

## A quantitative Learning Model for Software Test Process

Ghaffari Abu  
Department of Computer Science  
University of Texas at Dallas  
Richardson, TX - USA 75083-0688  
gabu@utdallas.edu

João W. Cangussu  
Department of Computer Science  
University of Texas at Dallas  
Richardson, TX - USA 75083-0688  
cangussu@utdallas.edu

Janos Turi  
Department of Mathematical Sciences  
University of Texas at Dallas  
Richardson, TX - USA 75083-0688  
turi@utdallas.edu

### Abstract

*Learning through experience shows improvement in productivity. Many models and approaches related to learning or experience curve have been established and successfully applied to the traditional industries. This paper investigates the learning process and extends the idea of improvement through learning to software test processes. A novel quantitative learning model for software life cycle is proposed and compared with the existing learning models. An existing formal software test process model is modified to include effects of learning based on the developed learning model. Finally, the extended quantitative software test process model is applied to several industrial software test projects to validate the improved prediction capabilities of the model.*

### 1 Introduction

The state variable approach has shown considerable potential in modeling of the Software Test Process (STP)<sup>1</sup> [1, 2, 3]. The main advantage of this framework arises from the use of control theory to control, optimize and calibrate the model in a per-company/per-project basis. Formal approaches make the modeling process difficult, however the quantitative results obtained from such models are more accurate and practical than the current ad-hoc industrial prac-

tices. The first level model<sup>2</sup> [3] for STP did not account for important implicit elements affecting the STP. Sensitivity analysis [4] suggests that learning and communication overhead are examples of two such elements.

In this paper, an investigation of learning phenomenon during STP is presented and a novel model is proposed to account for learning behavior during software processes. The effect of learning on STP is then analyzed and quantitatively evaluated by introducing learning parameters into the current state model of STP. As described in Section 6, this analysis shows enhancement in the accuracy of the existing STP model.

Incorporation of continuous quantities such as learning and experience into the existing STP model has brought a quantitatively unique and significant improvement in the model. Unlike discrete event simulation models, present model not only goes beyond “what if?” questions but applies closed loop feedback and control mechanism. While formal process languages such as Little-Jil [5] are limited to procedural (coordination in the process steps) aspects of the process, our model at the very least does provide quantitative estimation of the STP metrics. Present model can be used as complementary model to process models like Little Jil where the quantitative analysis is not available. The well proven feedback and control approach accompanied by quantitative recommendation to managers is the major strength of this model. Feedback solution in the present model is closed loop and hence provide the opportunity to the project managers to target/achieve the modified qual-

<sup>1</sup>Software Test Process hereafter will be referred as STP

<sup>2</sup>Original STP Model over which the current improvement has been applied.

ity objective of the project by making controlled changes in the model parameters similar to the system dynamic models [6, 7].

The remainder of this paper is organized as follows. Section 2 presents an overview of the first level state variable model. In Section 3, a variety of currently available learning models are compared and the behavior of software learning is inferred from that in Section 4. In Section 5 the state model is refined and new features are inserted into the model. A time variant system approach for the inclusion of learning in the model is also presented. A validation exercise of the new model is the subject of Section 6. Section 7 finishes the paper presenting the conclusions of this work.

## 2 Overview of Existing STP State Variable Model

The linear deterministic model of the STP is based upon three assumptions. These assumptions and the corresponding equations are presented below [2, 3]. The model has been validated using sets of data from testing projects and also by means of an extremal case and a sensitivity analysis [4].

**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 ( $s_c$ ) of the program under test, i.e.,*

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

**Assumption 2:** *The effective test effort ( $e_f$ ) is proportional to the product of the applied work force ( $w_f$ ) and the number of remaining errors ( $r$ ), i.e., for an appropriate  $\zeta(s_c)$ ,*

$$e_f(t) = \zeta(s_c) w_f r(t) \quad (2)$$

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.

**Assumption 3:** *The error reduction resistance ( $e_r$ ) opposes and is proportional to the error reduction velocity ( $\dot{r}$ ), and is inversely proportional to the overall quality ( $\gamma$ ) of the test phase, i.e., for an appropriate constant  $\xi$ ,*

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

Combining Eqs. 1, 2, and 3 in a force balance equation and organizing it in a State Variable format

( $\dot{x} = Ax + Bu$ ) [8, 9] produces the following system of equations.

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-\zeta w_f}{s_c^{(1+b)}} & \frac{-\xi}{\gamma s_c} \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{s_c} \end{bmatrix} F_d \quad (4)$$

$F_d$  above is included in the model to account for unforeseen disturbances such as hardware failures, personnel illness, team dynamics, economic or any event that slows down or even interrupts the continuation of the test process.

