

# Software Release Control using Defect Based Quality Estimation

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

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

Richard M. Karcich  
Sun Microsystems  
Network Storage  
Broomfield, CO 80021  
Richard.Karcich@sun.com

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

## Abstract

*We describe two case studies to investigate the application of a state variable model to control the system test phase of software products. The model consists of two components: a feedback control portion and a model parameter estimation portion. The focus in this study is on the assessment of the goodness of the estimates and predictions of the model parameters and their utility in the management of the system test phase. Two large network management applications developed and tested at Sun Microsystems served as the subjects in these studies. Unlike the release of products based on marketing or deadline pressure, estimates of the number of residual defects are used to control the quality of the product being released. The estimates of the number of defects in the application when the test phase began and at the current checkpoint are obtained. In addition a prediction is made regarding the reduction in the number of remaining defects over the remaining period. The estimates and predictions assist the management in planning the test phase and allow inferring the level of customer support needed subsequent to product release. The results of both case studies are satisfactory and, when viewed in light of other studies conducted at Sun Microsystems, show the applicability of the state variable model to the management of the software test process.*

## 1 Introduction

Estimation of residual defects in a software product is a difficult though important activity during the software development process. An accurate and timely estimate can be of enormous use to the management in deciding when to release the product. Though software product releases are often based on market issues and deadline expiration, availability of reliable estimates of residual defects will likely help improve customer satisfaction and reduce maintenance costs.

The objective of the two case studies presented here was twofold. First, we wanted to assess the accuracy of the proposed technique for the estimation and prediction of residual defects in software applications. Second, we wanted to investigate how useful such estimates and predictions are in fixing and revising checkpoints during the system test phase. In this study by “estimation” we refer to (a) estimates of the initial number of defects, i.e. when the system test phase started and (b) the number of defects at the current state in the test phase. Estimate (a) is useful in answering questions such as “What percentage of defect reduction have we achieved so far?” Estimates (a) and (b) are useful in making decisions on whether or not the test team should exit the current sub-phase of the system test phase, e.g. the  $\alpha$ -phase, and enter the next, e.g. the  $\beta$ -phase. In addition to obtaining (a) and (b), we also predict the residual defects over the remaining duration of the system test phase. This prediction assists management in resource allocation and a possible revision of the testing and maintenance budget. Our approach to estimation and prediction is incremen-

---

\*Financial support provided in part by SERC and NSF.

tal in that we recalibrate the model parameters as additional defect data becomes available and hence revise (a) and (b) and the predictions. As the recalibration is done automatically via a tool, it can be done as often as necessary.

Software products that are released late and/or with a lower quality than expected are common place in the software industry. One of the main reasons associated with this problem is the lack of techniques to predict and control the development process. Initial predictions and changes during the process are done, in most cases, exclusively based on the experience of managers. Existing techniques such as COCOMO [1] and System Dynamics [2] help in the initial predictions, but they lack a closed feedback loop solution to improve the overall controllability of the process. The need for a quantitative and deterministic solution is supported, for example, by Lehman’s statement: “Mastery of feedback mechanisms demands adequate quantitative models individually calibrated against real world software evolution environments.” [3]

The use of a state variable model to control and predict software project outcomes has proven to be very promising. Unlike simulation models, the state variable model not only goes beyond answer “what if” questions but applies feedback and control mechanism [4]. While formal process languages such as Little-Jil [5] is limited to procedural (coordination in the process steps) aspects of the processes, the model used here, at the very least does provide quantitative estimation of the process steps. It can be used as a complementary model to process languages 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 the state model. Feedback solution in the present model is closed loop where output of the process is used to manipulate the input for any desired effects such as stabilize the system or decrease sensitivity to parameter variations. Closed feedback solution provides the opportunity for project managers to target/achieve the modified quality objective of the project by making controlled changes in the model parameter unlike system dynamic models where improvement can only be achieved by uncontrolled iteration.

