A State Model for the Software Test Process with Automated Parameter Identification

João W. Cangussu[1], Ray A. DeCarlo and Aditya P. Mathur[2]

Department of Computer Sciences, Purdue University
Department of Electrical and Computer Engineering, Purdue University
Department of Computer Sciences, Purdue University
West Layette, IN, USA, 47907-1398.

Abstract
A model is proposed to assist software test managers in controlling the behavior and progress of the Software Test Process (STP) by allowing them to compare predicted behavior against observed progress made at various checkpoints. The model, whose parameters are based on measured data and process characteristics, generates the predicted behavior. An algorithm for the parameter estimation is set forth. The error between the predicted and desired behavior is used to drive a parametric control algorithm that tells the manager how to correct for schedule deviations.
keywords: Software process, parameter estimation, modeling, software test process, state model.

1 Introduction

Project managers are always facing problems in the control of phases of the Software Development Process (SDP). Two major management objectives are the accurate prediction of completion time and the cost of a specific project. Further, managers want solutions for deviations from these pre-specified objectives. This paper focuses on these problems in the context of the Software Test Process (STP), a sub-phase of the SDP. Many techniques having been proposed to address these problems with partial success. That is, models are available to make initial predictions or to answer "what if" questions [1]. So far no model for the STP has a closed feedback solution to correct deviations as the STP moves forward. Also, the wide variability and qualitative approaches in parameter estimation is a major concern when using such models because in some cases they are based on empirical data and are not re-calibrated on a per project/company basis.

In this paper we present a model that goes beyond answering "what if" questions by providing a space of solutions to correct deviations in the process. A technique to automatically calibrate parameters dependent on the process behavior is presented. Also presented is an approach to compute parameters dependent on the specific software product and test team features. A technical report providing more complete details of the modeling process is [2].

The remainder of this paper is organized as follows. Section 2 presents the assumptions and corresponding justifications upon which the state model builds. The approaches to compute the parameters characterizing the STP are described in Section 3. The results of a case study using data from a large industrial project are presented in Section 4. Finally, Section 5 draws the conclusions of this work.

2 A State Model for the STP

2.1 Parameters of the STP

The dominant behavior of the STP can be captured by relating variables and parameters intrinsic to the STP. The relations are established through key assumptions (Section 2.2). These parameters and variables are: (i)r - the number of remaining errors; (ii)w_f - size of the test team; (iii)s_c - program complexity; (iv)t - time measured in appropriate units; and (v)γ - a constant characterizing the overall quality of the test process.

As stated before, the parameters above are

[1]Financial support by CAPES and UFMS
[2]Financial support in part by SERC and NSF

used to relate elements acting on the STP. Two elements or forces are identified as having a great impact on the STP. The first is the effective test effort (e_f) that represents how much of the effort spent by the work force is translated into the actual removal of errors. The error resistance (e_r) is the second element and represents an opposing force related to the quality of the process. The balance of the forces acting on the process leads to the net applied effort (e_n). The assumptions describing each of these forces are presented next.

2.2 Key assumptions

The three key assumptions about the STP are presented and justified below. 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.

Assumption 1: *The rate at which the speed of decrease of the remaining errors changes is directly proportional to the net applied effort and inversely proportional to the complexity of the program under test, i.e.,*

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

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

The first assumption is justified as follows. When the same metric or 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 [3] and LOC are used to determine s_c, a larger program with more regions will likely require more test effort than a smaller program with a small number of regions.

The net applied effort (e_n) is the balance of all the effort applied during the test phase. This results from the difference of the effective effort applied by the test team minus any "frictional" forces that decrease 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. Thus $\ddot{r}$ (an acceleration) times s_c (the mass of the software product) equals the net applied effort.

It is widely believed that the difficulty of 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 Assumption 2.

Assumption 2: *The effective test effort is proportional to the product of the applied work force and the number of remaining errors, i.e.,*

$$e_f = \zeta\ w_f\ r \tag{2}$$

for an appropriate ζ.

Justification of this assumption follows by analogy with the predator-prey system described by Volterra [4]. Here, the decline in the prey population is proportional to the number of possible encounters between the predators and the prey. In Assumption 2 the probability of finding an error is proportional to an encounter between a tester and an error. The tester plays the predator role and errors are the prey. There are $w_f \times r$ possible encounters. 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 that this combination will be discovered by the testers only in the final phases of testing, if it is discovered at all. This behavior is partially captured by changes in ζ over different periods of the STP. The behavior of Assumption 2 is similar to the rate of decrease of errors [5] when software reliability models are applied to the STP [6].

The effective test effort is opposed by a force intrinsic to the STP. This force is the error reduction resistance, e_r. This observation leads to the last of the three assumptions.