Along with the model in Eq. 4 an algorithm, based on system identification techniques [10], has been developed to calibrate the parameters of the model [2]. Using data available and an estimation of  $\dot{r}$  [3] the error difference for a specific period of time  $D^i = [r(i) \ \dot{r}(i)]^T - [r(i-1) \ \dot{r}(i-1)]^T$  is computed. It can be shown that  $D^i = M D^{i-1}$ , where  $M = e^{AT}$ ,  $T$  being the time increment between two consecutive measurements of data. Therefore, matrix  $M$  can be computed from Eq. 5

$$R_1 = M R_2 \implies M = R_1 R_2^{-R} \quad (5)$$

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

The Spectral Mapping Theorem [8] 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 Eq. 4, which are

$$\begin{aligned} \pi_A(\lambda) &= \det[\lambda I - A] \\ &= \lambda^2 + \frac{\xi}{\hat{\gamma} s_c} \lambda + \frac{\zeta w_f}{s_c^{(1+b)}} \end{aligned} \quad (6)$$

Since  $\xi$  and  $\zeta$  are the only unknowns at this time, they can be estimated by matching the roots of  $\pi_A(\lambda)$  to  $\lambda_1$  and  $\lambda_2$  computed above.

The fast convergence presented by the algorithm increases the model applicability and accuracy [11]. Finally, a parametric control procedure is used to compute required changes in the model in order to converge to desired results according to time constraints [3].

Several case studies were successfully conducted and additional ones are still being conducted in a large corporation. The outcomes of these test processes are predicted, by calibrating parameters governing the behavior of the process, with outstanding accuracy<sup>3</sup>.

## 3 Learning Process

Improvement in quality and hence in the productivity with an increase in knowledge and experience has been

<sup>3</sup>Due to proprietary reasons these results can not be disclosed

observed traditionally in a variety of systems and processes [12, 13, 14, 15]. Software development and testing processes have been also observed to follow the traditional rule of “Increased productivity with improved learning” [16]. However, improvement in learning with time for software systems and its effect on the Software Test Processes during the software life cycle has not been completely investigated. A detailed description of the existing learning models is provided next. A novel model to characterize the learning behavior during the STP is proposed and applied to the STP model. The new model uses the foundational work observed in the existing techniques.

### 3.1 Log Linear Model

Phenomenon of learning curves was first observed by Wright [14]. He observed that the direct labor hours/unit decreases at uniform rate as the number of units produced doubles. This led to the derivation of the most commonly used Log Linear Model [14, 15, 17, 18, 19]. Two variations of this model are available, the first is based on unit cost [18] and the second on cumulative average cost [17] of the product. According to the unit cost model, the cost of production of the  $x^{th}$  unit is the product of the theoretical production cost of the first unit and the  $n^{th}$  power of  $x$  as described in Eq. 7. The theoretical cost is an estimate of the production cost of the first unit.

$$y_x = y_1 x^n \quad (7)$$

where  $y_x$  is the production cost of the  $x^{th}$  unit,  $y_1$  is the theoretical production cost of the first unit, and  $n$  is a constant reflecting the change in cost rate.

Cumulative Average Cost model is the most common approach toward learning. The cost is averaged over  $x$  units instead of the  $x^{th}$  units as represented in Eq. 8.

$$y_{avg\ x} = y_1 x^n \quad (8)$$

where  $y_{avg\ x}$  is the average cost of  $x$  units,  $y_1$  is the production cost of the 1<sup>st</sup> unit(theoretical), and  $n$  is a constant reflecting the change in cost rate. The typical behavior for both models is linear if plotted on the log-log scale and the slope of the curve is given by the rate constant  $n$ . The results of the application of a Log Linear Model are shown in Figure 1.

### 3.2 Stanford-B Model

The Stanford-B Model [17, 19] is based on the assumption that experience is carried over to the next project/production. Hence, Stanford-B model is applied where experience carries over from one production run to another production run. In the case of the STP, this could

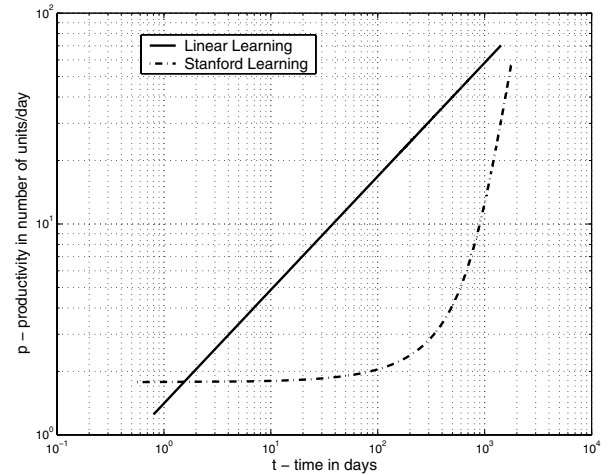


Figure 1. Log Linear & Stanford learning model

mean the experience is carried over when testing the next release of the software product or even for another software. For a new release of most of the projects, the tools, platform, and language used for testing remains unchanged and hence the experience of the tester certainly improves the productivity relative to a new tester. The Stanford-B Model is represented in Eq. 9 and its learning pattern is shown in Figure 1.

$$y_x = y_1 (x + B)^n \quad (9)$$

where  $y_x$  is the production cost of the  $x^{th}$  unit,  $y_1$  is the theoretical production cost of the 1<sup>st</sup> unit,  $B$  is a constant reflecting the experience at the beginning of next release, and  $n$  is a constant reflecting the change in cost rate.

