# A State Variable Model for fhe Software Test Process

João Cangussu [*]
Department of Computer Sciences - Purdue University
West Lafayette-IN 47907-1398, USA
Phone: (765)494-7812 Fax: (765)494-0739
cangussu@cs.purdue.edu

Raymond A. DeCarlo
Department of Electrical and Computer Engineering - Purdue University
West Lafayette-IN 47907-1285, USA
decarlo@ecn.purdue.edu

Aditya P. Mathur [†]
Department of Computer Sciences - Purdue University
West Lafayette-IN 47907-1398, USA
apm@cs.purdue.edu

**Abstract**

A novel approach to modeling the software development process is presented. This approach is based on the use of concepts and techniques from the theory of state variables and feedback control. The reasons to use this approach and its advantages are presented. A model for the Software Test Process is developed to show the approach applicability to the software development process. The assumptions and choice of parameters used in the model are discussed. Two sets of data are used to conduct a validation exercise. The first is the error log reported by Knuth during the development of TEX78 and the second is the data from a ongoing project in a company. The results of both exercises are reasonably accurate.

**Key Words:** software process model, software process control, software test process, state variable, feedback control.

[*]Financial support by CAPES and UFMS
[†]Financial support in part by SERC and NSF

## 1 Introduction

This paper develops a state model [3, 11] of the software test process (STP) and validates the model with data from a large industrial project supplied by Razorfish Corporation. The STP is one of the final phases of the software development process (SDP) studied quite extensively [2]. The work described here is related, among others, to the FEAST Project[9] , to System Project Dynamics [1], and to Statistical Process Control [5], although these endeavors do not focus in the STP, but rather consider the entire SDP.

State models [11, 3] are a special type of differential equation model well suitable to modeling processes and to control system analysis, design and optimization [3, 10, 11]. In this paper we adapt the modeling flexibility of the state model to capture the dominant behavior for the STP knowing that: (i)the large number of psychological variables associate with the test process and the intrinsic creative nature of the STP make it difficult to accurately predict the test process behavior; and (ii)more extensively modeling to account for more variables will be necessary in the future. By capturing the essential dynamical behavior of the STP with a state model, we can then apply the well developed theory of feedback control to guide the project manager in improving test process performance, meeting deadlines, etc. Specifically we focus on the control of the time and effort required to reduce the errors in a software product to a desired (minimal) level. Thus, upon the completion of the code and unit testing, any effort to test and debug the software product is considered part of the STP and our model. Our model does not account for error categorization such as specification or design errors, critical and non-critical errors [8, 13]. Fortunately, the modeling is made possible because the STP has several distinct but interconnected elements which can be characterized by a set of (at least) three assumptions or basic principles.

In summary this paper offers four contributions: (i) a set of three basic principles which govern the dominant behavior of the STP, (ii)the development of a quantitative (non probabilistic) state model for characterization of the STP, (iii)a validation of the model using real data from a large industrial project, and (iv)a development of a quantitative feedback mechanism for adjusting variables such as work force and quality level of the test process in order to meet managerial performance objectives.

The remainder of this paper is organized as follows. Section 2 presents the model basic principles (assumptions) and the state variable model based on principles. A discussion about paramerters choices and values is presented in Section 3. An application of our model is discussed in Section 4. Finally, in Section 5 we present conclusions and outline directions for future work in the area of process modeling using feedback control theory.

## 2 Modeling the Software Test Process

As stated before, our focus here is the modeling of the test phase using differential equations as a first step in constructing a state model. The input to the test phase is a software product with a known estimate of inherent errors. During the testing process errors are found and removed. The output of the test phase is a transformed version of the input software product.

A number of variables and parameters are specific to the test phase. They include but are not limited to:

1. $(r)$ - the number of remaining errors.
2. $(w_f)$ - work force, in manpower.
3. $(s_c)$ - overall software complexity.
4. $(t)$ - time in appropriate units of days, weeks, months, etc.
5. $(\gamma)$ - a constant characterizing the overall quality of the testing process.
6. $(e_{et})$ - effective test effort.
7. $(e_r)$ - error reduction resistance.

Here $s_c$ represents a general measure of complexity instead of a specific one. In that way, our model will be more flexible and will not be restricted to a predetermined metric. An organization using the methodology described here can calibrate the model according to their data and metric. For example, an organization can use program size; Cyclomatic Complexity [12] or a combination of both to determine the overall software complexity measure $s_c$. See Section 3.

