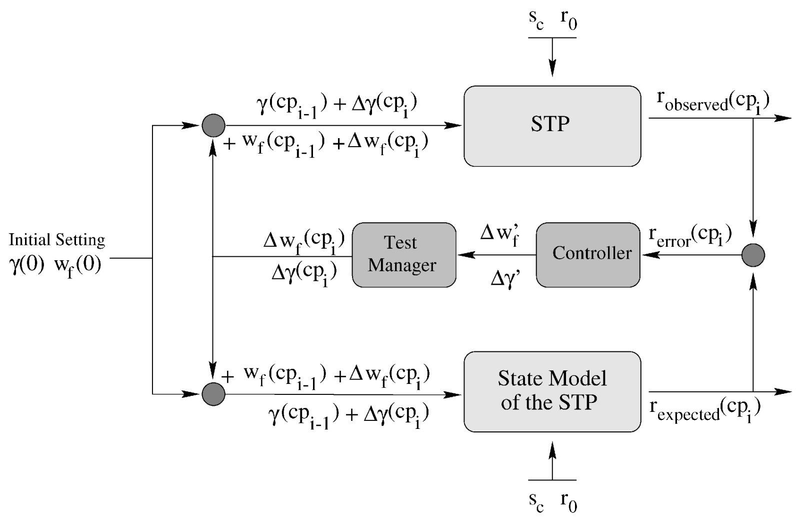

Fig. 1. Feedback control of the software test process. w_f, γ, s_c, $r(cp_i)$, and r_0 denote, respectively, the size of the test team, the overall quality of the test process, the complexity of the product under test, the number of remaining errors at the ith check point cp_i, and an estimate of the number of remaining errors at the start of the STP.

stated in several other ways. Finding and reducing the number of remaining errors is often a significant objective of the STP. It is due to this emphasis that we decided to use the number of remaining errors in the formulation of the quality objective. In our work, we assume that any reduction in software errors that remain in a product improves the quality of that product. In the work that follows, we do not distinguish among the various types of errors such as specification errors, critical and noncritical errors, etc.

Hence, we specify the quality objective in terms of the number of remaining errors. The difficulties in estimating the number of remaining errors are overcome using techniques described in Section 6. We are also aware that an STP might be driven by objectives other than, or in addition to, the two specified here. However, for the purpose of the control of STP, our current focus is on schedule and quality. We also assume that, prior to the start of the STP, the project manager sets up a monitoring schedule that consists of a sequence of $k, k > 0$ checkpoints over time. The ith checkpoint, denoted by cp_i, is specified at time t_i when monitoring is to take place and r_i is the number of errors expected to remain in $\mathcal{P}$ at time t_i. The first checkpoint occurs some time after the start of the STP, i.e., at $t_1 > 0$, and the last checkpoint coincides with the deadline. The economics of a software project will most likely constrain its budget. The budget is not included explicitly in our model. However, the proposed feedback control mechanism assists a project manager in tracking possible budget overruns. Also, a project manager need not explicitly specify r_i. Instead, the specification could be in terms of a fraction by which the number of errors is expected to be reduced. Thus, for example, this fraction could be $0 < f_i < 1$ at the ith checkpoint. This would imply that the number of errors expected to remain in $\mathcal{P}$ at checkpoint cp_i is $f_i \times r_{i-1}$.

The use of feedback control can be understood from Figs. 1 and 2. The continuous line in Fig. 2 shows the expected variation in $r(t)$ over the course of the STP; $r(t)$ is computed using the state model described in Section 4.3. The dashed lines indicate the prediction used by a test manager to generate a schedule in terms of the checkpoints. The checkpoints are determined by the test manager and the expected values of the number of remaining errors at each checkpoint, denoted by $r_{expected}(\mathrm{cp}_i)$ at checkpoint cp_i, can be read off the $r(t)$ function. The test process is started at time $t = t_0$, at which point, $\mathcal{P}$ contains $r(0) = r_0$ errors.

At checkpoint cp_i, the observed value of the number of remaining errors, denoted by $r_{observed}(cp_i)$, is compared with $r_{expected}(cp_i)$ to generate the input $r_{error}(\mathrm{cp}_i)$ to the controller. The controller computes $\Delta w_f'$ and $\Delta\gamma'$, which are, respectively, the changes needed to the workforce and the quality of the test process, for the objectives to be met. The test manager uses the changes computed by the controller to decide whether or not any change is needed in the test process. Thus, for example, the test manager might decide to ignore the controller output. In this case, we assume that the STP continues until the next checkpoint without any changes in w_f and γ. However, the manager might decide to make use of the output from the controller and change only

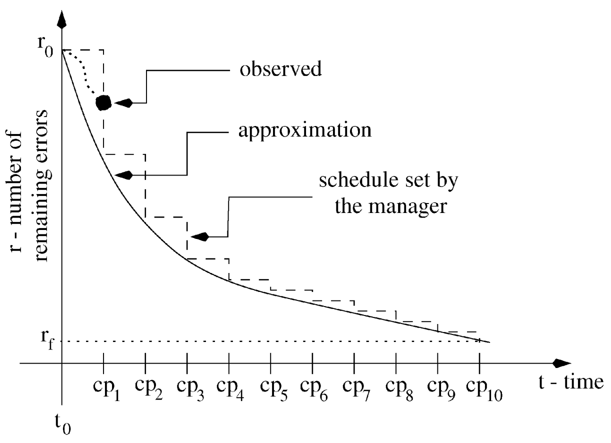

Fig. 2. Checkpoints in a Software Test Process. cp_i denotes the ith checkpoint.

the workforce to $w_f + \Delta w_f$ and keep γ at its current value. Obviously, several possibilities exist. In any case, the STP continues with the workforce and process quality set to, respectively, $w_f + \Delta w_f$ and $\gamma + \Delta\gamma$. Note that these updated values of the workforce and the quality of the test process are also input to the model which in turn recomputes $r(t)$.

The process described above continues until the STP is completed. During this process, the manager might decide to change the checkpoints and, possibly, the objectives of the STP. In any case, the STP model and the controller are provided with the updated values and generate useful data throughout the STP. Of course, there is no guarantee that there exists a feasible solution to the problem of completing the STP with the desired objective being met. However, if a feasible solution exists, then the model finds it and assists the test manager in steering the STP.

4 Modeling the Software Test Process

A number of variables and parameters are specific to the system test phase:

1. $r(t)$—the number of remaining errors at time t,
2. w_f—size of the test team,
3. s_c—program complexity,
4. t—time measured in appropriate units,
5. γ—a constant characterizing the overall quality of the test process,
6. $e_f(t)$—effective test effort at time t, and
7. $e_r(t)$—error reduction resistance at time t.

Our model allows one to choose from a variety of or a combination of existing complexity measures to obtain a value of s_c. For example, we could use program size, cyclomatic complexity [12], or a combination of both to compute s_c as explained in Section 6.3.

The coefficient γ characterizes the overall quality of the test process and represents environmental factors such as pressure due to the deadline, test methodology used, organization of the test team and the process, experience and expertise of the test team personnel, and possibly other factors. Although a single coefficient is unable to fully represent the quality of the test phase, when appropriately chosen, it appears to be adequate. The effective test effort (e_f) is the actual effort expended by the test team to perform all activities required of them to test the system.

The process of error removal might cause the introduction of new errors in the application under test. This results in additional effort by the test team. Further, other factors that reduce the quality of the test process also tend to offset the benefits of applying the effective test effort. We use the error resistance, e_r, to model the impact of γ on e_f.

4.1 Model Assumptions

Three key assumptions are presented next. We believe that these assumptions govern the STP. The assumptions and the resulting equations lead to a state model of the STP. The solution of this state model allows a test manager to answer schedule related questions mentioned earlier in Section 1. Let $\dot{r}(t)$ and $\ddot{r}(t)$ denote, respectively, the first and second derivative of $r(t)$ with respect to time.

Assumption 1. *The rate at which the velocity of the remaining errors changes is directly proportional to the net applied effort (e_n) and inversely proportional to the complexity of the program under test. This leads to.*

$$\ddot{r}(t) = \frac{e_n(t)}{s_c} \Rightarrow e_n(t) = \ddot{r}(t)s_c. \tag{1}$$

The first assumption is justified as follows: When the same metric or a combination of metrics is used to compute software complexity for two different programs under test, it is reasonable to expect that more effort will be necessary to test the more complex program. If, for example, cyclomatic complexity [13] and LOC are used to determine s_c, a larger program with more paths will likely require more test effort than a smaller program with a smaller number of paths. The net applied effort (e_n) is the balance of all the effort acting on the product under test. This results from the difference of the effective test effort applied by the test team minus any "frictional" forces that oppose the applied effort. Since r represents the number of remaining errors, its first derivative $\dot{r}$ is the error reduction velocity (v_e). Consequently, $\ddot{r}$, which denotes the rate of change of $\dot{r}$, is an acceleration.