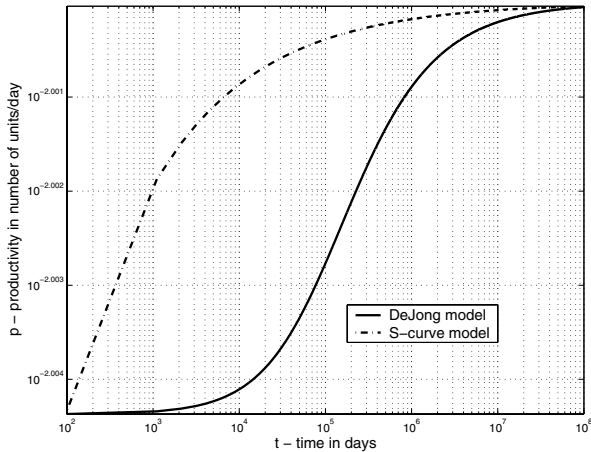
### 3.3 DeJong Model

The DeJong model [17, 19] assumes that a portion of process cannot be improved, i.e. there is a productivity upper limit for the automated production processes. Such behavior is observed in situations where assembly lines ultimately limits the improvement.

The DeJong model is represented in Eq. 10 and the learning pattern described by the model is shown in Figure 2.

$$y_x = y_1 + D x^n \quad (10)$$

where  $y_x$  is the production cost of the  $x^{th}$  unit,  $y_1$  is the theoretical production cost of the 1<sup>st</sup> unit,  $D$  is a constant reflecting the limiting nature of the process, and  $n$  is a constant reflecting the change in cost rate.



**Figure 2. DeJong & S-curve learning model**

### 3.4 S-curve Model

The S-curve [17, 19] model presented in Figure 2 is based on the assumption derived from the combination of Stanford-B model and DeJong model. This Model assumes that the experience carries over to the next process and an improvement in each stage of the process is not possible. This model is a good candidate for developing the learning pattern in the software industry as explained in Section 4. The behavior of this model is actually exponential as expressed in Eq. 11, however it appears S-shaped when plotted on log-log scale.

$$y_x = y_1 + D(x + B)^n \quad (11)$$

where  $y_x$  is the production cost of the  $x^{th}$  unit,  $y_1$  is the theoretical production cost of the 1st unit, the experience factor  $B$  is a constant reflecting the experience at the beginning of next release, the incompressibility factor  $D$  is a constant reflecting the limiting nature of the process, and  $n$  is a constant reflecting the change in cost rate.

## 4 Characterizing Learning in the STP

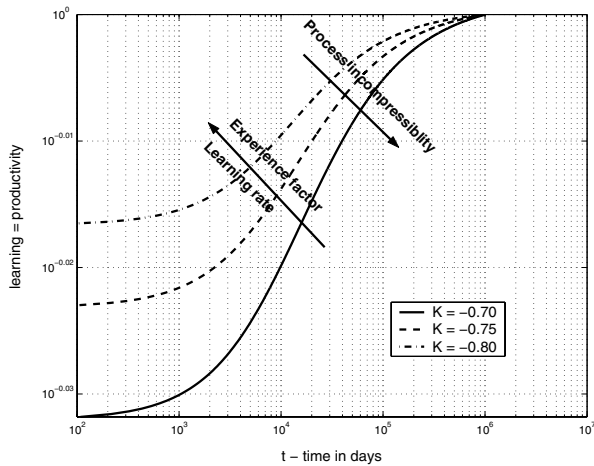
The performance in all stages of the software life cycle is dependent on the experience of the software engineers involved on the development process. An experienced software engineer performs better irrespective of the stages of the software life cycle. The domain knowledge, programming skills, and testing ability carries over to the next project or next release. With regard to the STP, this reflects that the knowledge acquired during a testing process can be carried over to the next release or project. On the other hand for a given software, due to the limited resources (workforce, tools adequacy, etc), the performance of a portion of a process cannot be improved over a certain

limit. If at least one hour is required to verify the results of one test case, we cannot expect that a tester can find more than 8 defects in a regular day of work. Under these considerations, the behavior of the S-curve model [17] appears to be the best alternative to represent the learning process for the STP. The saturation can be reached later within the same release or even in a future release of the product. Such behavior has also been reported for the software development process [20]. The learning behavior and its effect on the STP process can be essentially described initially as gradual increase and later a sharp growth leading to certain saturation value. The S-curve model suites the requirements and is adopted here as the expansion of the state model of the STP.

Learning<sup>4</sup> derived from the S-curve model by taking the inverse of the  $Y_x$ , as obtained from Eq. 11, is plotted against the time in Figure 3. The figure also shows the effect of changing the components of the S-curve model. The starting point is dependent on the initial learning  $Y_1$  and shifts upward with increase in initial knowledge. The Learning grows faster with increasing learning rate ( $n$ ) and experience factor ( $B$ ) as shown in the plot. This indicates that an experienced tester or testing team and a fast learner would gain knowledge faster and supports the idea of skilled staff doing better on the project. Such improvement in skills due to experience has unlimited potential of learning and effectively reducing the cost. On the other hand, increasing the process incompressibility factor ( $D$ ) implies increasing a portion of the process that cannot improve and hence limiting the growth in learning. Such behavior is illustrated by downward shifts (slower rate) of the S-curve model with increasing incompressibility as shown in Figure 3. This captures the behavior during the life cycle of a project. The improvement in the testing process itself is insignificant or limited over the life cycle of the project.