The way the test phase is conducted has a great impact on the results. The coefficient $\gamma$ will be used to characterize the overall quality of the testing process and it will represent environmental factors such as deadline pressure, applied methodology, organizational aspects, experience and expertise of the work force and possibly other factors. A guideline to compute $\gamma$ is also provided in Section 3.

The effective test effort ($e_{et}$) represents how much of the work force effort is translated into decreasing the number of errors. For example, in any organization, the work force spends time on ancillary tasks such as administrative and reporting responsibilities, and learning new but pertinent software tools. Such tasks decrease the effective test effort.

The variable $e_r$, error reduction resistance, is important because as errors are removed, new errors are sometimes inserted and in this case effort in an opposite direction of the test effort is generated.

Using these variables, we state the first of three critical assumptions in our modeling strategy.

**Assumption 1**: The magnitude of the rate at which the remaining errors are decreasing is proportional to the net applied effort for the test phase and inversely proportional to the software complexity.

In equation format, Assumption 1 says that:

$$\ddot{r} = \frac{e_n}{s_c} \quad \Rightarrow \quad e_n = \ddot{r}\ s_c \tag{1}$$

where $\ddot{r}$ is the second derivative of r and $e_n$ is the net applied effort.

The first assumption is easily justifiable. When the same metric or combination of metrics is used to compute software complexity for two different softwares, one can always expect that more effort will be necessary to test the more complex one. If, for example, Cyclomatic Complexity [12] and LOC are used to determine $s_c$, a larger program with more regions will definitely require more test effort than a smaller program with a small number of regions.

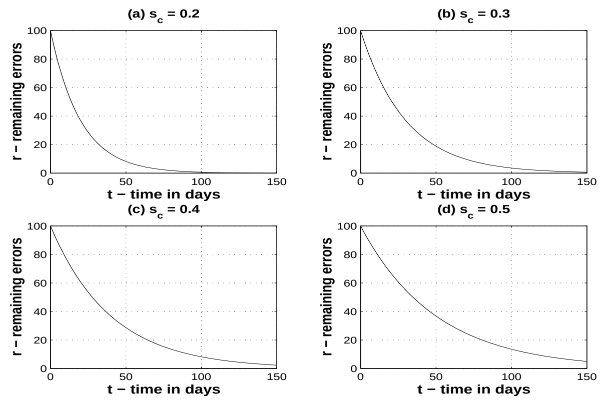

Figure 1: Behavior of remaining errors versus time

The net applied effort ($e_n$) is the balance of all the effort produced during the test phase. The variable $e_n$ results from the difference of the effective effort of the work force minus any "frictional" forces that decreases the applied effort. Since r represents the number of remaining errors,the first derivative $\dot{r}$ is an error reduction velocity ($v_e$). Consequently $\ddot{r}$ representing the rate of change of $\dot{r}$ is an acceleration

Thus, the concepts of velocity and acceleration can be applied to the test phase. Equation 1 is analogous to Newton's second law of motion for physical systems. Here $e_n$ is analogous to force, $\ddot{r}$ is analogous to acceleration, and $s_c$ analogous to mass. The larger the mass, the greater the force needed to move the mass at a desired velocity. In the software world, the greater the software complexity ($s_c$), the larger must be the effort ($e_n$) to remove errors at a certain error reduction velocity ($\dot{r}$). As stated before, we are not concerned here about which metric will be used to generate the coefficient of the software complexity, but we know it can be calibrated according to the metric used by a specific organization.

At the beginning of the test phase errors are easy to find and they became more difficult to find as r decreases. Supposing the work force is constant, this implies that the effective test effort is bigger for larger r. This observation suggests Assumption 2 and equation 2.

**Assumption 2**: The magnitude of the effective test effort is proportional to the product of applied work force and the number of remaining errors, i.e., for an appropriated $\zeta$:

$$e_{et} = \zeta\ w_f\ r \tag{2}$$

This assumption can be justified through the same arguments used in the predator-prey system described by Volterra [11]. In this system, the number of possible encounters equals the product of the numbers in the predator and prey population and each encounter decreases the prey population according to some decline rate dependent on the species. 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. The parameter $\zeta$ defines the decline rate and according to Ehrlich [4] it will decrease as r gets smaller($\zeta = \zeta(r)$). Thus we expected $\zeta$ to decrease as the process continues. That is, as the test process goes on, the errors become more difficult to find, not only because there are less errors, 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 will be captured by changes in $\zeta$ over periods of the STP. The behavior of assumption 2 is similar to the rate of decreasing of errors [14] when software reliability models are applied to the STP [4].