The products used in the two case studies are large network storage management applications developed at Sun Microsystems and were developed as part of two projects hereafter referred to as Project  $\Omega$  and Project  $\Theta$ . The “large network storage management applications” provide detailed information on the entire storage infrastructure (i.e., hosts, switches, storage devices) including network configuration, operational status, performance monitoring and trending, utilization, assignment and resource availability. A user’s view is automatically adjusted to conform to their role in the enterprise. The information provided is valuable for intelligent budgeting, capacity plan-

ning, analysis of short-term and long-term requirements, etc.

The estimates of residual defects from the calibration (estimation and prediction) algorithm are presented to the management at different checkpoints during the test process. The time to reach a desired quality, as measured in the number of remaining defects, is directly computed from the estimates. Though, in these two case studies, the results were not used to determine the release time for the products, their accuracy shows the applicability of the approach for the estimation of an appropriate release time. In addition, estimation of the remaining number of defects and the effort associated with their removal allows for the estimation of customer support effort. The outcomes of the studies are a strong evidence of the applicability of the state variable model to control the release of software products based on a desired quality objective.

The remainder of this paper is organized as follow. A brief description of the state variable model of the software test process (STP) [6, 7] is presented in Section 2. Section 3 presents a brief description of the test process in the Network Storage unit at Sun Microsystems. The description, results, and analysis of the case studies of two network management projects are the subject of Section 4. The computation of the parameters used for the case studies is also addressed in Section 4. A discussion about estimated and actual number of defects is presented in Section 5. Section 6 presents and compares related work while the general conclusions drawn from the study are in Section 7.

## 2 The State Model for the STP

A linear deterministic model of the system test phase of the STP is based upon three assumptions. The assumptions are based on an analogy of the STP with the physical process of a spring-mass-dashpot system and also in Volterra’s predator-prey model [8]. A description and justification of this analogy and the choice of a linear model is given elsewhere [6]. The model has been validated using sets of data from testing projects and also by means of an extremal case and a sensitivity analysis [7]. The assumptions and the corresponding equations follow.

**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 num-*

ber 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  $\zeta(s_c) = \frac{\xi}{s_c^b}$  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 ( $\dot{x} = Ax + Bu$ ) [9, 8] 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 [6]. Using data available and an estimation of  $\dot{r}$  [6] 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  $M = e^{A T}$ ,  $T$  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 [9] 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

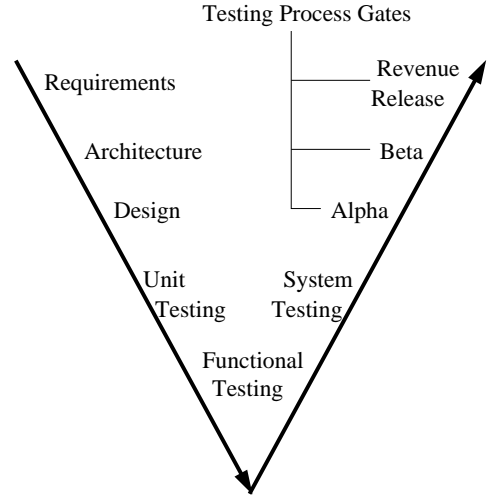
$$\begin{aligned} \pi_A(\lambda) &= \det[\lambda I - A] \\ &= \lambda^2 + \frac{\xi}{\gamma s_c} \lambda + \frac{\zeta w_f}{s_c^{(1+b)}} \end{aligned} \quad (6)$$

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 [6].

The STP state model has been applied to data from two projects with encouraging results [6, 12]. The subjects used in the two studies reported here are indeed much larger than the ones used in previously. Further, the estimates for Project  $\Omega$  and Project  $\Theta$  were generated and revised for the management several times during the study as additional defect data became available.

### 3 Software Test Process at Sun Microsystems



**Figure 1. Overall structure of the development process at Sun Microsystems with the three management testing gates: alpha, beta, and revenue release during the system test phase.**

The following description of the Sun Microsystems Network Storage test process is based on a series of gates at which management is presented with updates. The first gate is an Alpha gate, the second is Beta and the third is Revenue Release. The quality of the product is the major factor driving the determination of the last two gates. For example, the entrance to Beta Gate may be constrained to achieving 70% of the specified desired quality while Revenue Release may require attaining at least 95% of the desired quality.