The existing S model as described by equation 11 shows a hyperbolic curve when plotted on linear scale, but when compressed to log-log scale takes the form of S curve. The complete nature of the S curve evolves over longer period of time compare to any software project. This is inherent with traditional manufacturing industries as the production is truly repetitive in nature. Unlike software industries, bringing a change in the manufacturing process requires large amount of resources and time commitment. On the other hand two software products are not an exact replica of each other even if they are two different releases of the same software product. Also the production time for software is much smaller than manufacturing industries in which case same product is continuously produced for long period of time. Any change in the process is difficult to attain and takes much longer than ever changing software development process. Hence the behavior of our learning model

<sup>4</sup>Productivity measured as number of units/day is used as Learning



**Figure 3. Learning characteristics of the Software Test Process derived from the S-curve Model.**

as described by the equation 13 is inherently S shaped as shown in Figure 4. The rate of improvement in productivity as observed in such curves are comparatively much slower than the software industry where the increased productivity or Learning<sup>5</sup> may go at much faster rate ( $K=0-1$ ) based on the previous experience, existing tools and other such established parameter discussed in Section 5. Also, the productivity for a given project and given span of project time has been assumed to saturate relatively much faster based on the initial learning and learning rate. The traditional S curve is not supposed to reach the peak as there is always room for improvement but pragmatically it is assumed to reach the saturation after an extremely long time.

## 5 State Variable Model Improvement

Adaptation effort has shown a large impact in the outcomes of a STP [4]. Such an aspect is now inserted into the STP state model to increase its applicability and completeness related to the test phase. When a project is starting or new people are inserted later on the project, some effort will be spent in adapting them to the project. The work force may have to adapt to the programming language or testing tool to be used. They will also have to learn about the project to be tested. Even if they are experts on the language and testing tool used and have a good knowledge about the project they will have to spend some time planning the test phase. The amount of effort to accomplish this task depends on work force and project attributes. Assumptions 4 and 5 below address these issues.

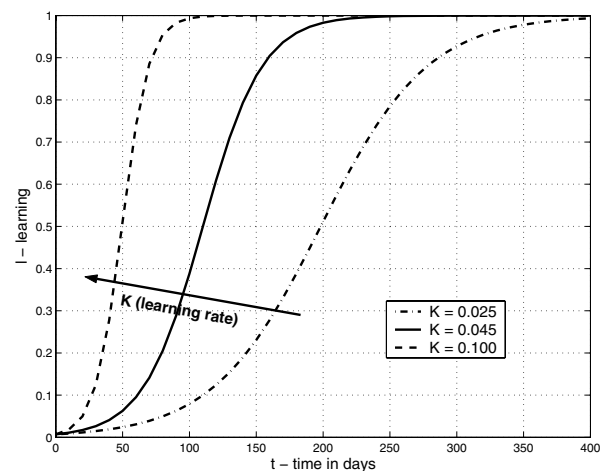
<sup>5</sup>Learning and productivity will be used synonymously hereafter

**Assumption 4:** Adaptation effort is a fraction of effective test effort  $e_f$ .

$$a_e = (1 - l) e_f \quad (12)$$

where  $l$  represents the learning aspect and ranges from 0 to 1.

Assumption 4 is based on the fact that errors cannot be detected during adaptation effort. The effective effort  $e_f$  represents the effort done by the test team in detecting errors. Part of this effort is invested into learning or adapting to the test process or the product under test. Therefore,  $a_e$  represents a fraction of  $e_f$  and is less than  $e_f$ . The adaptation effort slows down the process and this deceleration is determined here by the learning aspect  $l$ . A fast learning process means less effort is consumed in adaptation.



**Figure 4. Effect of learning rate (K) on learning**

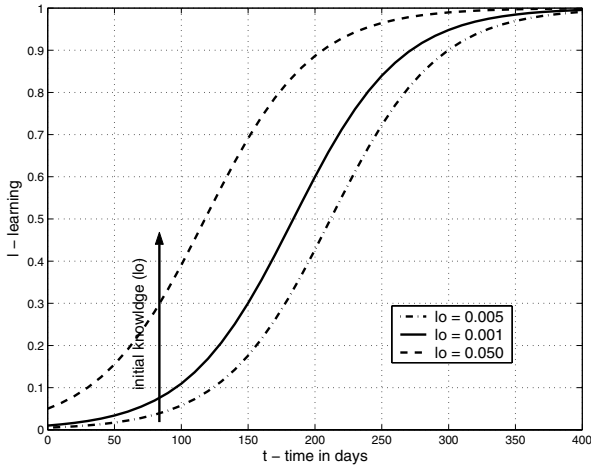
Autonomous change in knowledge with time is inevitable in any given context. It is a common observation of isolated experience becoming obsolete and regular experiences being escalated in daily life. However, the rate of such changes is governed by the existing knowledge e.g. for example in case of a software developer with knowledge of one object oriented programming language such as C++ can understand another object oriented programming language like Java much faster than a developer with no knowledge of any object oriented language. On the other hand, the amount of knowledge you can gain for a particular project during its development is limited and this notion of limited increment to a maximum value under the given condition has been captured and defined as the "Saturation Level". These simple but universally observed behaviors lead us to assumption 5.