The effective test effort is opposed by a force intrinsic to the test phase process. This force is called error reduction resistance and is denoted by $e_r$.

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

The equation equivalent to Assumption 3 is

$$e_r \quad = \quad \xi\ \frac{1}{\gamma}\ \dot{r} \tag{3}$$

for an appropriate constant $\xi$.

The assumption above can be justified by analyzing its behavior under extremal conditions. For example, a very low quality will induce a large resistance: $\gamma \ \rightarrow\ 0 \quad \Rightarrow \quad e_r \ \rightarrow\ \infty$. The same is true for $\dot{r}$ values: the higher $\dot{r}$, the higher the error resistance $e_r$.

Also, if errors are removed in a fast way the probability that more errors are inserted is higher, essentially opposing error reduction. If the quality of the test phase is high ($\gamma \ \rightarrow\ 1$) the resistance will decrease but still it will be present, consistent with equation 3. In the case where $\dot{r} \ \rightarrow\ 0$, the resistance will also decrease. This behavior is present under various circumstances: errors are being removed in a slow careful way and so the resistance is small; or the applied effort is small and so is the resistance.

Assumption 3 represents the relation between how fast errors are removed and the oscillations (insertion of new errors) caused by it. A dashpot is a natural analog of this behavior. The coefficient of viscosity of the liquid inside the dashpot is equivalent to $\frac{1}{\gamma}$. Therefore, a small coefficient of viscosity (which implies a high quality) means that the test phase is conducted in a smooth and careful way and in this case the number of new errors inserted is small. If we have a large coefficient of viscosity (small quality) then the test phase has been conducted in a careless way and more errors than normal are inserted. The velocity component in the dashpot can be related to the error reduction velocity ($\dot{r}$). Thus, the overall test phase quality ($\gamma$) and the rate errors are found will determine the error reduction resistance which is analogous to the damping force generated by the dashpot in a physical system. In equation 3 $\xi$ is just a constant of proportionality.

A force balance equation applied to the various effort yields

$$e_{et}\ -\ e_r\ =\ e_n \tag{4}$$

Combining equations 1 to 3 produces the following second-order differential equation :

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

The minus sign for the first term is due to the fact that $e_{et}$ is a restoring force and must have the opposite sign to r. The second term also has a minus sign because as a "damping" force, $e_r$ is always opposite in direction to the velocity. Since $\dot{r} < 0$ on average because in our coordinate system we are applying a restore force creating a velocity with a negative direction, then, $|e_n|\ <\ |e_{et}|\ \ for\ \dot{r} < 0$, i.e.,

$$|-\zeta\ w_f\ r\ \ -\xi\ \frac{1}{\gamma}\ \dot{r}|\ \ <\ \ |-\zeta\ w_f\ r| \tag{6}$$

Now, let $r_0$ denote the number of error remaining in the product at time $t = t_0$, i.e. at the start of the test phase, and the initial error reduction velocity, $v_e = 0$. The solution for equation 5 is:

$$r(t) = \frac{r_0\lambda_2}{\lambda_2 - \lambda_1}e^{-\lambda_1 t} + \frac{r_0\lambda_1}{\lambda_2 - \lambda_1}e^{-\lambda_2 t} \tag{7}$$

In equation 7, $\lambda_1$ and $\lambda_2$ denote the distinct roots of the characteristic equation [3, 11]. The fact that we have two negative distinct roots is due to the assumption of an overdamped process. If underdamping were allowed, r would reach negative values. This makes no sense in the software world. The overdamping requirement will restrict the values of $\gamma$ to less than $\frac{\xi}{2(s_c\ w_f\ \zeta)^{\frac{1}{2}}}$ [7].

The behavior of Eq. (5) is depicted in Figure 1 where we can observe the results for: different levels of system complexity, i.e., $s_c = 0.2$, 0.3, 0.4 and 0.5; initial number of remaining errors of 100 ($r_0 = 100$); and initial error reduction velocity of 0 ($v_e = 0$). As expected , the number of remaining errors decreases slower as system complexity increases, assuming that the overall quality ($\gamma$) and the work force ($w_f$) are constant.