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

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

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

The 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 values of $\dot{r}$: the larger $\dot{r}$, the larger the error resistance e_r.

This assumption implies that the faster one tries to reduce the remaining errors, the more likely one is to make mistakes which slows the entire process. A physical dash-pot illustrates such behavior. The coefficient of viscosity of the liquid inside the dash-pot is $\frac{1}{\gamma}$. Therefore, a small coefficient of viscosity is analogous to the test phase being conducted in a smooth and careful way, i.e. high quality. A larger coefficient of viscosity is analogous to a low quality. The velocity component in the dash-pot is analogous to the error reduction velocity ($\dot{r}$). Thus, the overall quality of the test phase, denoted by (γ), and the rate at which errors are found, determines the error reduction resistance effort which is analogous to the damping force generated by the dash-pot. In Eqn. 3, ξ is an appropriate constant of proportionality.

The net applied effort, e_n, of assumption 1 comes from a force balance equation that uses assumptions 2 and 3:

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

where e_f has a negative sign because it opposes the increase in errors. Replacing e_f, e_r and e_n by their values from Eqns. 1, 2 and 3 results in the following second-order differential equation:

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

As in our coordinate system we are applying a restoring force creating a velocity with a negative direction. Thus we find on the average that $\dot{r} < 0$. Hence, $|e_n| \; < \; |e_f|$ for $\dot{r} < 0$. This leads to the following equation:

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

which legitimizes the sign choices of Eqn. 5.

Using Eqn. 5 and r and $\dot{r}$ as state variables produces the following state model for the STP.

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} \; = \; \begin{bmatrix} 0 & 1 \\ -\frac{\zeta \; w_f}{s_c} & -\frac{\xi}{\gamma \; s_c} \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{s_c} \end{bmatrix} F_d \tag{7}$$

where F_d represents a disturbance such as hardware failure.

3 Parameter Estimation

3.1 Estimation of ξ, ζ, and r_0

The algorithm, described later in this section, is used for computing ξ, ζ, and r_0 for data from the current project. Obviously, this data is not available initially. However, a manager may have data from past similar 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 Eqn. 7 has the general form of $\dot{x}(t) \; = \; Ax(t)$, where $x(t) = [r(t) \; \dot{r}(t)]^T$ and A is the proper 2×2 matrix. It is well known that the solution of this state model is given by $x(t) = e^{At}x(0)$ for all $t \geq 0$. To compute ξ and ζ we need to compute the matrix A, from which we can obtain ξ and ζ as all the other parameters are know 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, i.e., project data consists of $d^{(k)} \equiv r(kT_1) - r((k-1)T_1)$. Although $r(t)$ has the general form specified by a double exponential solution, we first use a single exponential approach to obtain a local approximation for $\dot{r}(t)$, i.e., we locally approximate $r(t)$ as

$$r(t) = \alpha e^{-\lambda t} \tag{8}$$

Hence for 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:

$$\left[d^{(k)} \; d^{(k-1)} \; ...d^{(3)}\right] = m \left[d^{(k-1)} \; d^{(k-2)} \; ...d^{(2)}\right]$$

in which case

$$m = \left[d^{(k)} \; d^{(k-1)} \; ...d^{(3)}\right] \left[d^{(k-1)} \; d^{(k-2)} \; ...d^{(2)}\right]^{-R}$$

and α can be computed as

$$\begin{aligned} \alpha \; &= \; \left[(m^2 - m) \; (m^3 - m^2) \; ... \right. \\ &\qquad \left. (m^n - m^{n-1})\right]^{-L} \left[d^{(2)} \; d^{(3)} \; ...d^{(n)}\right] \end{aligned}$$

where superscript $-R$ and $-L$ represent the Moore-Penrose pseudo right and left inverses, 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 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 $D^i = M\ D^{i-1}$, where $M = e^{A\ T_1}$, T_1 being the time increment between two consecutive measurements of data, and A the A-matrix of equation 7. Therefore, we can compute M as follows

$$R_1 = M\ R_2 \implies M = R_1\ {R_2}^{-R} \qquad (9)$$

where $R_1 = [\ D^n\ D^{n-1}\ D^{n-2}\ ...\ D^4\ D^3\ ]$; $R_2 = [\ D^{n-1}\ D^{n-2}\ D^{n-3}\ \ldots\ D^3\ D^2\ ]$.

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: ${\lambda_1}^M = e^{\lambda_1 T_1}$ and ${\lambda_2}^M = e^{\lambda_2 T_1}$. Therefore $\lambda_1 = \frac{1}{T_1} ln({\lambda_1}^M)$ and $\lambda_2 = \frac{1}{T_1} ln({\lambda_2}^M)$. Since ξ and ζ are the only unknown at this time, we can compute them by matching the roots of the characteristic polynomial $\Pi_A(\lambda)$ to λ_1 and λ_2 computed above.