In the world of software, (1) is analogous to Newton's second law of motion for physical systems, where e_n is analogous to physical force, $\ddot{r}$ to acceleration, and s_c to mass. In the physical world, a larger mass requires a greater force to move it a given distance at a desired velocity. In the world of software, higher program complexity requires larger effort to reduce errors by a given fraction at a desired error reduction velocity ($\dot{r}$).

It is widely believed that the difficulty in finding program errors increases as the test phase progresses. Assuming a fixed team size, this implies that the effective test effort is directly proportional to r. This observation suggests the next assumption.

Assumption 2. *The effective test effort is proportional to the product of the applied work force and the number of remaining errors. This leads to*

$$e_f(t) = \zeta(s_c)\, w_f r(t), \tag{2}$$

where $\zeta(s_c) = \frac{\zeta}{s_c{}^b}$ is a function of software complexity. Parameter b depends on the characteristics of the product under test. Borrowing from COCOMO [14], [15], we set b to 1.05, 1.12, or 1.20, for organic, semidetached, *and* embedded *mode projects, respectively.*

Justification of this assumption follows by analogy with the predator-prey system described by Volterra [16]. Here, the decline in the prey population is proportional to the number of possible encounters between the predators and the prey, i.e., the product of the populations of predators and prey. Assumption 2 above presents similar characteristics to this widely accepted model. The probability of finding an error is equivalent to an encounter between a tester and an error. The tester plays the predator role and errors are the prey. There are $w_f\, r$ possible encounters. ζ may also depend on r, although we do not make this dependency explicit. The parameter ζ defines the decline rate and it may decrease as r gets smaller ($\zeta = \zeta(r)$). That is, as the test process continues, the errors become more difficult to find, not only because there are less of them, but also because some errors require a combination of events to be triggered and it is most likely this combination will be discovered by the testers only in the final phases of testing, if it is discovered at all. This behavior is captured by changes in ζ over different periods of the STP.

Assumption 2 can be understood with another analogy. In a spring-mass system, the restoring force is determined by the spring stiffness and by the extension of the spring beyond its natural length. Increasing the spring stiffness, or its extension, increases the restoring force. The number of remaining errors is analogous to the spring length. At the beginning of the test phase, r is larger than it is toward the end. Hence, the effective effort decreases with r. The work force can be related to the spring stiffness. The larger the work force, the greater the restoring force, i.e., the effective effort. Thus, spring stiffness is analogous to w_f and spring extension to the number of remaining errors (r) in the application. In (2), ζ remains constant over a period and must be calibrated for the STP under consideration. The behavior of Assumption 2 is similar to the rate of decrease of errors [17], [18], [19] when software reliability models are applied to the STP [20].

The effective test effort is opposed by a force intrinsic to the test process. We refer to this force as the error reduction resistance, denoted by e_r. This observation leads to the last assumption.

Assumption 3. *The error reduction resistance opposes, is proportional to the error reduction velocity, and is inversely proportional to the overall quality of the test phase. This leads to*

$$e_r(t) = -\xi \frac{1}{\gamma} \dot{r}, \tag{3}$$

for an appropriate constant ξ. The negative sign indicates that the error reduction always opposes $\dot{r}$.

The justification of Assumption 3 relies on an analysis of its behavior under extremal conditions. A very low quality will induce a large resistance, i.e., as $\gamma \rightarrow 0$, $e_r \rightarrow \infty$. The same is true for values of $\dot{r}$; i.e., the larger $\dot{r}$ is, the larger the error resistance e_r. This implies that the faster one attempts to reduce the remaining errors, the more likely one is to make mistakes that slow the test process. This behavior resembles that of a physical dashpot. The coefficient of viscosity of the liquid inside the dashpot is $\frac{1}{\gamma}$. Therefore, a small coefficient of viscosity corresponds to the test phase being conducted in a smooth and careful way. For example, this may imply that the number of new errors inserted is relatively small. Similarly, a larger coefficient of viscosity corresponds to an increased tendency to insert errors during the debugging process. The velocity component in the dashpot is the dual of the error reduction velocity ($\dot{r}$). Thus, γ and the rate at which errors are found determine the error reduction resistance, which is analogous to the damping force generated by the dashpot. In the remainder of this paper, we use r, $\dot{r}$, $\ddot{r}$, e_f, e_r, and e_n, to denote, respectively, $r(t)$, $\dot{r}(t)$, $\ddot{r}(t)$, $e_f(t)$, $e_r(t)$, and $e_n(t)$.

4.2 A Differential Equation Model of the STP

e_n, e_f, and e_r are related by the following force balance equation:

$$-e_f + e_r = e_n, \tag{4}$$

where e_f has a negative sign because it opposes the increase in errors. Substituting e_f, e_r, and e_n by the righthand sides of (1), (2), and (3), respectively, results in the following second-order differential equation:

$$-\zeta\, w_f\, r \quad -\xi \frac{1}{\gamma} \dot{r} = s_c\, \ddot{r}. \tag{5}$$

To help clarify the behavior of (5), consider a physical system that consists of a solid block attached to an extended spring and a dashpot. The spring is extended by 100 units affixed to a wall, as is the dashpot. The spring restoring force will move the block from the initial position (100 units) to a position as close to zero as possible; the dashpot will retard this movement. Here, we assume an overdamped system and, hence, the block will never reach a negative position. This behavior is analogous to what happens in the system test phase where the resistance offered by the dashphot is related to e_r and the restoring force to e_f. By assumption, the system test phase starts with a program of complexity s_c and with 100 percent of remaining errors. The ideal goal is to remove errors until r approaches zero. The effective effort due to w_f and r, and the error reduction resistance due to γ and $\dot{r}$, will determine the rate of decrease of r. Unfortunately, despite the overlap, the physical analogy given earlier can break-down. For example, if a spring were quickly detached from a moving block, inertia would keep the block moving. An analogous behavior during the STP would imply the impossible, that errors would continue being removed after the work force was removed from the project.

Given $r(0) = r_0$, the initial error reduction velocity is $v_e = 0$. Under this condition, the solution to (5) is:

$$r(t) = \frac{r_0 \lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 t} - \frac{r_0 \lambda_1}{\lambda_2 - \lambda_1} e^{-\lambda_2 t}, \tag{6}$$

where λ_1 and λ_2 denote the distinct roots of the characteristic equation of (5) [7], [16]. The two distinct negative roots are due to the assumption of a stable overdamped process. If underdamping were allowed, r would reach a negative value, which is meaningless in the STP. The overdamping requirement [21] restricts the values of γ to less than

$$\frac{\xi}{2(s_c^{(1-b)} w_f\, \zeta)^{\frac{1}{2}}}.$$

This is calculated by forming the characteristic equation and requiring the discriminant of the associated quadratic formula to be nonnegative.

From Fig. 3, we note how r, $\dot{r}$, and $\ddot{r}$ change over time. Errors are generally easy to find at the beginning of an STP and, therefore, $\dot{r}$ is relatively high. As the STP progresses over time, error detection becomes increasingly difficult

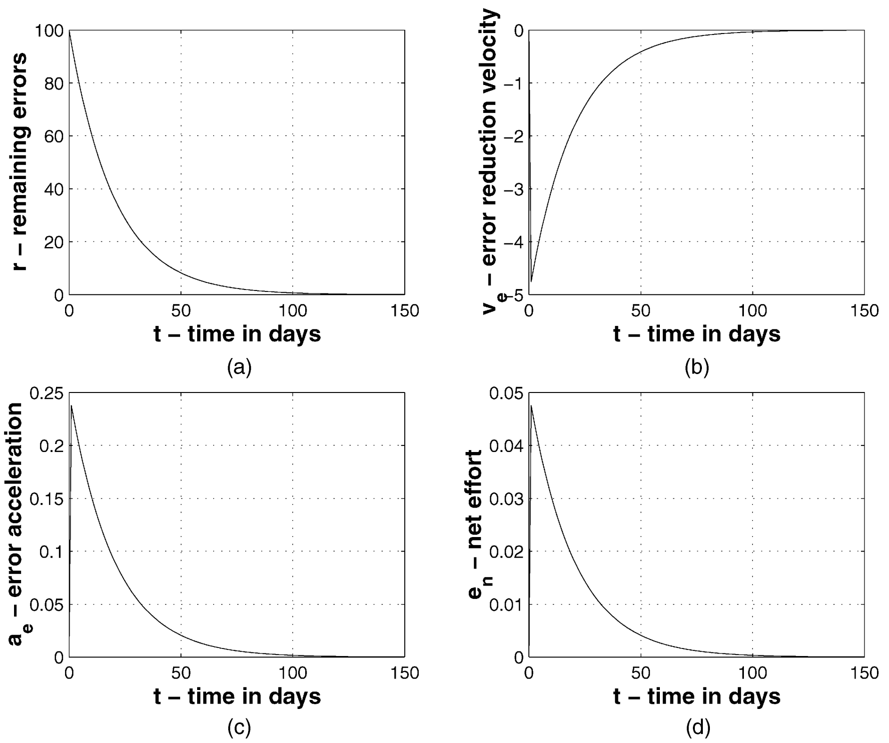

Fig. 3. Number of remaining errors, reduction in the number of remaining errors, rate of reduction in the number of remaining errors for $s_c = 0.2$, $\gamma = 0.005$, and $w_f = 2$.

and, hence, $\dot{r}$ decreases toward zero, as observed in Fig. 3b. Therefore, we have a deceleration until chances of finding new errors becomes almost zero, as in Fig. 3c. As expected, in accordance with Assumption 1, e_n approaches zero as $t \rightarrow \infty$ as depicted in Fig. 3d.