From the point of view of the test process, entry to Alpha typically means the software has all its intended func-

tionality. Once in the Alpha Gate and beyond this point, no more significant changes are expected in the software product. That is, though it may happen, the changes are not expected to affect the design or the functionality of the product. Testing at the Alpha Gate is done by in-house testers and customers are not involved in the process.

Entry to the Beta gate typically involves release of the software to a number of internal and external customer sites for further testing. At this point the product should have achieved a reasonable quality. It should be clear that in-house testers continue to be involved in the process.

Often, the Revenue Release gate tends to be fixed and the project schedules and efforts must fit into that pre-set schedule. The expectation is that the quality of the product should be very close to the pre-specified values. After releasing the product, any defect that needs to be fixed is considered as maintenance and enters a specific maintenance cycle for removal.

There are three major groups of software engineers involved in the process of meeting the Revenue Release deadline:

- development engineers,
- functional test engineers, and
- system test engineers.

Prior to the Alpha gate, the development engineers are concerned with the design, coding, unit testing and integration testing of the product. Functional testing is basically meant to prove functionality (i.e., whether the product operates according to the specification). System testing does some performance and interoperability testing, generally in accord with operational profiles.

There are test plans for every phase of testing of the software. These plans are reviewed in order to remove any testing “holes” and a cross-functional test plan is prepared. The cross-functional test plan is primarily a high-level document for management.

The management of projects centers around the preparation/execution of product contracts that describe what the product is as well as the “how’s” and “when’s” of its development.

The data analyzed to prepare our models of the test process come from a defect-tracking database. Additional data is generally collected via conversations with particular individuals.

## 4 Storage Management Case Studies

Managing external storage arrays in current enterprise IT environments (corporate data centers) requires detailed knowledge of the complex configuration options a device

may provide. Such complex configuration options are influenced by the variety/heterogeneity of devices from different vendors typically present in a storage network. The complexity of configuring a management application is influenced by security requirements, reliability & availability requirements.

Storage or system administrators are responsible for keeping the storage network running, i.e., monitoring the health of the network (to include all the devices & applications in the network) to minimize downtime etc. Providing tools which reduce the administrative burden of external storage provides a more attractive return-on-investment for such devices. Many devices come with native administration interfaces which expose all the capabilities of a device. Often times these interfaces are targeted at expert level users and require in-depth knowledge of the device and its configuration options. Even when vendors provide a more intuitive interface for device management there is little consistency across such tools in both terminology and user experience. This inconsistency leads to unnecessarily large incremental cost of managing a new storage device, particularly in heterogeneous environments. The two projects described in Sections 4.2 and 4.3 provide both reduced complexity of device management and consistency across devices in an effort to address these issues.

### 4.1 Error Compensation of the Predictions

A description of an extra step that was used in the predictions of the initial number of defects is presented here before start analyzing the two case studies. This extra step is the compensation of the error from one prediction to the subsequent prediction. Let  $R_0(t_i)$  be the prediction of the initial number of defects computed using data from the period  $t = 0, \dots, t_i$ . Assume two consecutive predictions  $R_0(t_i)$  and  $R_0(t_{i+1})$  are available and  $t_i = x$ ,  $t_{i+1} = y$ ,  $y > x$ . An error  $e_i$  for estimation  $R_0(t_i)$  can be computed as follow.

$$e_i = 1 - \frac{R_0(t_i)}{R_0(t_{i+1})} \quad (7)$$

The error  $e_i$  at estimation  $R_0(t_i)$  can now be used to compensate the subsequent estimation  $R_0(t_{i+1})$  as in Eq. 8.

$$R_0(t_{i+1}) = R_0(t_{i+1}) \times (1 + e_i) \quad (8)$$

The time difference ( $t_{i+1} - t_i$ ) between two estimations may influence the results of the predictions. A small time difference may result in a sequence of small changes in the estimation. While this would be desired for the control/analysis of a physical system, a software process cannot be changed on a daily basis and more time is needed to evaluate and perform corrective actions (schedule or work

force changes, for example) under the light of the predictions. On the other hand, a large time difference may result in a potential problem being discovered too late. An optimal time difference appears to be project dependent but its analysis is beyond the scope of this paper. For both projects described In Sections 4.2 and 4.3, the initial estimation was done when data was first made available to us. The subsequent estimations were also done based on data availability until convergence to a common estimation or a small error is perceived.