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

$$P = \begin{bmatrix} M^2 - M \\ M^3 - M^2 \\ \vdots \\ M^n - M^{n-1} \end{bmatrix}, \quad Z = \begin{bmatrix} D^2 \\ D^3 \\ \vdots \\ D^n \end{bmatrix}$$

$$\text{and } x_0 = \begin{bmatrix} r(0) \\ \dot{r}(0) \end{bmatrix}$$

We know that $Z = Px_0$ and we can compute $x_0 = P^{-L}Z$. 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 [8]:

$$\begin{aligned} Z &= PWx_0 \Rightarrow \\ x_0 &= ((PW)^T(PW))^{-1}(PW)^T Z \end{aligned} \qquad (10)$$

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 $\frac{1}{\sigma}$ [8] 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 as more data becomes available. This will make the value of the weights decrease when the opposite behavior 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. This idea derives from the exponential weighted moving average [9].

3.2 Software complexity

There is no widely accepted metric to compute the complexity of a program. Therefore, any of the existing complexity measures could be used to obtain a value for s_c. Our model does not prescribe any specific complexity measure. For example, program size, cyclomatic complexity [3] or a combination of both could be used to compute s_c.

Instead of trying to define a new metric to be used with our model, we decided to use convex combination of existing metrics providing an opportunity to simultaneously use multiple metrics for software complexity [10].

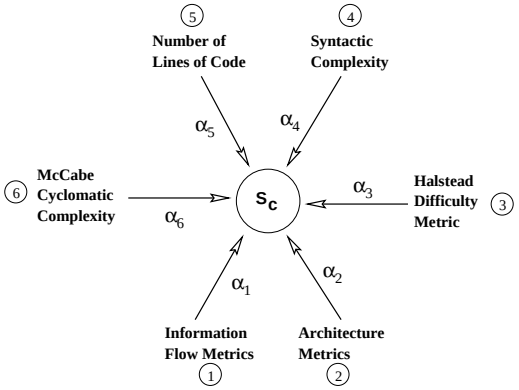

Figure 1: s_c as a convex combination of six software complexity metrics. α_i's are the corresponding weights.

Figure 1 lists six different metrics useful in computing software complexity. The test manager must choose the weights α_i in order to estimate the value of the parameter s_c. Two of these six metrics are based on structural properties of programs, three on program size, and one is a combination of the other five. A software complexity metric based on information flow (metric # 1) is defined by Henry and Kamura [11] to capture the relation between procedure size and information flowing into (fan-in) and out (fan-out) of procedures.

Syntactic Complexity (metric # 4) is defined

by Basili [12] based on the attributes product size, control paths, and product decomposition. These attributes are combined to produce the definition of a family of control structure based complexity metrics. Halstead [13] (metric # 3) defined components of software science and program difficulty (D) as a measure of software complexity. The Cyclomatic Complexity (metric # 6) defined by McCabe [3] is based on the number of regions contained in the directed graph of the program. For a large program an average of the Cyclomatic Complexity per method can be used. The number of lines of code (metric # 5) is defined directly and simply as a size relation proportional to complexity.

Based on the six metrics mentioned above, software complexity (s_c) is defined as a convex combination

$$s_c = \sum_{i=1}^{n} \alpha_i \; M_i \tag{11}$$

where M_i is a normalized software complexity metric and $\sum_{i=1}^{n} \alpha_i \; = \; 1$.

The software complexity ranges from a lower bound of 0 to an upper bound of ϕ. Parameter calibration techniques [14] can be used to define the upper bound ϕ for individual organizations.

3.3 Quality of the Process

The coefficient γ characterizes the overall quality of the test process and represents environmental factors such as pressure due to deadline, test methodology used, structure of the organization within which testing is carried out, experience and expertise of members of the test team, and possibly other factors. Although a single coefficient is unable to fully represent the quality of the testing phase, when appropriately chosen it appears to be adequate. The same observations about the computation of s_c are also pertinent to the quality of the process (γ). Therefore we decided to use the convex combination approach to compute γ. Work force experience and expertise, test strategy/adequacy, tool use/adequacy, adequacy of the test plan, and coverage criteria are some examples of quality aspects that can be used in the computation of γ.

3.4 Size of the test team

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, any variation in w_f is accounted for by computing the state variables over successive periods.