In our model, we consider only two forces acting on the product under test: the effective test effort (e_f) and the error resistance (e_r). However, there are other forces that affect a product during the STP. Hence, it is wise to include the model forces represented by the auxiliary effort when a test tool is being used and also to include an opposite force when the test team spends time learning the use of a new test tool during the STP. Other forces, not considered in this paper, include efforts for communication and adaptation.

Because our model captures the dominant dynamics of the STP, leaving out certain forces as described above may introduce some error between the predicted behavior and the observed behavior. In addition, the STP is often beset by disturbances that may cause a delay in the process. For example, suppose a test team is using a populated database to test the product and, for some reason, the database becomes unavailable for one working day, making the team unable to test the program for the entire day. This would constitute a 100 percent disturbance if the time unit were days and a 20 percent disturbance if the time unit were weeks. In all cases, the disturbance represents a force, say F_d, opposing the effective test effort, e_f [22]. Thus, it can be seen as a possibly event-dependent percentage of the effective test effort. Of course, as more elements of the STP are accounted for in the model, then the contribution of, for example, learning and communication to F_d will diminish to zero. Nevertheless, denoting $F_d(t)$ by F_d and incorporating F_d into (5) results in (7).

$$\ddot{r} = -\frac{\zeta\, w_f}{s_c^{(1+b)}} r - \frac{\xi}{\gamma\, s_c}\dot{r} + \frac{1}{s_c} F_d. \tag{7}$$

4.3 A State Model for the STP

The state model is a matrix differential equation in a vector of state variables which, in our case, are the remaining errors, r, and the error reduction velocity, $\dot{r}$. These state variables are sufficient to model the dominant dynamics of the STP and also serve as the output variables of interest. Hence, with r and $\dot{r}$ as state variables, the following controllable canonical state model [7] results from (7).

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{\zeta\, w_f}{s_c^{(1+b)}} & -\frac{\xi}{\gamma\, s_c} \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{s_c} \end{bmatrix} F_d. \tag{8}$$

The matrix multiplying the vector of state variables is called the A-matrix. The model must be initialized at $t = 0$, which requires at least an estimate of $r(0) = r_0$ and the observation that the error reduction velocity $\dot{r}(0) = 0$. The value of r_0 can always be updated as observed data becomes available over the course of the STP. Indeed, data is needed to make initial estimates for the proportionality constants ξ and ζ. Also, in this formulation, the workforce variable, w_f, is taken as a parameter in the A-matrix rather than as an external input, such as F_d, because, in the STP, it is typically constant over one or more checkpoint periods. For example, if $w_f = 5$ from time t_i to t_j, $i < j$ and changes to 6 from t_{j+1} to t_k, $(j+1) < k$, we use $w_f = 5$ to observe the behavior for the first period and then switch to a system with $w_f = 6$ for the second period. To extract w_f from the A-matrix and make it an external input would move the model into the nonlinear category. By viewing w_f as a parameter, then the model remains in the piecewise-constant linear category and is amenable to well-known solution techniques with the use of feedback as a parametric control to achieve management schedule objectives and reduce the impact of F_d. In addition, by having a piecewise-constant A-matrix, we may also update estimates for ξ and ζ as new data becomes available.

4.4 Using the Model for Feedback Control

We now explain how feedback control can be applied to adjust STP parameters to meet the desired objective. The objective of an STP is restated below after combining the constraints on time to completion and the number of remaining errors.

> Given that the program under test contains $r(0) = r_0$ errors at the start of STP, it is desired to complete the STP in t_α weeks, such that the number of remaining errors r is reduced to $\alpha \times r_0$.

Once the STP objective has been set up, the project manager has two options. Option 1 is to organize a team of testers and start the STP. Under this option, the manager does not estimate any of the model parameters and, hence, does not apply the model to check if the desired objective is indeed feasible.

Option 2 is to estimate the model parameters, using data from past similar projects, required to meet the objective and use the model to test if indeed the objective can be met. If the model indicates that the objective cannot be met with the estimated set of parameters, then another set is tried. This process continues until a reasonable set of parameter estimates is found, at which point the STP is started. Note that only w_f and γ are under the control of the manager. Also, budgetary restrictions might impose additional constraints on these parameters. The estimation of parameters is discussed in Section 6.

5 Extreme Case Analysis

We now subject the model in (8) to an extreme case analysis with $F_d = 0$. Our purpose is to evaluate its behavior under extreme conditions to find if the behavior is consistent with what one would expect of an STP under such conditions. We consider extreme conditions at the intersections of low and high values of software complexity and quality of the test phase. Note that it is inappropriate to analyze our model for the effects of extreme values of the work force because communication among testers is not included. For the purpose of this analysis, we arbitrarily set $w_f = 5$.

The extreme case analysis proceeds as follows: For each of the four possible combinations of high/low s_c and γ, we first compute s_c and set γ. These values, and that of w_f, are plugged into (8), which is then solved. The solution is depicted by an $r(t) \times t$ plot. From this plot, we read off $t_{0.05}$, which denotes the time needed in days to reduce the number of remaining errors to 5 percent of its initial value. Both s_c and γ affect $t_{0.05}$. We then compare the values of $t_{0.05}$ to determine the nature of their effect and compare it with what a software tester would expect intuitively.

For all four extremal cases, the parameter ξ is set to 100 as a factor of normalization. The parameter ζ cannot be estimated because the expected deadline is not available. Thus, we set $\zeta(s_c)$ to $\frac{20}{s_c^b}$ for b = 1.12 representing a *semi-detached mode* project [15].

For the purpose of our analysis, s_c is considered to be a convex combination of M_1, the lines of code measured in 10 KLOCs, and M_2 the average of cyclomatic complexity per function. The weights for M_1 and M_2 are set to, respectively, $\alpha_1 = 0.75$ and $\alpha_2 = 0.25$. Thus,

$$s_c = \sum_{i=1}^{2} \alpha_i M_i.$$

Case 1—low s_c and low γ: Consider a program with 5,000 lines of code with an average cyclomatic complexity of 2. Thus, we have $M_1 = 0.5$, $M_2 = 2$ and, hence, $s_c = 0.75 * 0.5 + 0.25 * 2 = 0.875$. We quantify a low quality test phase by setting $\gamma = 0.05$. Substituting for parameters in (8) and solving for r, we observe the behavior exhibited in Fig. 4a. As is evident from Fig. 4a, it requires 52 days to reduce the number of remaining errors in this product to less than 5 percent.

Case 2—low s_c and high γ: Fig. 4b represents the behavior predicted by our model for an almost perfect, though unrealistic, test phase. Some characteristics of such a test phase are: Each member of the test team knows exactly what each part of the product does, can apply 100 percent of available time to the test effort, does not communicate with other members of the test team, and requires no learning once testing has begun. For this test phase, we set $\gamma = 0.95$. We also consider a program containing 5,000 lines of code with an average cyclomatic complexity of 2. This yields $s_c = 0.875$. These parameter values lead to a solution depicted in Fig. 4b. From this figure, we observe that it will take about three days to reach the desired level of error reduction. As one might expect, as $s_c \to 0$ and $\gamma \to 1$, the time required to remove the errors diminishes and becomes proportional to the net applied effort.

Case 3—high s_c and low γ: Here, we assume that the program under test contains 300,000 lines of code and has an average cyclomatic complexity of 30. This leads to $s_c = 30 * 0.75 + 0.25 * 30 = 30$. We set $\gamma = 0.05$ to represent low quality. The solution to (6) is plotted in Fig. 4c. The plot reveals that it will take about 2,700 days for five testers to reduce the number of remaining errors to less than 5 percent of the initial count. As expected, this is considerably longer than in Case 2. In general, we can state that, as $s_c \to \infty$ and $\gamma \to 0$, then $T \to \infty$, where T is the time required to reduce the errors to less than 5 percent.