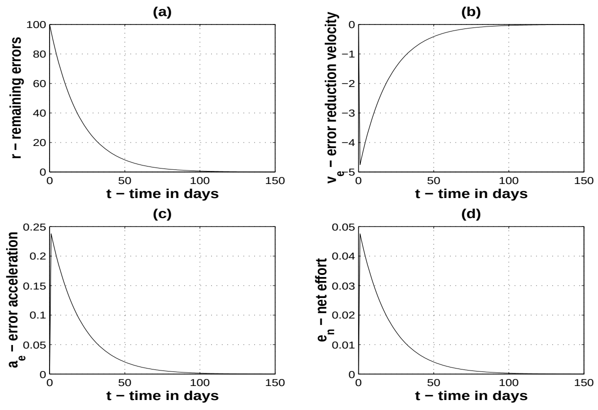

Figure 2: Results for $s_c = 0.2$, $\gamma = 0.005$ and $w_f = 2$

From Figure 2 we observe how error reduction velocity and acceleration error change as the remaining errors decrease. At the start of the test phase errors are easier to find, and therefore the velocity is relatively high. As the test phase continues error detection gets increasingly difficult and hence the velocity decreases until it gets close to zero, as can be observed in Figure 2b. Therefore, we have a deceleration until the probability of finding a new error becomes almost zero as shown in Figure 2(c). As expected according Assumption 1, the net applied effort approaches zero (as $t \to \infty$), as depicted in Figure 2(d).

In our initial model we consider only two forces acting on the software product: the effective test effort ($e_{et}$) and the error resistance ($e_r$). However, as is known there are other forces that affect the product during the test phase. For example, it is wise to include in the model an auxiliary effort when a test tool is being used and also an opposite force when the test team needs to learn the use of the tool. Other forces, also not considered in this paper, include a communication effort and an adaptation effort.

## 2.1 State Model for the STP

State Model relates the input vector to the state vector to produce the output [3]. The state vector $x$ is the set of internal variables required to produce the output. For the test phase, we have two state variables: the remaining errors ($r$) and the error reduction velocity ($v_e$). It is easy to define the first derivative of the state variables as a function of the state variables. We have the following relation:

$$\ddot{r} = -\frac{\zeta\ wf}{s_c} r - \frac{\xi}{\gamma\ s_c}\dot{r} + \frac{1}{s_c} F_d \tag{8}$$

The force $F_d$ represents a disturbance [1] during the test phase and feedback will be used to minimize this disturbance and produce the expected output in the appropriated time, if this task is feasible. That is, suppose a hard disk failure prevents the test to be conducted for two days. This "disturbance" will cause a decreasing in the error reduction velocity. Taking r and $\dot{r}$ as state variables, we arrive at the controllable canonical state model form [3].

$$\dot{x} = Ax + Bu \Rightarrow \begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{\zeta\ w_f}{s_c} & -\frac{\xi}{\gamma\ s_c} \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{s_c} \end{bmatrix} F_d \tag{9}$$

$$y = Cx \Rightarrow \begin{bmatrix} r \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} \tag{10}$$

The use of feedback to adjust the eigenvalues of the A-matrix and hence modify the overall performance is taken up in Section 4.5 in the context of the Razorfish project.

## 3 Parameter Guideline

All models depends on parameters whose quantification are critical for the model to accurately predict process behavior. Parameters for models of the SDP are often intuitively or empirically derived. Different organizations often have different metrics for various parameters. The purpose of this section is briefly provide some guidelines for parameters choices.

Work force ($w_f$) is simply the number of tester per time unit. On the other hand software complexity is a difficult variable to quantify. Among many metrics available for software complexity one can consider: Lines of Code (LOC), Cyclomatic Complexity, Syntactic Complexity, and Halstead Difficult metric. Specifically one could consider the number of grammar productions as a complexity metric in a compiler project, as in the Razorfish project presented in Section 4. Rather than pick up a single metric we define the software complexity metric as a convex combination: $s_c = \sum_{i=1}^{n} \alpha_i \; M_i$. Where $M_i$ is a specific normalized software complexity metric and $\sum_{i=1}^{n} \alpha_i \; = \; 1$.

The quality of the test process ($\gamma$) has no widely accepted metric. Features essential to $\gamma$ can be defined in a per project/company basis and a convex combination can again be used to provide a quantification of $\gamma$. We are basically replacing the complexity metrics for $s_c$ by quality features in the convex combination approach. Other parameters in the model depends on deadlines and desired error reduction but the discussion of their quantification is beyond the scope of this paper.