Though many distinct and more elaborate compensation schemes are available, the simple computation presented above has shown to be adequate for the case studies conducted here. The application and comparison of additional error compensation procedures such as linear regression and the least-squares criterion [10] are deferred to future work.

## 4.2 Project $\Omega$

The software product of this project provides device management for external storage arrays in enterprise environments (SAN-Storage Area Networks and DAS-Direct Attached Storage). Project  $\Omega$  provides configuration, control and asset information viewing for early-generation Sun storage devices with follow-on storage-management releases addressing more advanced SUN storage products and third-party storage arrays.

### 4.2.1 Parameters

As can be noticed from Eqn. 4, the state variable model of the STP makes use of six parameters in the prediction and control of the process. Regarding software complexity, any metric or set of metrics can be used in the model. When more than one metric is used, they are combined using a convex combination approach [13]. Assume two metrics,  $M_1 = KLOC$  and  $M_2 = Cyclomatic\ Complexity$ , are available for a project. The question “Does the choice of the metric used affect the outcomes of the model?” arises. The parameter calibration algorithm normalizes the use of different metrics and minimizes its effect. Therefore, as long as the same metric or combination of metrics is used during the project, the choice of metrics does not impose a problem. If a different metric becomes available and the manager decides to use it in the middle of a project, recalibration must be done at all previous steps to ensure compatibility. One question that remains is weather a 10% change in  $M_1$  is equivalente to a 10% change in  $M_2$ . Though this equivalence is likely not true it does not affect the CDM Model as it is applied to the system test phase where almost all the functionality has already been included and changes on complexity are expected to be minimal.

With respect to Project  $\Omega$ , KLOC (kilo lines of code) was used as a complexity metric. On a scale using values

of  $s_c$  as very small, small, medium, large, and very large; Project  $\Omega$  can be considered a medium project based on its number of KLOC.<sup>1</sup> Though the use of additional metrics in combination with KLOC would be desired, due to availability issues a single size-oriented metric to capture the complexity of the product is used here. In any case, the CDM Model should be able to capture the dominant behavior of the process. A discussion of a more appropriate metric for network storage management products is beyond the scope of this paper. Also, since there were not significant changes in the size of the code, the value of  $s_c$  is assumed constant over the period under consideration.

The second parameter used in the model is the work force  $w_f$  representing the number of elements in the test team. The test team for Project  $\Omega$  started with 5 members and was increased by two at the end of the eighth week. The size of the test team remained at 7 until the end of the test process.

**Table 1. Weights and values of quality factors used in the computation of parameter  $\gamma$  for two distinct periods of the test process process for Project  $\Omega$ .**

Period I - week 1 to 8			
	Weight $w$	Value $v$	$w \times v$
Test team experience	0.5	0.7	0.35
Test plan adequacy	0.3	0.7	0.21
Automation level	0.2	0.2	0.04
$\gamma =$			0.6
Period II - week 9 to 21			
	Weight $w$	Value $v$	$w \times v$
Test team experience	0.5	0.8	0.4
Test plan adequacy	0.3	0.7	0.24
Automation level	0.2	0.2	0.04
$\gamma =$			0.65

The quality  $\gamma$  of the test process is the third parameter in the model and may assume any value ranging from 0 (extremely poor test process) to 1 (almost perfect test process) [6]. The initial value of  $\gamma$  is calculated to be 0.6 at the beginning and was increased by about 10% at the end of the eighth week as a consequence of the inclusion of two experienced testers in the test team. The computation of  $\gamma$  is done according to the manager’s perception of the most important factors in the process. Once identified, these factors are weighted (constrained to their sum being equal to 1) and a qualitative assessment determines a value ranging from 0 to 1 for each factor. A weighted average sum of the product of factors and weights is used to compute a value for  $\gamma$  [14].