4 Case Study

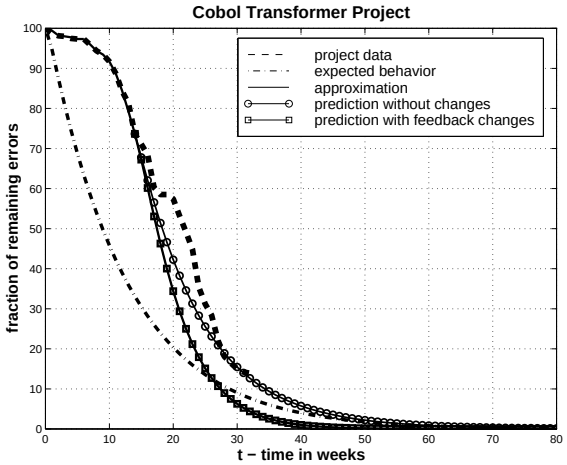

Figure 2: Actual and predicted behavior of the COBOL Transformer project observed in the change in the fraction of remaining errors.

The case study presented in this section uses data from a commercial project at Razorfish, a company located at Cambridge, MA. Razorfish currently has an application that contains about 4 million lines of code in COBOL. This application is to be transformed into a functionally equivalent application in SAP/R3 and a tool was developed to automate this transformation. The results of the application of our model to this project are presented in Figure 2.

The values of the parameters used to obtain the results of Figure 2 are listed in Table 1, where F_d is the average disturbance in the period, P1 represents the period for weeks 1 to 6, P2 is the period from weeks 7 to 10, P3 is the period from weeks 11 to 14, and P4 is the remaining time period. These values were computed using the techniques described in Section 3. A high disturbance can be observed in the first period. This value is due to the discover of a major error that prevent the continuation of the test process as expected. As more data became available and the parameters were re-calibrated, the disturbance force decreased for periods 2, 3 and 4. At week 14, we predicted that it will not be possible to finish the project within the expected time (25 weeks). Our prediction was that it would take 35 weeks to complete the project if no changes were made to accelerate it. Despite this, the project manager opted

for no changes in the process and project was completed in 37 weeks. The prediction for completion and also a 95% accuracy in the prediction of r_0 show the precision of the model. The actual number of errors is the proprietary data of Razorfish and is presented here in a normalized fashion.

Table 1: Parameters values used in Figure 2.

Parameter	P1	P2	P3	P4
γ	0.44	0.56	0.75	0.75
ξ	19.57	22.73	18.86	18.86
ζ	0.25	0.54	0.75	0.75
s_c	48	48	48	48
w_f	3	3	3	3
F_d	62%	36%	17%	25%

5 Concluding Remarks

The Software Development Process (SDP) has a degree of controllability that is far behind the degree achieved by physical systems. A great effort has been done to increase the controllability of the SDP. By focusing on a specific phase of the SDP, the STP, the model presented in this paper is the first step towards the achievement of this goal. The state model for the STP provides a feedback mechanism to help the test manager to correct deviations found during the STP. The feedback mechanism consists of determining the minimum decay rate needed to meet management objectives followed by adjustments in w_f and/or γ to achieve the necessary eigenvalues of A.

The rigorous, but not formal, aspects of many approaches used to model the SDP makes the calibration of the parameters of the model a difficult task. In some techniques the calibration is done in a "trial and error" approach. As described in this paper, the use of a state variable approach allows the application of System Identification techniques with algorithms to automatically calibrate some parameters of the model. The other parameters are computed using a convex combination of known software metrics. Thus, the increase of the degree of controllability combined with the automated calibration of the parameters makes the state model proposed here a promising approach. This fact is supported by the reasonable results obtained from the case study.

References

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

[2] J. W. Cangussu, R. DeCarlo, and A. Mathur, "A formal model for the software test process," Tech. Rep. SERC-TR-176-P, Purdue University-SERC, March 2001.

[3] R. S. Pressman, *Software Engineering - A Practitioner's Approach*. McGraw-Hill International Editions, 1992.

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

[5] K. Kanoun, M. Kaanicke, and J.-C. Laprie, "Qualitative and quantitative reliability assessment," *IEEE Software*, vol. 14, no. 2, pp. 77–87, 1997.

[6] 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.

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

[8] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics (SIAM), 1995.

[9] J. S. Oakland, *Statistical Process Control*. Butterworth Heinemann, fourth ed., 1999.

[10] S. R. Lay, *Convex Sets and their Applications*. New Yor: John Wiley & Sons Inc., 1982.

[11] S. Henry and D. Kafura, "The evaluation of software systems' structure using quantitative software metrics," *Software Practice and Experience*, vol. 14, no. 6, pp. 561–573, 1984.

[12] V. R. Basili and D. H. Hutchens, "An empirical study of a syntatic complexity family," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 664–672, 1983.

[13] M. H. Halstead, *Elements of Software Science*. New York: Elsevier North-Holland, 1977.

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