## 4 Model Validation

Two validation exercises were conducted in our work. The first one uses data loged by Knuth during the development of TEX78 and the second one uses data from an ongoing commercial project underway at Razorfish, a company located at New York, NY. We focus, in this paper, only in the Razorfish project.

### 4.1 Project Description

Razorfish currently has an application that contains about 4 million lines of code in COBOL. We will refer to this application as $S_{Cobol}$. This application is to be transformed into a functionally equivalent application in SAP/R3 hereafter referred to as $S_{SAP\ R/3}$. Razorfish is developing a tool, hereafter referred to as *transformer*, to automate this transformation. The information presented here about the project was obtained through interviews with the project manager, developers and the test team. Razorfish maintains data on all errors found in the *transformer*, 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 us.

In order to present the results of the *transformer* project we need to obtain an estimate of the initial number of errors and the values for the parameters used in the model. The next two sections will discuss these issues.

### 4.2 Estimate of the initial number of errors

We assume that an initial estimation of the number of errors present in a project has to be available to start using our model. Software Reliability models can be applied to the STP in order to obtain this initial estimate [4]. One can always argue that a initial estimate is not always precise enough and can compromise the use of our model. Therefore, we present below a method to improve the accuracy of the initial estimate or to provide the initial estimate in case one is not available.

As we assume an exponential decay in the number of errors present in a program, we use the equation $r \; = \; (t_e \; + \; \Delta)e^{\frac{-t}{\lambda}} \; - \; \Delta$ to approximate the data from the first 15 weeks of test, where $t_e$ is the total number of errors in this period and $\Delta$ is some constant. Computing the derivative of this equation gives the approximate error reduction velocity ($v_e$). Using the data provided by Razorfish and the approximated $v_e$ we compute the error difference for a specific period of time (one week in this case) as $D^i \; = \; r(i) \; - \; r(i-1)$. In terms of our state model $D^i \; = \; M \; D^{i-1}$, where $M \; = \; e^{A \; T}$, $T$ is the time increment between two consecutive measurements of data, and A is the desired matrix in equation 9. Therefore, in the *transformer* project we can compute $M$ by

$$R_1 \; = \; M \; R_2 \;\; \Longrightarrow \;\; M \; = \; R_1 \; {R_2}^{-R} \tag{11}$$

where $R_1 = [\ D^{15}\ D^{14}\ D^{13}\ ...\ D^4\ D^3\ ]$; $R_2 = [\ D^{14}\ D^{13}\ D^{12}\ \dots\ D^3\ D^2\ ]$ and ${R_2}^{-R}$ is the right inverse of $R_2$ [11].

We used the solution of equation (11) to compute a estimate of the initial number of errors for the *transformer* project . That is, we know that $D^2 = (M^2 - M)x(0)$ and therefore $x(0) = (M^2 - M)^{-L}\, D^2$. Using the computed values of $M$ and $D^2$ we obtained an initial condition of $r_0$ errors which was considered reasonably good by the project manager. The nominal value of $r_0$ is the proprietary data of Razorfish and so is presented here in a normalized version ranging from 0 to 100, which can be thought is as percentages. The same approach was used for TEX78. That is, using data from the first 15 testing days we predicted $r_0 = 419$ errors which represents 78% of the total number of errors (533) found over the life of TEX78. Considering that only 239 errors were found during TEX78 test phase, our prediction represents an improvement of 34% over this value.

## 4.3 Parameter Choices

In this section we discuss how the parameters were computed to model the project described above.

Software complexity ($s_c$) is computed using the convex combination approach described on Section 3. To measure the complexity of the tool under development we decided to use two size oriented metrics $M_1$ and $M_2$. $M_1$ is the number of ten's of KLOC. The *transformer* has approximately 135,000 lines of code resulting in $M_1 = 13.5$. $M_2$ is based on the number of grammar productions used to specify the COBOL syntax. Every 10 productions will increase the software complexity by one. The *transformer* is designed to deal with different versions and dialects of COBOL. This requirement increased the language specification and complexity significantly. The COBOL specification has around 1,580 productions resulting in $M_2 = 158$. Using the convex combination of $M_1$ and $M_2$, and setting $\alpha_1 = 0.8$ and $\alpha_2 = 0.2$ we obtain $s_c = 42.4$. Here we assume that KLOC account for 80% of the complexity measure

The second parameter to be determined is $\gamma$. The testers at Razorfish have significant experience in testing similar systems. The test team also uses a test tool that increases the quality of the test phase. Upon fixing the errors a regression test is carried out to determine if any new errors have been introduced.