**Assumption 5:** The rate of change in knowledge is proportional to the product of the existing knowledge ( $l$ ), the

knowledge remaining to be acquired ( $1 - l$ ), i.e. for a proportionality constant  $K$  representing the learning rate,

$$\dot{l} = Kl(1 - l/s) \quad (13)$$

where  $s$  is the saturation level of knowledge and can be normalized to a value of 1 as the maximum knowledge for the given software and time period limited by the project schedule.  $\dot{l}$  is the rate of learning



**Figure 5. Effect of initial knowledge ( $l_0$ ) on learning**

Figure 4 shows the learning characteristics against the time spent on the project as derived from Eqn. 13. The S-shaped behavior of the learning curve is governed by the existing knowledge, the value of  $K$  and the remaining knowledge to be acquired to achieve the saturation.

Figure 4 shows that the higher the value of  $K^6$  the higher the slope coefficient and hence the faster the learning. Higher value of  $K$  means increased learning and that means the faster achievement of the saturation level. On the other hand a better knowledge of the project at the very beginning of the project also leads to faster learning and hence shifts the learning curve up as shown in the Figure 5.

### 5.1 Determination of Initial Parameters

Determination of the initial knowledge and rate of learning for the individual team member and of the entire team is very critical. These parameters have to be estimated based on various abstract factors such as skills of the personnel, resources (tools, techniques, programming language, debugger, etc.) available for the testing process, complexity of the project, and stages of the project. Most organizations constantly keep evaluating or assessing their personnel for the skills and ability. This could be one of the

<sup>6</sup>The value of the constant  $K$  lies between 0-1.

possible sources of information for a manager to determine individual knowledge at the beginning of the project. Individual learning rates can also be determined based on previous performance of the employee. For any software project, before starting the test process, parameters could be recognized and evaluated for their impact on the process. A weighed quantitative value ( $\alpha$ ) could be associated with each selected parameter according to their importance to the test process. Such parameters could be, for example, product, tools, language, and process knowledge. The

weight of the parameters adds to 1 ( $\sum_i^n \alpha_i = 1$ , where  $n$  is the number of selected parameters). Such characterization and weighted parameters can be carried out at the planning stages. This would also help in selecting the team members and qualifying them against the chosen parameters. The team members can also be rated on certain scales based on their ability against the chosen parameters and associated with a factor( $f$ ). Eq. 14 can be used to get individual initial knowledge by taking a convex combination of individual skills and knowledge. The weight here represents how much knowledge is pertinent to the chosen parameter for the project. The initial knowledge of individual members can be weighed ( $\mu$ ) relative to other individuals in the team and then combined according to Eqn. 15 to arrive at the initial knowledge for the entire team. Initial learning rate ( $k$ ) of an individual has to be determined based on his performance in previous projects and can be assessed by the manager. Eqn. 16 provides a convex combination method to achieve the initial learning rate ( $K$ ) for the team based on individual learning rates ( $k$ ).

$$l_{o_j} = \sum_i \alpha_i f_i \quad (14)$$

$$l_0 = \sum_j \mu_j l_{o_j} \quad (15)$$

$$K = \sum_j \mu_j k_{o_j} \quad (16)$$

Addition of new members in the team would require assessment of initial knowledge and effective learning rate of the new members, impact of new members and adjustment in impact of old members resulting in a new overall  $K$ . All the parameters can be estimated in the same way as it was done during the beginning of the project except the effective learning rate of the individual member. New member can be added at any time of the project, the time elapsed does have significant effect on the ability of new members' learning rate. If a new member is added in the very later stages of the project, the overall process doesn't incur significant effect. However, overall learning will be effected according to the initial knowledge and learning rate of the new member. On the other hand new member addition to the team at the

beginning of the project will manage to achieve the overall learning efficiency and the overall learning rate will not be affected as much.

Combining equations from assumptions 1 to 5 produce the following system of differential equations:

$$\dot{r} = f_1(r, \dot{r}, l) = \dot{r} \quad (17)$$

$$\ddot{r} = f_2(r, \dot{r}, l) = -\frac{\zeta w_f}{s_c^{1+b}} r l - \frac{\xi}{\gamma s_c} \dot{r} \quad (18)$$

$$\dot{l} = f_3(r, \dot{r}, l) = kl(1 - l/s) \quad (19)$$

The above system is a nonlinear system due to the interdependency of state variables  $r$  and  $l$  in the second term of function  $f_2$ . Two alternatives are possible to avoid the complexity of using a nonlinear system. The first alternative is the linearization of the system by approximating the system to a Jacobian matrix according to Liapouov's indirect method [9]. However the equilibrium points<sup>7</sup> obtained by this method are either unstable or there are no conceivable effect of learning on  $\dot{r}$  or on  $\ddot{r}$ . These results led us to search another alternative to incorporate learning in the state model of the STP. Second alternative is the use of a time variant system as described in the next section.