<sup>1</sup>The nominal value of KLOC for Project  $\Omega$  cannot be disclosed here due to proprietary reasons.

For Project  $\Omega$  three factors were used to compute the quality of the process: the experience of the test team, the adequacy of the test plan, and the level of automation of test process. A justification of the choice of these factors is beyond the scope of this paper but their values and weight are given in Table 1.

The parameter  $b$  in Eqn. 4 accounts for the type of the project as in COCOMO [1]. It assumes values ranging from 1.05 (easy low-risk projects) to 1.20 (large high-risk projects). Project  $\Omega$  is assumed here to have an intermediate level and a value of 1.12 has been used in the model.

Assuming the selected users as testers post Beta gate, the number of the testers increases when entering the Beta and the Revenue Release gate. However, this increase may also decrease the overall quality of the process due to the lack of coordination amongst members of the larger test team. Such changes in the process cannot be properly quantified and are accounted for by recalibrating the model and computing new values for the proportionality constants  $\xi$  and  $\zeta$ . The recalibration is done at whenever changes in the process are observed. For Project  $\Omega$ , end of week eight and the entrance to the Beta and Revenue Release gates represent three change points.

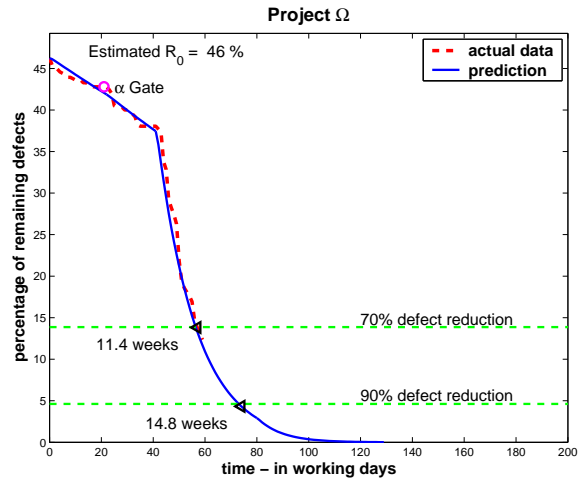
#### 4.2.2 Results

Before presenting the results of the case study for Project  $\Omega$ , a few clarifications are in order and are also valid for the Project  $\Theta$  case study reported in Section 4.3. For all the plots, the x-axis represents the number of working testing-days and the y-axis represents the percentage of remaining defects.<sup>2</sup> It should also be noticed that the value along the y-axis at time zero represents an estimate of the initial number of defects. The accuracy of estimates improves with the availability of additional data [11]. Here we assume that the estimation using all the data (including field defect data) represents the actual number of defects. Therefore, estimation of the initial number of defects during the early phases of the test process represents only a fraction of the total defects and in this case is not 100%. Finally, it appears from the plots that the testing profile matches exactly the user's profile. Though the company strives to test in accord with user profiles, existing field data for the projects presented here may not be large enough to draw conclusions regarding how closely the test profile may or may not match the users' profiles.

The results of the predictions for Project  $\Omega$  are presented in Figures 2, 3, and 4. The differences among them are the time of the prediction and consequently the amount of defect data used in the predictions, as well as the estimate

<sup>2</sup>The actual values are not provided due to proprietary reasons. Instead the data presented is normalized uniformly while retaining the relationship amongst individual elements.