In accordance with the project manager we divided the first 15 weeks of the test phase into three periods. The first period is composed of the first 6 weeks of the test phase. We also decided to use four features to define the overall quality of the test phase ($\gamma$): (i)experience/expertise of $w_f$, (ii)tool use and adequacy of the tool, (iii)test plan adequacy, and (iv)quality of the test cases. The details of the choices are beyond the scope of this paper. It should be clear that the features has to be defined in a project/company basis and not as a standard for all projects/companies.

A quantification of $\gamma$ was determined using convex combination and the four features above. For period 1 we compute $\gamma = 0.55$. This value for $\gamma$ is due to the fact that the tool was not as useful as it is latter in the project because they were testing screen conversions and the generated layout could not be checked automatically by the tool. Also, they were using an *in vitro* data base to test the *transformer* and so the quality of the test cases were not completely satisfactory. The second period (weeks 6 to 10) showed improvement 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 tool adequacy is also present. Thus, for this second period we compute $\gamma = 0.6$. The third period corresponds to weeks 10 to 15 and was demarcated from the previous phase by the fact that test data from a “large” company using the native COBOL application became available and the system could be exercised more completely. As in the second period, utilization of the tool increased. Thus, for the third period, we compute $\gamma = 0.75$. The values for work force experience/expertise and the test plan adequacy are the same for all periods since there were no changes in the test team and the test plan was followed and seemed to be quite appropriate for the *transformer* project.

The size of the test team remained constant at 3 testers for the period under consideration. This led us to $w_f = 3$. The value of $\xi$ is set to a constant value of 100 in order to normalize the results of equation 3. Parameter $\zeta$ determines the decline rate of errors and is computed based on the expected behavior. The entire test phase is estimated to last 25 weeks and to reach this goal $\zeta$ is set to 0.4 for the first period and will decrease to 0.35 for the following two periods.

In Figure 3 we can see the expected behavior for the *transformer* project plotted using the parameters described in the previous section. We can also see that the observed behavior diverges from the expected one for the first 15 weeks of the project. This divergence is due to disturbances present during the STP and alternatives to correct its effect, using feedback, is discussed later.

A disturbance is a force opposing the effective test effort ($e_{et}$). We are not concerned here in the identification of sources of disturbance but in the quantification of the disturbance itself. The disturbance is computed by taking the expected behavior and introducing an opposite force ($F_d$) that will produce the observed behavior (real data).

## 4.4 Project Results

At the time of writing this paper, the *transformer* project was in its $15^{th}$ week. We used our model to understand the behavior of the test process during the first 15 weeks and to predict the behavior for the remaining weeks.

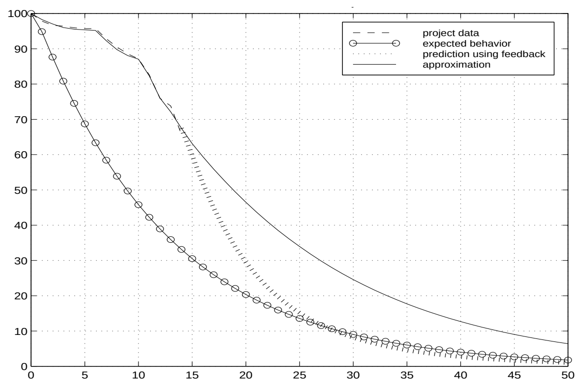

Figure 3: Behavior of the COBOL Transformer Project

Figure 3 depicts the test phase results of the COBOL Transformer project. As can be observed, the approximation obtained from the use of our model presents reasonable results. This is based on the integral mean square error which produced a 2.85 error norm when data from the real project is compared to the model approximation for the first 15 weeks.

All parameters values used to produce Figure 3 can be seen in Table 1. These parameters and the disturbance inserted during the process produce the approximation depicted in Figure 3.