Case 4—high s_c and high γ: In this case, we assume a high quality test phase, as described in Case 2, and a high level of complexity, as described in Case 3. The other parameters are set as before. Fig. 4d reveals that the high quality of the test phase has a significant impact on the time required to reduce errors. In this case, it will take about 141 days to reach the desired error reduction. The behavior observed in Fig. 4 is close to what a test engineer might expect under extreme conditions. The following common sense expectation of a test engineer is mimicked well by our model:

> Complexity of the application under test and quality of the test process have a significant effect on the time to test.

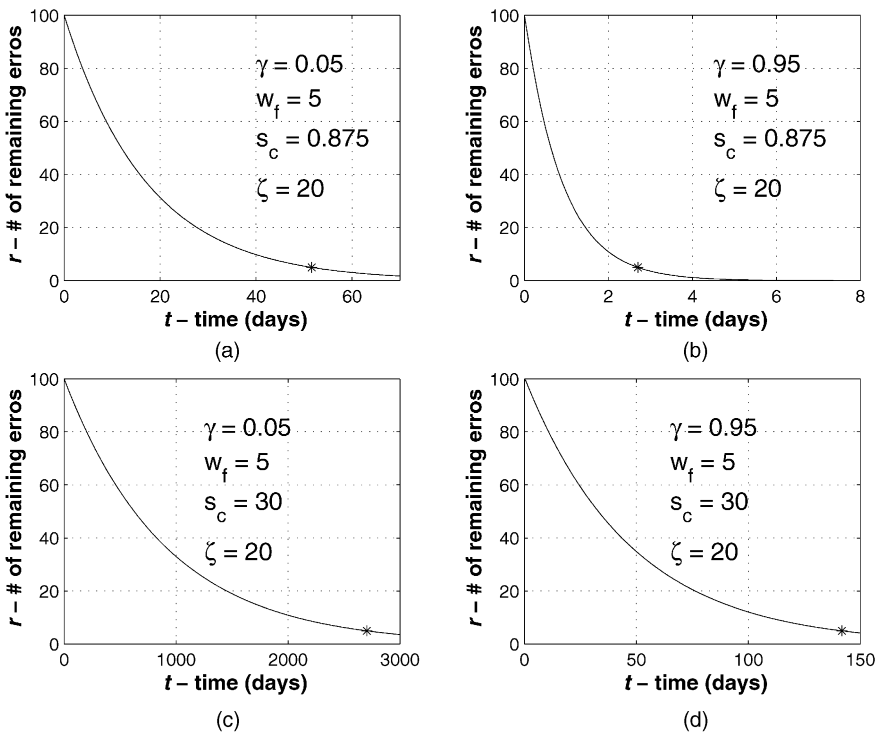

Fig. 4. Variation of r for four extreme cases based on s_c and γ.

6 Estimating Model Parameters

The set of parameters listed in Section 4 is representative of the most significant aspects of the STP. Estimation of these parameter values is essential to a successful application of the state model. Some parameters are relatively easy to quantify, while others are subjective. Also, different organizations use different metrics and methodologies in the STP. There are no globally accepted metrics for the parameters and variables involved in the STP. Most models for the SDP rely on intuitively and/or empirically derived values for the parameters. This situation suggests a need for a methodology to help guide the estimation process. In the remainder of this section, we discuss how one could estimate each of the several parameters needed to apply our model.

6.1 Size of the Work Force

The work force, w_f, is defined as the number of testers per unit time. As testers might be added or taken away as the STP progresses, w_f is updated at each checkpoint.

6.2 Estimation of ξ, ζ, and r_0

ξ, ζ, and r_0 are computed from data obtained from the current project using an algorithm described in this section. Obviously, this data is not available at the start of the STP. However, a manager may have data from similar past projects that can be used to obtain initial estimates until data from the current project becomes available, when the estimates can be improved.

The state model of (8) has the general form of

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{9}$$

where $x(t) = [r(t)\ \ \dot{r}(t)]^T$, A is the proper 2×2 matrix, B is a 2×1 vector, and $u(t)$ is the input. It is well-known that the solution of (9) is given by $x(t) = e^{At}x(0)$ for all $t \geq 0$ for a zero input. To compute ξ and ζ, we need to compute

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{\zeta\, w_f}{s_c^{(1+b)}} & -\frac{\xi}{\gamma s_c} \end{bmatrix}, \tag{10}$$

from which we can obtain ξ and ζ, as all the other parameters are known at this time.

Initially, $\dot{r}(0) = 0$, but $r(0)$ is not known. Further, project data ordinarily consists of the number of errors found and fixed in a time period of length, say, T_1. This number is denoted by $d^{(k)} \equiv r(kT_1) - r((k-1)T_1)$. Although $r(t)$ has the general form specified in (6), we use a single exponential approach to obtain a local approximation for $\dot{r}(t)$, i.e., we locally approximate $r(t) = \alpha e^{-\lambda t}$. For a fixed T_1, let $m = \alpha e^{-\lambda T_1}$, then

$$\begin{aligned} d^{(k)} &\equiv r(kT_1) - r((k-1)T_1) \\ &= mr((k-1)T_1) - mr((k-2)T_1) = md^{(k-1)}. \end{aligned}$$

This allows us to generate the following equation based on available data whose solution will provide a least square fit form:

$$\begin{aligned} &\left[d^{(k)}d^{(k-1)}...d^{(3)}\right] = m\left[d^{(k-1)}d^{(k-2)}...d^{(2)}\right] \Rightarrow \\ &m = \left[d^{(k)}d^{(k-1)}...d^{(3)}\right]\left[d^{(k-1)}d^{(k-2)}...d^{(2)}\right]^{-R} \end{aligned}$$

and α can be computed by

$$\alpha = \left[(m^2 - m)(m^3 - m^2)...(m^n - m^{n-1})\right]^{-L}\left[d^{(2)}d^{(3)}...d^{(n)}\right],$$

where superscripts $-R$ and $-L$ represent the Moore-Penrose pseudoright and left inverse, respectively. Having

m and α computed as above, we obtain $\lambda = \frac{1}{T_1}(ln(\alpha) - ln(m))$ and, therefore, $\dot{r}(kT_1) \approx -\lambda\alpha e^{-\lambda T_1} = -\lambda m$.

Inherent in the above is a single exponential approximation to obtain a reasonable estimate of the velocity data. Now, we must redo the above development in the proper matrix format. Using the data available and the approximated $\dot{r}$, we compute the difference for a specific period of time as $D^i = [r(i)\ \dot{r}(i)]^T - [r(i-1)\ \dot{r}(i-1)]^T$. It can be shown that $D^i = MD^{i-1}$, where $M = e^{AT}$, T is the time increment between two consecutive measurements of data, and A is the A-matrix of (9). Therefore, we can compute M from

$$R_1 = MR_2 \Longrightarrow M = R_1 {R_2}^{-R}, \qquad (11)$$

where

$$R_1 = [D^n D^{n-1} D^{n-2} \dots D^4 D^3] \text{ and}$$
$$R_2 = [D^{n-1} D^{n-2} D^{n-3} \dots D^3 D^2].$$

The Spectral Mapping Theorem [7] shows that the eigenvalues of M, say ${\lambda_1}^M$ and ${\lambda_2}^M$, have the following relation with the eigenvalues of matrix A: ${\lambda_1}^M = e^{\lambda_1 T}$ and ${\lambda_2}^M = e^{\lambda_2 T}$. Therefore, $\lambda_1 = \frac{1}{T} ln({\lambda_1}^M)$ and $\lambda_2 = \frac{1}{T} ln({\lambda_2}^M)$. The eigenvalues are the roots of the characteristic polynomial of the A matrix, as in (10), which is

$$\Pi_A(\lambda) = det[\lambda I - A] = \lambda^2 + \frac{\xi}{\hat{\gamma} s_c}\lambda + \frac{\zeta \hat{w}_f}{s_c^{(1+b)}}. \qquad (12)$$

Since ξ and ζ are the only unknowns at this time, we can compute them by matching the roots of $\Pi_A(\lambda)$ to λ_1 and λ_2 computed above.