## 5.2 A Time Variant System of the STP

The time dependent coefficient in the state variable model ( $\dot{x} = A(t)x + Bu(t)$ ) provides another alternative to introduce time variant learning behavior into the STP model. The state variables are the remaining errors  $r(t)$  and the error reduction velocity  $\dot{r}(t)$  since they are sufficient to model the dominant dynamics of STP and also serve as the output variables of interest [3]. System of Eqs. 17- 19 can be reduced to a 2 dimensional time dependent system by solving Eq. 19 for  $l$  and substituting it into Eq. 18. Hence, the following controllable canonical modified state model results from equations 17, 18, 19.

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{\zeta w_f}{s_c^{1+b}} l(t) & -\frac{\xi}{\gamma s_c} \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{s_c} \end{bmatrix} F_d \quad (20)$$

The A-matrix multiplying the vector of state variables is time dependent. It contains learning parameter which is assumed to follow the S-shaped behavior as described in Section 4. A trivial observation of  $\dot{r}(0) = 0$  and an estimate of the initial number of remaining errors  $r(0)$  at time  $t=0$  are required to describe the initial state of the model [3, 2]. The linear time invariant STP system has been well described elsewhere [3, 4]. However, the STP system in Eq. 20 is

<sup>7</sup>An equilibrium point of a dynamic system is the system state which never changes i.e. once system state reaches this point then it remains there for all future time.

Constituents	Weightage	Tester1	Tester2	Tester3
Product	0.5	0.03	0.06	0.05
Tools	0.1	0.03	0.06	0.05
Languages	0.2	0.03	0.06	0.05
Process	0.2	0.03	0.06	0.05
individual Learning →	initial	0.041	0.054	0.058
individual impact →		0.5	0.25	0.25
individual Learning Rate →	initial	0.03	0.01	0.02

**Table 1. Parameter values used in the validation of the learning model for the *Transformer* project**

time variant due to the presence of time dependent learning parameter in the A-matrix.

## 6 Improved Model Validation

Improvement due to Learning in the existing software test process were modeled and validated against several projects. However, only the results of the *Transformer* project are presented here. The model was also applied to data collected by Knuth during the development of TEX<sub>78</sub> and to additional four large industrial projects. Data from these projects were previously used to validate the original software test process model [1, 2, 3, 4].

### 6.1 Transformer Project Description

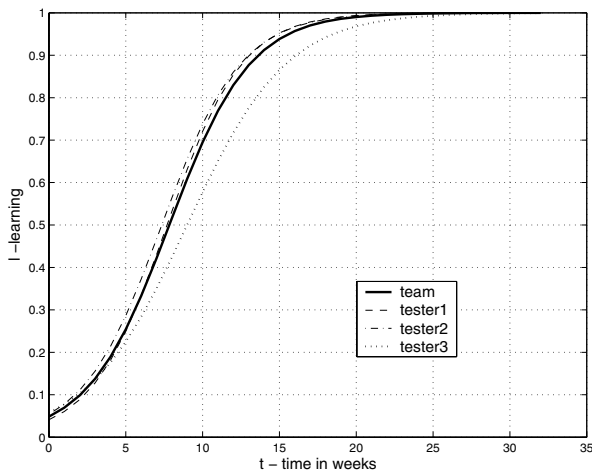
A company was given an application, denoted here *SCOBOL*, with 4 million lines of COBOL code to be transformed into a functionally equivalent application in SAP/R3, hereafter referred to as *S<sub>SAPR/3</sub>*. A tool, hereafter referred to as a *Transformer*, was developed to automate this transformation. The company maintained the detailed account of the history of detected errors, from its detection to resolution. The information presented here was either directly obtained from the project manager or by interviewing the project manager. The learning parameters were assumed based on the skills and knowledge of the team members.

### 6.2 Choice of parameters for Transformer project

Various parameters required for the model validation related to the current project were established based on following assumptions and also borrowed from the previous work [3, 1, 2].

**Initial Knowledge:** The initial knowledge (L0) was determined based on the knowledge of the testers/developers

about the current project. Its assumed to be very small since this was a new project and moreover it was unique *Transformer* project which involved experts of two different languages (COBOL/SAP) separated by long span of time in software development industries. At the beginning of the software project, the project was divided into multiple constituents and appropriate weight were assigned to these constituents as shown in the Table 1 and explained in Section 5. These constituents were selected and weighted for their dominance in the software test process (STP) for the *Transformer* project. These constituents provided a guideline to the manager at the planning stage to select and form a team. During the selection of the team members, each individual team member was rated against the chosen parameters on a scale of 0-1 as shown in the Table 1. The initial knowledge of the individual team members was calculated as the convex combination of each individual's skills in the each constituents of the project. Each tester was then rated for their expected contribution to the project relative to other team members. The initial knowledge of the team was further determined using equation 15 as appears in the Table 1.