Table 1: Parameters values used in Figure 3

| three segments local approximation | | | | | | | |
|---|---|---|---|---|---|---|---|
| | weeks | $\gamma$ | $\xi$ | $\zeta$ | $s_c$ | $w_f$ | $F_d$ |
| Period 1 | 1 to 6 | 0.55 | 100 | 4.0 | 42.4 | 3 | 80% |
| Period 2 | 6 to 10 | 0.60 | 100 | 3.5 | 42.4 | 3 | 65% |
| Period 3 | 10 to ... | 0.75 | 100 | 3.5 | 42.4 | 3 | 25% |

where $F_d$ is the average disturbance in the period

When analyzing the results presented in Figure 3 two considerations can be done. First, it can be considered that any error in the *Transformer* will produce an error in the $S_{SAP\ R/3}$ generated code, i.e, a one by one relation is assumed. A second consideration can be that not all errors in the *transformer* will affect this specific project, i.e., the transformation from $S_{COBOL}$ to $S_{SAP\ R/3}$ will not be able to exercise all features of the *Transformer*. The second consideration implies that by the end of the test phase (25 weeks) some errors will still be present in the *transformer* but the goal of the project will be achieved, that is, the generated system will be functionally equivalent to the original COBOL system and project success is ensured.

## 4.5 Feedback Application

Our model predicts that it will not be possible to finish the project by the expected deadline as one can see when comparing our approximation with the expected curve. What changes can be done in the STP in order to accomplish the deadline? Feedback will help us to answer this question. By week 15 the fraction of errors dropped to around 67.5% and if no change is done it will take around 35 weeks to reach the expected level of error reduction (approximately 14.7 %). Suppose the project manager desires to achieve the same results in only 10 weeks. What "feedback" modifications are necessary?

The largest eigenvalue of a system determines the slowest rate of convergence and dominates how fast the variables converges. Therefore, we need to adjust the largest eigenvalue of the model so that the responses converge to the desired values within the remaining weeks.

Equation 12 below can be used to achieve this goal.

$$r(T) \; - \; r(T + \Delta t) \;\; = \;\; r(T) \; e^{-\lambda_{max} \; \Delta t} \tag{12}$$

where r(T) is the number of remaining errors at timer T, $r(T + \Delta t)$ is the desired value for r after a lapse of $\Delta t$ time has occurred, and $\lambda_{max}$ is the eigenvalue to be computed. In the Razorfish project we want the system to converge from 67.5% at week 15 (r(15)=67.5%) to 14.7% at week 25 (r(15+$\Delta t$)=14.7%) for $\Delta t = 10$. Solving equation 12 for these values results in $\lambda_{max} = 0.152$.

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

$$det[\lambda I \; - \; A] \;\; = \;\; det \begin{bmatrix} \lambda & -1 \\ \frac{\zeta \; \hat{w_f}}{s_c} & \lambda + \frac{\xi}{\hat{\gamma} \; s_c} \end{bmatrix} \; = \; \lambda^2 \; + \; \frac{\xi}{\hat{\gamma} s_c}\lambda \; + \; \frac{\zeta \hat{w_f}}{s_c} \tag{13}$$

where $\hat{\gamma} = \gamma + \Delta_\gamma$ and $\hat{w_f} = w_f + \Delta_{w_f}$.

To set the eigenvalue of the model described by equations 9 and 10 to 0.1522 we have to make changes in the values of these parameters. Considering that no changes can be done in $\xi$, $\zeta$ 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 ($0 \; < \; \Delta_\gamma \; < 0.25$).

Varying $\Delta_{w_f}$, keeping all other values constant and then finding the roots of the characteristic polynomial produces the results depicted in Figure 4(a). We can observe that $\lambda_{max}$ reaches the desired value of -0.152 when $\Delta_{w_f}$ reaches 2.5. This implies that the $w_f$ has to be increased by 2.5 in order to accomplish the deadline, assuming all other parameters are kept constant. The feedback result of increasing the work force by 2.5 is presented in Figure 3. Figure 4(b) presents similar results achieved from the variation of $\Delta_\gamma$. As can be observed, it is not possible to reach the deadline by only increasing the quality of the test phase. Therefore, an increase in the work will be necessary, even if we consider that $\gamma$ can achieve the highest level.