An initial estimate of r(0) can also be computed using M obtained from the observed data. Let

$$P = \begin{bmatrix} M^2 - M \\ M^3 - M^2 \\ \vdots \\ M^n - M^{n-1} \end{bmatrix} \mathit{and}\ Z = \begin{bmatrix} D^2 \\ D^3 \\ \vdots \\ D^n \end{bmatrix} \mathit{and}\ x_0 = \begin{bmatrix} r(0) \\ \dot{r}(0) \end{bmatrix}.$$

We know that $Z = Px_0$ and we can compute $x_0 = P^{-L}D$. However, this results in a high initial velocity and penalizes the computation of $r(0)$. The problem is solved by applying a weighted least squares approach [23]:

$$Z = PWx_0 \Rightarrow x_0 = ((PW)^T(PW))^{-1}(PW)^T Z,$$

where

$$W = \begin{bmatrix} w_{r_0} & 0 \\ 0 & w_{v_0} \end{bmatrix}$$

is the weight matrix. The weights w_{r_0} and w_{v_0} are usually defined as $\frac{1}{\sigma}$ [23], where σ is the standard deviation computed from the observed data for r and from the estimates of $\dot{r}$. In the case of the STP, due to the exponential decay, σ will increase with additional data. This will make the weights decrease when an increase is expected. To avoid this problem, we defined the weights as $w_{r_0} = \frac{w}{\sigma_r}$ and $w_{v_0} = \frac{w}{\sigma_{\dot{r}}}$ for $w = 1 + e^{-\frac{d/2}{\varrho}}$, where d is the expected deadline and ϱ is the number of observed values used in the computation.

6.3 Software Complexity

Since software complexity (s_c) significantly impacts the behavior of our model, this section presents a combination of metrics for its representation. Specifically, we allow for the contribution of multiple metrics to compute a value for s_c through the use of a convex combination [24]. Assume n normalized metrics $M_1 \dots M_n$ with, respectively, $\alpha_1 \dots \alpha_n$ weights are under consideration. $s_c = \sum_{i=1}^n \alpha_i M_i$ is used to compute the final value for software complexity with $\sum_{i=1}^n \alpha_i = 1$. The software complexity ranges from a lower bound of 0 to an upper bound of ϕ. Parameter calibration techniques [25] can be used to define the upper bound ϕ for individual organizations.

The convex combination approach is similar to the weighted sum metric by Khoshgoftaar and Muson [26]. Fenton and Pfleeger [27] describe the risks of using such an approach. However, no complexity metric is widely used and/or accepted and a single value characterizing the overall complexity is needed in our model, justifying such a flexible choice.

6.4 Quality of the Test Phase (γ)

A number of factors enter into γ for a given organization, including:

1. work force experience and expertise,
2. test strategy/adequacy,
3. coverage criteria,
4. tool use/adequacy, and
5. deadline pressure.

Let us denote each of the factors in 1-5 by, respectively, $\gamma_1, \dots, \gamma_5$. In our model, we assign each of these a value between 0 and 1. As in the case of software complexity, we take a convex combination of these values to obtain $\gamma = \sum_{i=1}^5 \alpha_i \gamma_i$ under the condition that $\sum_{i=1}^5 \alpha_i = 1$. The values of γ lie between 0 and 1, with $\gamma = 1$ denoting the highest quality process. Other criteria can be added or those listed here deleted, depending on their respective importance. Note that γ_1, γ_4, and γ_5 have correspondence with parameters in COCOMO II [15].

7 Case Study: The Cobol Transformer Project

Two case studies were carried out to investigate the performance of the state-based model of the STP. The first case study was conducted using data reported by Knuth [28] during the development of $\mathrm{T_EX78}$ [29]. In this section, we present a second case study using data collected from a commercial project to transform a program written in Cobol into a functionally equivalent program in SAP/R3, as described next.

7.1 Project Description

Razorfish, a company located in Cambridge, Massechusettes, was given an application, denoted S_{COBOL}, that contains about 4 million lines of code in Cobol to be transformed into a functionally equivalent application in SAP/R3, hereafter referred to as $S_{SAPR/3}$. Razorfish developed a tool, hereafter referred to as *transformer*, to automate this transformation. Errors in the transformer were detected

TABLE 1
Parameters Values Used in Fig. 5

Quality Features	α_i	Period 1		Period 2		Period 3	
		q_{i1}	$q_{i1}\alpha_i$	q_{i2}	$q_{i2}\alpha_i$	q_{i3}	$q_{i3}\alpha_i$
(1) Experience/expertise of w_f	0.3	0.60	0.18	0.70	0.21	0.80	0.24
(2) Tool use and adequacy	0.2	0.20	0.04	0.50	0.10	0.60	0.12
(3) Test Plan adequacy	0.2	0.80	0.16	0.80	0.16	0.80	0.16
(4) Test cases quality	0.3	0.20	0.06	0.30	0.09	0.77	0.23
		$\gamma_1 = 0.44$		$\gamma_2 = 0.56$		$\gamma_3 = 0.75$	

TABLE 2
Parameters Values Used in Fig. 5

three segments local approximation							
	weeks	γ	ξ	ζ	s_c	w_f	$F_d{}^*$
Period 1	1 to 6	0.44	19.89	22.78	48	3	59%
Period 2	6 to 10	0.56	31.46	69.86	48	3	61%
Period 3	10 to 14	0.75	35.07	156.45	48	3	33%
Remaining Time	14 to ...	0.75	49.09	156.45	48	3	25%

F_d is the average disturbance during the period.

by executing and comparing the outputs of both the generated $S_{SAPR/3}$ code and the original Cobol code. Therefore, any difference in the comparison implied an error in the *transformer*. Upon fixing the errors, a regression test was carried out to determine if any new errors were introduced.

The information presented here about the project was obtained through interviews with the project manager, developers, and the test team. Razorfish maintains data on what errors are found, by whom and when an error was found, and who is responsible for fixing the error. This data was tabulated by the project manager and made available to the authors.

7.2 Parameter Estimation for the Cobol Transformer Project

Data from the first six weeks of the project was used to obtain an estimate of the initial number of errors in the transformer. These estimates constitute proprietary data for Razorfish and, therefore, the values are presented here after normalization. Our estimate of r_0 was considered reasonable by the project manager. This estimate was subsequently improved when data from weeks 6 to 14 became available and proved to be 94 percent accurate at the end of the project. s_c, γ, and w_f were computed in collaboration with the project manager. The remaining parameters were computed using MATLAB [30] based on the procedure described in Section 6.

Estimation of software complexity: s_c was computed as a convex combination of two metrics: 1) M_1, the number of 10s of KLOC, resulting in $M_1 = 25$ and 2) M_2, the number of 10s of grammar productions used to specify the Cobol syntax. The *transformer* had to deal with different versions and dialects of Cobol significantly increasing the language specification and complexity. The Cobol specification had approximately 1,400 productions, resulting in $M_2 = 140$. Assuming that KLOC accounts for 80 percent of the complexity measure,

$$s_c = \alpha_1 M_1 + \alpha_2 M_2 = 0.8 \times 25 + 0.2 \times 140 = 48.$$

Estimation of the quality γ of the test phase: The testers at Razorfish had significant experience in testing similar systems and had access to an available test tool that partially automated the testing process and, hence, significantly increased the quality of the test phase.

In consultation with the project manager, we divided the first 14 weeks of the test phase into three periods, as indicated in Table 2. Four features (column 1 of Table 1) were used to define the overall γ. Column 2 lists the effective contribution of the feature to the overall quality. The quality value of each feature changed over the three periods. Each "Period" in Table 1 has two columns: Column 1 lists the quality level q_{ij} (0 to 1) of the feature i for period j; Column 2 lists the product $q_{ij}\alpha_i$, $i = 1, \ldots, 4$, and $j = 1, 2, 3$. γ_j, the average quality for period j, is given by $\sum_{i=1}^{4} q_{ij}\alpha_i$. The details of the choice of qs and αs are beyond the scope of this paper. We emphasize that the features needed to be defined on a per project/company basis.

$\gamma = 0.44$ for period 1 indicates a low quality. This was due to the minimal use of the testing tool as the testers were analyzing screen conversions and the generated layout could not be checked automatically by the tool. Also, the testers were using an *in vitro* data base to test the *transformer* and, hence, the quality of the test cases was not satisfactory.

The second period showed improvement ($\gamma = 0.56$) due to the use of an improved test set to test the Cobol application. During this period, the testers began using real data from a "small" company that makes use of the native Cobol application. This led to an increase in the number of parts of the application that were exercised. An increase in the use of the tool also occurred. The third period was demarcated from the previous by the fact that test data from another company using the native Cobol application became available and the system could be exercised more completely with a further increase in tool utilization. Thus, we set $\gamma = 0.75$. The values for the test plan adequacy are the same for all periods since test plans were used and seemed to be quite appropriate for the *transformer* project. The values for work force experience/expertise increased from 0.6 to 0.8 as the test team adapted to the project.

Size of the work force: The size of the test team remained constant at three testers for the period under consideration ($w_f = 3$).