**Figure 6. Individual and Team Learning estimated behavior for the *Transformer* project.**

Individual Learning rate were assessed by the manager as shown in the Table 1 based on the previous performance and learning ability shown by the testers. It needs to be emphasized that such parameters are chosen by the manager and these empirical values can always be improved based on new data as the project advances. The learning rate of the team is shown in the Table 1 and were assessed using their individual learning rate and their relative impact on the process as determined while establishing the initial knowledge of the team. The Learning pattern of the entire team is shown in Figure 6 based on the assessed date as presented

in the Table 1. The faster learning rate is due to the experience, knowledge and skill set of the testers. However, since this was a new project the initial knowledge were assumed lower. Data from initial six weeks were used to determine the initial estimates of the number of errors in the project. This estimate was further normalized due to proprietary reasons. Various other important model parameters were estimated and presented in the Table 2. A detail description of methodologies used for the estimation of these parameters reflecting the original STP model has been discussed in various places [3, 1, 2].

	weeks	$\gamma$	$\xi$	$\zeta$	$F_d$
period 1	1 to 6	0.44	19.89	22.78	59%
Period 2	6 to 10	0.56	31.46	69.86	61%
Period 3	10 to 14	0.75	35.07	156.45	33%
Period 4	14 to ..	0.75	49.09	156.45	25%

**Table 2. Original Model Parameters with a constant complexity  $s_c = 48$  and a constant size for the test team  $w_f = 3$  for the *Transformer* project.**

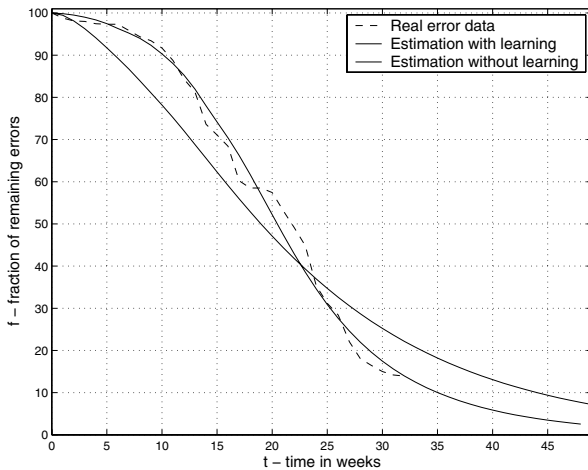
$F_d$  is the disturbance parameter reflecting the temporary discontinuity or delay in the software test process due to reasons like communication, adaptation, software build delay, installation failure, software/hardware failure or any unpredictable accidents. All such delays except adaptation delays are accounted by calibration algorithms. On the other hand delay for Adaptation has been quantitatively accounted for using learning relationship. Hence for the purpose of the analysis of Learning Effect,  $F_d$  has been reduced to zero.

### 6.3 Validation Results for Transformer Project

Figure 7 shows the results obtained by evaluating the effect of learning on the software test process for the *Transformer* project. A relatively better and close account of the actual data has been reflected in the modified STP Model. The qualitative improvement in the model has been further corroborated quantitatively using the Euclidean distance<sup>8</sup>. The Mean Euclidean distance from the real data for the *Transformer* project of the models with and without learning are 2.12847 and 7.41347 respectively. The accuracy of the model has increased 71.28% by accounting for the learning behavior during the STP based on our proposed S-shaped learning pattern. Further validation of improved STP model was conducted with software test process data collected from a large software industry<sup>9</sup>. Figure 8 shows

<sup>8</sup>The Euclidean distance between two points  $a(x_1, y_1)$  and  $b(x_2, y_2)$  is given by the equation  $d(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

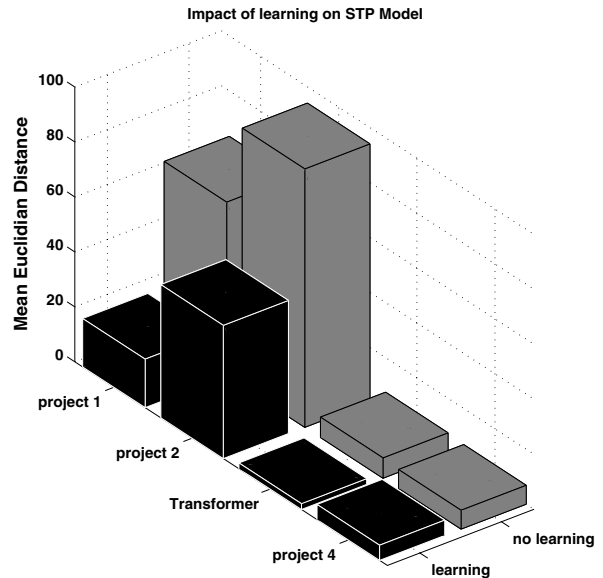
<sup>9</sup>company identity is not disclosed for proprietary reasons



**Figure 7. Results from the application of the two versions of the feedback state model to the *Transformer* project. The model accounting for learning shows a large improvement over the previous version.**

the mean euclidean distance between the actual error data and error data obtained using STP model with and without learning. Small values of mean euclidean distance show a better model of the process and large values of mean euclidean distance implies a poor modeling of the actual process. In all 4 projects under consideration, mean euclidean distance is lower in case of STP model process with learning. Projects 1, 2, and 3 shows approximately 50-70% improvement with learning than without learning. Such large improvement can be explained by the large learning phase at the beginning of the project which is not being accounted with the earlier model. A large learning phase in software projects are caused by very low or no initial knowledge accompanied with lower rate of learning. On the other hand, Project 4 shows approximately 20% improvement in the STP model with learning over STP model without learning. Smaller values indicates shorter learning phase in the software testing process. Initial learning phases are shortened due to very good initial knowledge and relatively better rate of learning. Such favorable conditions is realized if software testing team continue to work on the same projects for multiple releases.