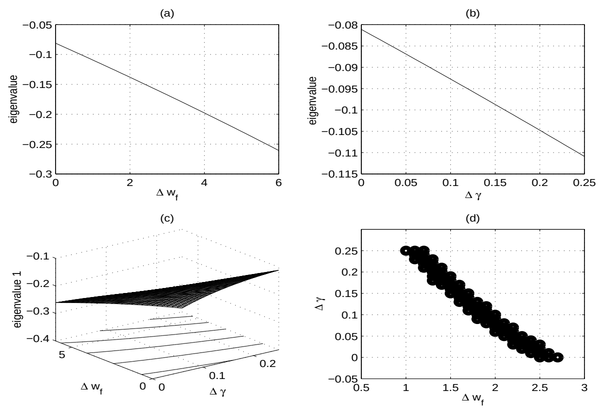

Figure 4: $\gamma$ and $w_f$ related to eigenvalue

Figure 4(c) presents the combined results when $\Delta_\gamma$ ranges from 0 to 0.25 and $\Delta_{w_f}$ ranges from 0 to 6. A project manager can use these results when more alternatives are available, such as a deadline extension. In the *transformer* project we are interested in finishing the test phase within the predetermined deadline and all possible combinations of increasing $w_f$ and/or $\gamma$ to accomplish this task are depicted in Figure 4(d).

The model can also be used to analyze different alternatives according to flexibility of the deadline and resource availability. That is, if 2.5 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 basically an exercise of maximizing customer satisfiability according to resource limitations. As stated before, optimization techniques are available in Control Theory [6] to provide these results, but it is beyond the scope of this paper.

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 cost of the STP.

## 5 Conclusions and Future Work

It is perhaps the creativity involved and the dynamic behavior of the SDP that have led to the avoidance of applying a formal and classical method to model it. However, these methods have been successfully applied to model other systems that possess characteristics similar to those of the SDP. This fact encouraged us to apply a state variable approach to model the SDP starting from the STP.

The model for the test phase presented in this paper was analyzed through a validation exercise conducted using the data logged by Knuth when working on $T_EX_{78}$. The results obtained from this validation are reasonably accurate as is evident from the integral mean square error. A second validation exercise using data from a large industrial project with a multi work force effort also presented reasonable results helping us to understand the model behavior and exercise prediction aspects of the model in a ongoing project. The behavior of our current model for the test phase enhances our belief that the application of state variable theory is appropriate and likely to result in an improved understanding of the changes during the software test process.

The approach presented here can be used to improve performance, maintenance and controllability of the STP. Techniques associated with control theory provide the tools to do so. 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 shipped product.

Several aspects of modeling and the application of control theory in the STP are not covered here. Some are not covered because we are considering a simple model to make the initial task easier to apply and understand; others are not covered because we are not considering all phases of the SDP. Besides the improvement of the current model, the use of calibration by System Identification techniques [10] and the optimization aspects are also on our agenda for the future.

## References

[1] T. Abdel-Hamid and S. E. Madnick. *Software Project Dynamics: an Integrated Approach.* Prentice Hall Software Series, New Jersey, 1991.

[2] G. Cugola and C. Ghezzi. Software processes: A retrospective and a path to the future. *Software Process Improvement and Practice*, 1999.

[3] R. A. DeCarlo. *Linear systems : a state variable approach with numerical implementation.* Upper Saddle River, New York: Prentice-Hall, 1989.

[4] W. K. Ehrlich, J. P. Stampfel, and J. R. Wu. Application of software reliability modeling to product quality and test process. In *Proceedings of ICSE*, 1990.

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

[6] B. Friedland. *Advanced control system design.* Upper Saddle River: Prentice-Hall, 1996.

[7] G. Fulford, P. Forrester, and A. Jone. *Modeling with Differential and Difference Equations.* Cambridge University Press, Cambridge, United Kingdom, 1997.

[8] D. E. Knuth. The errors of tex. *Software-Practice and Experience*, 19(7):607–85, 1989.

[9] M. M. Lehman. Feedback in the software evolution process. In *CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics*, Dublin, 1994. also in Information and Software Technology,sp. iss. on Software Maintenance, vol. 38, n. 11, 1996.

[10] L. Ljung. *System identification: theory for the user.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[11] D. G. Luenberger. *Introduction to Dynamic Systems: theory, models and applications.* John Wiley & Sons, 1979.

[12] T. H. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(6):308–320, 1976.

[13] D. E. Perry and C. S. Steig. Software faults in evolving a large, real-time system: a case study. In *4th European Software Engineering Conference – ESEC93*, Garmisch, Germany, September 1993.

[14] N. F. Schneidewind. Measuring and evaluating maintenance process using reliability, risk, and test metrics. *IEEE Trans. on Software Engineering*, 25(6):769–781, 1999.