Constant of Proportionality ξ and ζ: The constants of proportionality ξ and ζ were computed as in Section 6.2 for Periods 1-3 and appear in Table 2.

Disturbance Force: Fig. 5 shows the initial expected behavior for the *transformer* project plotted using the expectations of the project manager. We can also see that

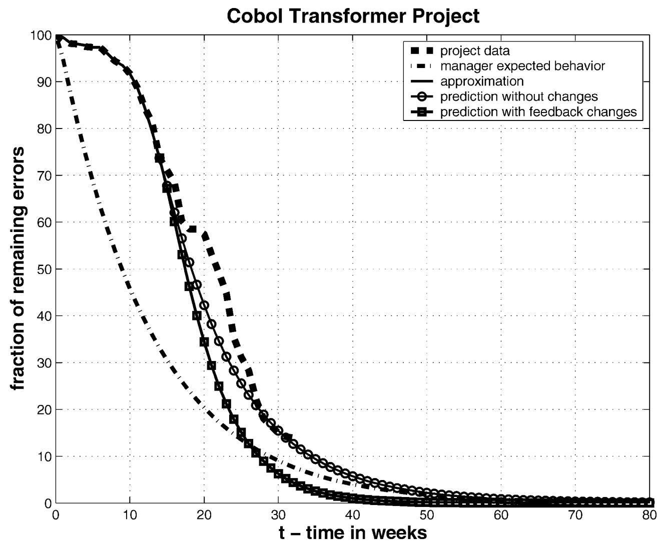

Fig. 5. Actual and predicted behavior of the Cobol Transformer project observed in the change in the fraction of remaining errors.

the observed behavior diverges from the expected for the first 14 weeks of the project. This divergence is due to disturbances present during the STP and alternatives to correct its effect, using feedback, are discussed later.

As explained earlier, in Section 4.3, disturbance is a force opposing the effective test effort (e_f). The disturbance is computed by taking the expected behavior and introducing an opposite force (F_d) that produces the observed behavior, i.e., matches the collected data. F_d is the input in our model and, so, can be computed by:

$$F_d(q) = B^T\Phi^T(t_1 - q)K^{-1}(t_0, t_1)[x_1 - \Phi(t_1 - t_0)x_0],$$

where $\Phi(t_1, t_0) = e^{A(t_1 - t_0)}$ is the state transition matrix and

$$K(t_0, t_1) = \int_{t_0}^{t_1} \Phi(t_1 - q)BB^T\Phi^T(t_1 - q)dq$$

is the the controllability Gramian [7].

The average disturbance for the three periods of the *transformer* project is, respectively, 59 percent, 61 percent, and 33 percent of the e_f. This means, for example, that, for the first period, an opposite force equivalent to 59 percent of the effective test effort (e_f) was present.

The disturbance is high during the first two periods and decreases subsequently. The disturbance is due to communication, adaptation, hardware and software failure, illness, and other forces not accounted for in our model. We believe disturbances are almost always present in software processes. Hence, a disturbance force equivalent to 25 percent of e_f was (minimally) extrapolated for the remaining period in view of the experience of the test team. Although the disturbance seems high at the beginning of the process, it is usual to have a 40 percent disturbance under normal conditions [22]. The high disturbance during periods 1 and 2 is due to a temporary slow down in the test process due to a slow debuging process. During period 3, the test team was more focused and able to concentrate on testing rather than on the source of the error and its removal.

7.3 Results

Fig. 5 depicts the results of the test phase of the *transformer* project. The integral mean square error was computed by

$$\hat{\varphi} = \sqrt{\int_0^\infty |f_1(\tau) - f_2(\tau)|^2 d(\tau)}.$$

This produced a 2.302 error norm when data from the real project is compared to the model approximation for the first 14 weeks. Parameter values used to generate the data in Fig. 5 are listed in Table 2. These parameters and the disturbance inserted during the process produce the approximation shown in Fig. 5.

Two assumptions could be made when analyzing the results shown in Fig. 5. First, any error in the *transformer* will produce an error in the $S_{SAPR/3}$ generated code. Second, not all errors in the *transformer* will affect this specific project, i.e., the transformation from S_{COBOL} to $S_{SAPR/3}$ will not be able to exercise all features of the *transformer*. If the first assumption is valid, then, according to Fig. 5, it will not be possible to meet the predetermined deadline. Assuming no change in the test process, i.e., maintaining the same parameters as in the third period described before and keeping the disturbance at 25 percent, our model predicts that it will take more than 50 weeks to deliver a product with a reasonable level of errors.

The second assumption indicates that, by the end of the test phase, i.e., after 25 weeks, some errors will remain in the *transformer*, but the goal of the project would be achieved. Stated differently, the generated system will be functionally equivalent to the original Cobol system, ensuring a successful project. The remaining errors in the *transformer* cannot be found by testing a specific Cobol system and more effort must be spent to decrease the number of remaining errors to a reasonable level. Thus, the second assumption seems more reasonable than the first one.

It should be clear that the predictions from our model do not depend on any of the above two assumptions. For the purpose of analysis, we are concerned with the remaining errors in the *Transformer* and not in the generated code.

Based on the solution to our model, one predicts that it will not be possible to complete the project by the expected deadline. This becomes evident by comparing the approximation to r with the expected curve. Hence, we ask: "What changes can be made to the STP in order to meet the deadline?" The use of feedback helps us answer this question.

We note that, by week 14, the number of errors dropped to approximately 67 percent of their initial value and, if the process continues without any alterations, it will take approximately 35 weeks to reach the expected level of error reduction that is approximately 14 percent. Indeed, no adjustments were made to the project and, in 37 weeks, the project reached the desired level of errors. This result shows a 3.4 percent accuracy in our prediction.

Now, suppose that the project manager desires to achieve the same results in only 10 weeks. We ask "What modifications are necessary?"

To answer this question, we note that the largest eigenvalue of a system determines the slowest rate of convergence and dominates how fast the variables converge. Therefore, we need to adjust the largest eigenvalue of the model so that the responses converge to the desired values within the remaining weeks. This goal can be achieved by

$$r(T+\Delta t) = r(T)\; e^{\lambda_{max}\Delta t}, \tag{13}$$

where $r(T)$ is the number of remaining errors at time T, $r(T+\Delta t)$ is the desired value for r after a lapse of Δt time units, and λ_{max} is the eigenvalue to be computed. In the Razorfish project, we want r to converge from 67 percent at week 14 ($r(14) = 67 percent$) to 14 percent at end of week 24 ($r(14+\Delta t)$ = 14 percent) for $\Delta t = 10$. Solving (13) for these values results in $\lambda_{max} = -0.1566$.

The eigenvalues of a system are defined by the roots of the characteristic polynomial ($\Pi_A(\lambda) = det[\lambda I - A]$). Computing the characteristic polynomial of our model produces

$$det[\lambda I - A] = det\begin{bmatrix} \lambda & -1 \\ \frac{\zeta\,\hat{w}_f}{s_c^{(1+b)}} & \lambda + \frac{\xi}{\hat{\gamma}\,s_c} \end{bmatrix} = \lambda^2 + \frac{\xi}{\hat{\gamma}\,s_c}\lambda + \frac{\zeta\,\hat{w}_f}{s_c^{(1+b)}}, \tag{14}$$

where $\hat{\gamma} = \gamma + \Delta_\gamma$ and $\hat{w}_f = w_f + \Delta_{w_f}$. To set the eigenvalue of the model described by (8) to -0.1566, we need to make changes in the values of these parameters. Considering that no changes can be made to ξ, ζ, and s_c, we are left with two options: increase the work force ($\Delta_{w_f} > 0$) or improve the quality of the test phase ($0 < \Delta_\gamma \leq 0.25$).

Varying Δ_{w_f}, keeping all other values constant, and then finding the roots of the characteristic polynomial, produces the results depicted in Fig. 6a. We can observe that λ_{max} reaches the desired value of -0.1566 when Δ_{w_f} reaches 1.5. This implies that the w_f must be increased by 1.5 in order to meet the deadline, assuming that all other parameters are kept constant. The result of increasing the work force by 1.5 is presented in Fig. 5. Fig. 6b presents similar results achieved due to the variation of Δ_γ. As can be observed, a maximum increase in γ is not sufficient to achieve the desired results.

In the *transformer* project, we are interested in completing the test phase by the predetermined deadline and all possible combinations of increasing w_f and/or γ to accomplish this task are depicted in Fig. 6c.

The model can also be used to analyze alternatives according to the flexibility of the deadline and the availability of resources. That is, if 1.5 people are not available, the manager can choose the alternatives of how many testers can be inserted and how much the deadline can be extended. It is essentially an exercise of maximizing customer satisfaction within the limitations of resources. As stated earlier, optimization techniques are available in Control Theory [8], [31] to provide these results, but a discussion on such techniques is beyond the scope of this paper.