From the results of the case studies presented above, we can conclude that the improved state model of the STP accounting for learning presents a major increase in the accuracy of the predictions. As expected, the larger the learning phase, the better the predictions of improved model when compared to the previous model.



**Figure 8. Average euclidean distance between the application of the State Model with and without learning. Projects 1, 2, and 3 (*Transformer*) show improvements around 50-70% and Project 4 shows a 20% improvement when using the model accounting for learning when compared to the previous version of the model.**

## 7 Conclusions

Formal Software Test Process(STP) modeling in the absence of quantitative approaches to disturbances like learning effect and communication overhead presents a clear weakness in the accuracy of predicting the behavior of the process [3, 4]. Current work has dealt with the Learning effect by proposing a novel “Learning Model for software test processes” extensible to any software development phase. Using the proposed Learning model, disturbances arising due to learning in the STP model have been explicitly accounted for during the Software Testing Process.

The improvement in the original model from inclusion of learning parameter can be observed from the lower values of the mean euclidean distance presented in Section 6.3. Actual amount of improvement is dependent on the nature of project and testing team involved. However, if software project involves learning, then the new improved STP model does provide better results. If the project doesn’t involve much learning then also new STP model will provide comparable modeling process.

In addition to the inclusion of learning in to the state model, communication overhead is also being investigated.

The insertion of elements that dominates the behavior of the STP to the state model increases its accuracy but it also increases the complexity of the model. Though many other aspects could be included into the model, to keep the balance between accuracy and complexity only these two aspects are being considered.

## References

- [1] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "A state variable model for the software test process," in *13th International Conference on Software and System Engineering and their Applications (ICSSEA'2000)*, vol. 2, (CNAM, Paris, France), pp. 1–10, CMSL CNAM, December 2000.
- [2] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "A state model for the software test process with automated parameter identification," in *In Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference (SMC2001)*, (Tucson, Arizona), pp. 706–711, 10 2001.
- [3] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "A formal model of the software test process," *IEEE Transactions on Software Engineering*, vol. 28, pp. 782–796, 8 2002.
- [4] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "Using sensitivity analysis to validate a state variable model of the software test process," *IEEE Transactions on Software Engineering*, vol. 5, pp. 1–39, 5 2003.
- [5] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterwill, S. M. Sutton, and A. Wise, "Little-jiljulitte: A process definition language and interpreter," in *Proceedings of the 22nd International Conference on Software Engineering*, (Limrerick, Ireland), pp. 754–757, 6 2000.
- [6] T. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*. Upper Saddle River, New Jersey: Prentice Hall Software Series, 1991.
- [7] R. Madachy, "Modeling software processes with system dynamics: Current developments," in *Proceedings of the 1996 System Dynamics Conference*, (Cambridge, Massachusetts), System Dynamics Society, July 1996.
- [8] R. A. DeCarlo, *Linear Systems: A state variable approach with numerical implementation*. New York: Upper Saddle River, Prentice-Hall, 1989.
- [9] D. G. Luenberger, *Introduction to Dynamic Systems*. John Wiley & Sons, 1979.
- [10] L. Ljung, *System Identification: Theory for the user*. Englewood Cliffs, New Jersey: Prentice-Hall, 1987.
- [11] J. W. Cangussu, "Convergence assessment of the calibration algorithm for the state variable model of the software test process," in *In Proceedings of the IASTED International Conference on Software Engineering (SE'03)*, (Innsburk, Austria), pp. 10–13, 2 2003.
- [12] D. Towill and J. Cherrington, "Learning curve models for predicting the performance of amt," *The International Journal of Advanced Manufacturing Technology*, vol. 9, pp. 195–203, 1994.
- [13] D. Towill, "Forecasting learning curves," *International Journal of Forecasting*, vol. 6, no. 1, pp. 25–38, 1990.
- [14] L. E. Yelle, "The learning curve: Historical review and comprehensive survey," *Decision Sciences*, vol. 10, pp. 302–328, 1979.
- [15] J. G. Carlson, "How management can use the improvement phenomenon," *California Management Review*, vol. 3, no. 2, pp. 83–94, 1961.
- [16] J. Kajihara, G. Aamiya, and T. Saya, "Learning from bugs," *IEEE Software*, vol. 10, pp. 46–54, 7 1993.
- [17] A. B. Badir, "Computational survey of univariate and multivariate learning curve model," *IEEE Transactions of Engineering Management*, vol. 39, pp. 176–188, 5 1992.
- [18] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, And Controlling*. John Wiley & Sons, 7th ed., 2000.
- [19] L.B.S.Raccoon, "A learning curve primer for software engineers," *Software Engineering Notes*, vol. 21, pp. 77–86, 1 1996.
- [20] N. Hanakawa, S. Morisaki, and K. ichi Matsumoto, "A learning curve based simulation model for software development," in *Proceedings of the 20th International Conference on software engineering*, (Kyoto, Japan), pp. 350–359, IEEE Computer Society, 1998.