We note that the test manager did not apply the suggestions from the model. However, in general, we can conclude that our model behavior is reasonably accurate when applied to the Cobol *transformer* project and that feedback can be used to answer questions related to performance and the cost of the STP.

8 Related Work

In this section, we examine some techniques used to model and/or control one or more phases of the SDP. We focus on characteristics such as dependency on life cycle model, ability to do optimization, self-regulation, coupling and cohesion, completeness, calibration, and friendliness [29]. These features are rated as high, medium, and low.

Software Project Dynamics: The approach developed by Adbel-Hamid and Madnick [22] makes two major contributions to the modeling of SDP. The first contribution is that the model is integrated and, hence, provides a macro understanding of the SDP through the integration of micro components. The second contribution stems indirectly from the first one. It is in that the model can predict the general behavior of the SDP by propagating the effect of changes from one phase to the subsequent phases. This ability to predict enhances our understanding of how a local change will affect the behavior of the entire project. The model's suitability to simulation is another useful characteristic of their work.

The sequence of software development phases assumed in Abdel-Hamid's model suggests the use of a waterfall life cycle model. We therefore consider the model to be highly dependent on which model of life cycle is used. The model is not restricted to any specific methodology. The strategy of modeling the design and coding phases as a single phase decreases model cohesion and increases coupling. The model does not address all phases of the SDP and, as ascribed, it does not address issues of optimization. The self-regulation and calibration features are also not addressed in their work.

Statistical Process Control (SPC): This control technique, as described by Florac and Carleton [6], is not a model in itself. However, it provides useful tools to improve the controllability of the SDP and, hence, an understanding of its concepts is important to our research. Two concepts used

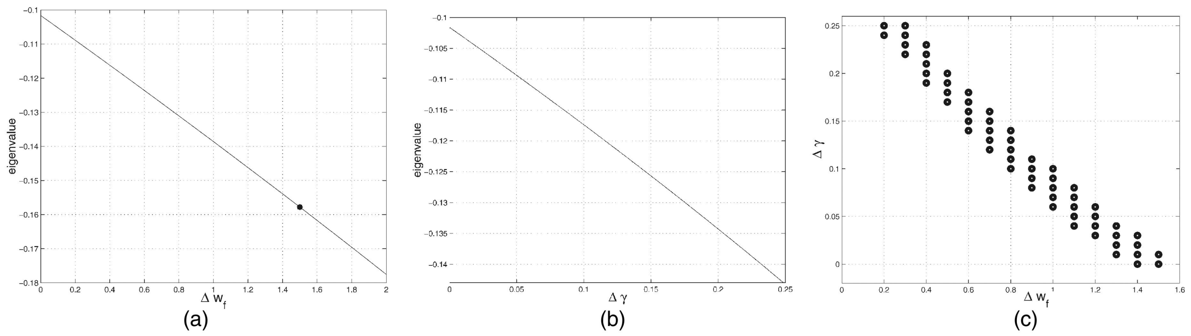

Fig. 6. Relationship that must be maintained between the largest eigenvalue of the system, Δ_γ, and Δ_{w_f} to achieve the desired number of remaining errors.

in Statistical Process Control are most relevant to our work: stability and capability [32]. Graphic tools, such as control charts and capability histograms, are used to analyze the process.

The applicability of SPC is independent of the life cycle model and the development methodology. It does require that a measurement process be instituted with the SDP. Optimization of process parameters can be addressed in SPC, though it requires an analysis of many alternatives. That is, when the number of parameters that affect the SDP is relatively high, the combinations of possible values are even higher and the analysis of all alternative choices becomes difficult if not impractical. The problem is not with the analyses of alternatives, but in their enumeration. SPC does not address self regulation.

Software Process Simulation: Discrete event simulation techniques are commonly used to model and evaluate the SDP [33]. The phases of the SDP can be modeled independently and then combined for the entire process. Simulation helps the managers to answer "what if" questions restricted to some constraints. It also provides reasonable answers to these types of questions.

Simulation models, such as those ones based on Petri Nets [34], can be developed in accordance with the life cycle model and the methodology used. The life cycle is represented by the sequence in which software development phases are considered and features regarding the methodology can be modeled through parameters and variable definitions. Process optimization is theoretically feasible though often impractical in a commercial setting. A self-regulation mechanism is not present in this approach. Since a model can be defined on a per-company basis, it is possible to achieve a high level of cohesion and a low level of coupling, thereby making Software Process Simulation models attractive. A new model can be easily defined if changes are detected in the SDP. Restrictions regarding the completeness of a model are due to creative aspects of the initial phases of the SDP and present the same measurements and evolution problems as other dynamic approaches. The models can also be calibrated empirically. The key question answered by the feedback control model is "How should the process parameters be altered to meet the process objectives?" Simulation is another method to answer this question. However, simulation is likely to became a highly inefficient means to do it when the number and range of parameters is large.

Identification of parameters, estimation, and model validation are crucial to a successful application of feedback control as proposed in this paper. The wealth of literature in the area of software metrics is likely to be of assistance in these tasks [15], [27]. We note that the work on cost and effort estimation is complementary to the work described herein, but supports different objectives. For example, COCOMO II can be used to estimate the cost and effort required for a given software project. Thus, it is a useful model for project planning and management. The feedback control as proposed herein is used for the dynamic control of the software process. It uses data from the test process to advise a test manager as to what should be done next in order to achieve the project goals. In contrast, COCOMO II assists a manager in planning the project budget, schedule, and resources. Feedback control can use some of this data, e.g., schedule, as input to assist a manager in maneuvering the test process.

9 Summary and Discussion

The widespread use of differential equations and the state variable approach to model different types of systems, combined with the advantages of using classical control theory techniques, encouraged us to investigate a formal approach to modeling the STP. Results from two case studies suggest that the formal approach presented in this paper is reasonably accurate in predicting the behavior of the test process. The use of parametric control to handle changes in the environment improves the flexibility and applicability of the model. Even though our model does not account for several features of the STP, such as time required for new personnel to adapt to the process and the communication overhead, we believe that, in its present form, it is useful in that it captures the essential behavior of the STP. The model can also be used to reduce the cost of STP and improve its performance in the presence of disturbances. The behavior of the model for the STP enhances our belief that the application of the state variable approach is appropriate and likely to result in an improved understanding of the changes during the software process.

The availability of an analytical model, ability to quantify and estimate model parameters, and an ongoing measurements process are the basic requirements for a successful application of any modeling approach grounded in classical control theory. Organizations at levels 4 and 5 of the

Capability Maturity Model (CMM) [35] are likely to have a measurements process in place. Data collected through this process could be used in the estimation of model parameters. However, the organization level of a company is not a requirement for a successful application of such approaches. Even when the SDP is not well-defined, the state variable approach can be applied when measurements of the variables to be controlled and estimates of model parameters are available. Therefore, the model can also be applied to organizations at level 3 or below subject to the availability of measurements. That is, even though organizations at levels 1 to 3 do not share the environmental aspects of levels 4 and 5, they can benefit from the use of the model described here. However, we cannot expect the same accuracy as one is likely to achieve within organizations at higher levels. If the software development process is not well-defined, it is unrealistic to expect availability of accurate data. Despite that, the use of our model might force an improvement in the quality of the data collected and perhaps in the SDP itself.

Several aspects of modeling the STP remain to be investigated. Also, one case study is currently underway to further investigate the behavior of the model. A sensitivity analysis of the model to small variations in parameters has been completed [36]. It indicates how changes in parameters affect the model. By comparing these results with the expected behavior, we are able to determine how the model ought to change to accomplish the expected behavior.

Any theory, especially one that is new, is likely to face barriers to its use. The theory of process control, based, as in this paper, on the theory of feedback control, faces several barriers three of which are identified and discussed below.

Estimation of parameters: Estimation of several parameters is a prerequisite to the use of our approach. The lack of standardized definitions and widely accepted procedures for estimation make parameter estimation an error-prone task. For example, there is no single definition of software complexity. Thus, as described earlier, one could combine several complexity measures and compute a composite complexity metric. However, the inclusion of reusable code adds a new dimension to the computation of complexity. The quality of the test phase is a subjective measure. No two test managers are likely to arrive at an identical quantification. We believe that experience with the test process and data from previous test processes within the same company could help in arriving at accurate parameter estimates.

Background of the test manager: Our model is formal and based on a knowledge of mathematics few test managers are likely to possess. Thus, one might argue, how could a test manager use such an approach in practice? We believe that this barrier could be overcome effectively by packaging our approach in a tool. This will hide the details, such as the solution to differential equations, not needed by the test manager.

Process elements: humans versus devices: Control theory was developed, and is applied, in situations where the various control elements are electro-mechanical devices and not human beings. In the software test process, the control signals are, among several things, directly effecting people such as when the work force is to be increased or when the quality of the test process is to be increased. Furthermore, the feedback control loop is closed by a human being, namely, the test manager. A natural question to ask is: "How effective will a formal control technique be in such a human-intensive environment?" Of course, only time will offer an answer to this question. However, we believe that a sound theory of software process control that has proven to be effective in controlled experiments is more likely than not to encourage test managers to adopt it.

We believe that success in this research will lead to a process which, when implemented rigorously, would lead to reduced delays in product development and higher reliability of the product shipped.

ACKNOWLEDGMENTS

The authors would like to thank Professor Jens Palsberg for his review and comments on the first version of this work. J. Cangussu's research was financially supported by CAPES and Universidade Federal de Mato Grosso do Sul. A.P. Mathur's research was financially supported in part by the Software Engineering Research Centre (SERC) and the US National Science Foundation.

REFERENCES

[1] G. Cugola and C. Ghezzi, "Software Processes: A Retrospective and a Path to the Future," *Software Process Improvement and Practice,* 1999.

[2] G. Cugola, "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models," *IEEE Trans. Software Eng.*, vol. 24, no. 11, Nov. 1998.

[3] S.L. Pfleeger, *Software Engineering: Theory and Practice.* Prentice Hall, 1998.

[4] E. Kit, *Software Testing in the Real World: Improving the Process.* Addison-Wesley, 1995.

[5] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process.* Addison-Wesley, 1998.

[6] W.A. Florac and A.D. Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement.* Addison-Wesley, 1999.

[7] R.A. DeCarlo, *Linear Systems: A State Variable Approach with Numerical Implementation.* Upper Saddle River, N.J.: Prentice Hall, 1989.

[8] B. Friedland, *Advanced Control System Design.* Upper Saddle River, N.J.: Prentice Hall, 1996.

[9] H. Khalil, *Nonlinear Systems,* Upper Saddle River, N.J.: Prentice Hall, 1996.

[10] J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "Feedback Control of the Software Test Process through Measurements of Software Reliability," *Proc. 12th Int'l Symp. Software Reliability Eng.*, Nov. 2001.

[11] B. Marick, *The Craft of Software Testing,* Englewood Cliffs, N.J.: Prentice Hall, 1995.

[12] R.S. Pressman, *Software Engineering—A Practitioner's Approach,* McGraw-Hill, 1992.

[13] T.H. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.,* vol. 2, no. 6, pp. 308–320, 1976.

[14] B.W. Boehm, *Software Engineering Economics.* Prentice Hall, 1981.

[15] B.W. Boehm et. al., *Software Cost Estimation with Cocomo II.* Prentice Hall, 2000.

[16] D.G. Luenberger, *Introduction to Dynamic Systems: Theory, Models, and Applications.* John Wiley & Sons, 1979.

[17] K. Kanoun, M. Kaanicke, and J.C. Laprie, "Qualitative and Quantitative Reliability Assessment," *IEEE Software,* vol. 14, no. 2, pp. 77–87, Feb. 1997.

[18] N.F. Schneidewind, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics," *IEEE Trans. Software Eng.,* vol. 25, no. 6, pp. 769–781, Nov./Dec. 1999.

[19] A. Wood, "Predicting Software Reliability," *Computer,* vol. 29, no. 11, pp. 69–77, 1996.
[20] W.K. Ehrlich, J.P. Stampfel, and J.R. Wu, "Application of Software Reliability Modeling to Product Quality and Test Process," *Proc. Int'l Conf. Software Eng.,* pp. 108–116, 1990.
[21] G. Fulford, P. Forrester, and A. Jone, *Modeling with Differential and Difference Equations.* Cambridge Univ. Press, 1997.
[22] T. Abdel-Hamid and S.E. Madnickdecar, *Software Project Dynamics: An Integrated Approach.* Prentice Hall, 1991.
[23] C.L. Lawson and R.J. Hanson, *Solving Least Squares Problems.* SIAM, 1995.
[24] S.R. Lay, *Convex Sets and their Applications.* John Wiley & Sons, 1982.
[25] L. Ljung, *System Identification: Theory for the User.* Englewood Cliffs, N.J.: Prentice Hall, 1987.
[26] T.M. Khoshgoftaar and J.C. Munson, "Prediciting Software Development Errors Using Complexity Metrics," *IEEE J. Selected Areas in Comm.,* vol. 8, no. 2, pp. 253–261, 1990.
[27] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach.* PWS Publishing, 1997.
[28] D.E. Knuth, "The Errors of Tex," *Software—Practice and Experience,* vol. 19, no. 7, pp. 607–685, 1989.
[29] J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "A Formal Model for the Software Test Process," Technical Report SERC-TR-176-P, Purdue Univ., Mar. 2001.
[30] W. Gander, J.H. Masaryk, and J. Hrebicek, *Solving Problems in Scientific Computing Using Maple and MATLAB.* Springer-Verlag, 1997.
[31] E.K.P. Chong and S.H. Zak, *An Introduction to Optimization.* John Wiley & Sons, 1996.
[32] D.J. Wheeler and D.S. Chambers, *Understanding Statistical Process Control.* SPC Press, 1992.
[33] G.A. Hansen, "Simulating Software Development Processes," *Computer,* vol. 29, no. 1, pp. 73–77, Jan. 1996.
[34] A. Diagne, J.M. Ilie, and D. Moldt, "A Petri Net-Based Support for Object-Oriented Specification of Complex Control Systems," *Proc. Petri Nets in System Eng.: Modelling, Verification, and Validation,* Sept. 1997.
[35] M.C. Paulk et al., "Capability Maturity Model for Software," technical report, Software Eng. Inst., Carnegie Mellon Univ., 1993.
[36] J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "Sensitivity Analysis of a State Variable Model of the Software Test Process," *Proc. 2001 IEEE Systems, Man, and Cybernetics Conf. (SMC 2001),* pp. 712–717, Sept. 2001.

João W. Cangussu received the BS degree in computer science from the Federal University of Mato Grosso do Sul-Brazil in 1990 and the MS degree in computer science from the University of Sao Paulo at Sao Carlos-Brazil in 1993. He worked as an assistant professor at the Federal University of Mato Grosso do Sul from 1993 to 1997, when he joined the PhD program in the Computer Science Department at Purdue University, where he is currently a PhD candidate. His research interests are software process modeling, management and control, and software testing. He is a student member of the IEEE, the IEEE Computer Society, and the ACM.

Raymond A. DeCarlo received the BS and MS degrees in electrical engineering from the University of Notre Dame in 1972 and 1974, respectively. In 1976, he received the PhD degree under the direction of Dr. Richard Saeks from Texas Tech University. His doctoral research centered on the Nyquist Stability Theory with applications to multidimensional digital filters. He taught at Texas Tech for one year before becoming an assistant professor of electrical engineering at Purdue University in the Fall of 1977 and an associate professor in 1982. He worked at the General Motors Research Laboratories during the summers of 1985 and 1986. He is a fellow of the IEEE (1989), past associate editor for technical notes and correspondence and for survey and tutorial papers, both for the *IEEE Transactions on Automatic Control*. He is a former secretary-administrator of the Control Systems Society and a member of the Board of Governors of the society from 1986 through 1992. He was program chairman for the 1990 IEEE Conference on Decision and Control and was the general chairman of the 1993 IEEE Conference on Decision and Control. Currently, he is a member of the Board of Governors of the IEEE Control Systems Society (1999 to present) and also serves as the vice president for financial activities. He received a distinguished member award from the IEEE Control Systems Society in 1990 and the IEEE Third Millennium Medal in 2000. He has written three books (*Interconnected Dynamical Systems*, *Linear Systems: A State Variable Approach*, and *Linear Circuit Analysis: Time Domain*, *Phasor and Laplace Transform Approaches*) has more than 40 journal and 70 conference publications, and has contributed various book chapters. His research interests include modeling of computer processes, variable structure control of linear systems, nonlinear systems and decentralized systems, modeling the software development process, biomedical modeling and control, hybrid and discrete event systems, numerical linear algebra as applied to control and stability problems, decentralized control of large scale systems, analog and analog-digital fault diagnosis of circuits and systems, and digital and active filter design.

Aditya P. Mathur received the BS, MS, and PhD degrees in 1970, 1972, and 1977, respectively, from the Birla Institute of Technology and Science, Pilani, India. He is currently a professor of computer science at Purdue University. His research interests are in software testing and reliability, feedback control of software processes, the management of connected spaces, and music composition. He has published more than 100 research papers, written two books, and composed more than 40 pieces in a variety of genres. He is a fellow of the IEEE.