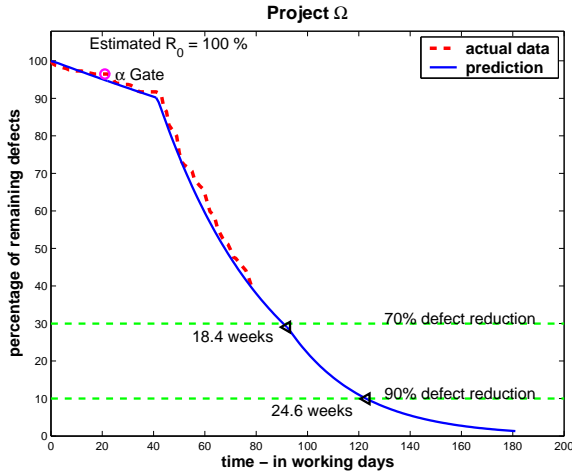
of  $R_0$ , the initial number of defects. A piecewise approximation is used here to accommodate unmodeled aspects of the CDM Model. One example is that the model used here does not account for learning which we believe is one of the major causes of the initial decay behavior of a test process. As can be noticed from the figures, two segments are used in the predictions, one for time  $t = 0, \dots, 45$  working days and one for  $t = 46$  working days and beyond.



**Figure 2. Results, using data from the first 60 days of testing, from the application of the CDM Model to the test process of Project  $\Omega$ .**

Independent of the amount of data used for the predictions, it can be observed that the calibration procedure makes the output of the model follow the actual behavior of the process very closely. Though estimates can be obtained at almost any point in time during the test process subject only to a minimum number of data points available for the prediction, the plots in Figures 2, 3, and 4 present, respectively, the predictions made at time  $t_1 = 60$ ,  $t_2 = 80$ , and  $t_3 = 100$ . The major difference between the three plots is in the estimates of the number of initial defects, referred hereafter as  $R_0$ . Figure 2 shows that the estimate using data from the first 60 days ( $t_1 = 60$ ) of testing estimates  $R_0$  which represents 46% of what would be found subsequently to be the approximate total number of defects ( $R_0(t_1) = 46\%$ ). In this case it was believed that a 70% defect reduction had been achieved at the middle of week 10 and a prediction that a 90% reduction would be achieved close to the end of week 14.

Using data from twenty additional days ( $t_2 = 80$ ), the estimate of  $R_0$  increases from 46% to 73% ( $R_0(t_1) = 73\%$ ). The use of Eq. 7 results in a 37% error ( $e_1 = 0.37$ ) between the estimations  $R_0(t_1)$  and  $R_0(t_2)$ . This value is now used in Eqn. 8 to compensate the second estimation



**Figure 3. Results, using data from the first 80 days of testing, from the application of the CDM Model to the test process of Project  $\Omega$ .**

( $R_0(t_2) = R_0(t_1) \times (1 + e_1)$ ) resulting in a 100% value, as can be observed in Figure 3. A 70% defect reduction is believed to be achieved at week 18.4 and a 90% reduction is expected to be achieved at week 24.6. When the predictions are made at time  $t_3 = 100$  (Figure 4), the estimate of  $R_0$  equals the value found in the previous estimation leading to a 0% error and no need for compensation. The accuracy in the predicted time to achieve the specified defect reduction also increases considerably.

### 4.2.3 Analysis

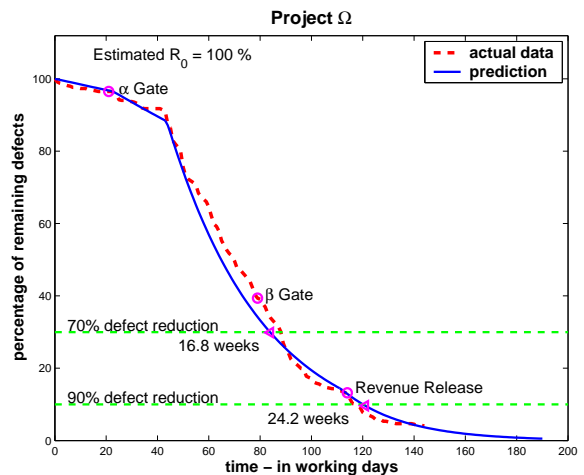
Recall that by the time the alpha gate is reached, it is desired to test a complete working product with no “drivers” and/or “stubs”. Entrance to the alpha gate for Project  $\Omega$  happened at day 22 of the test process as noticed from Figure 2. From this figure we observe two distinct periods, a “linear” decay in the number of remaining defects from  $t = 0, \dots, 45$  and an exponential decay for  $t > 45$ . We believe that two factors are responsible for the initial linear behavior. These are learning/adaptation and an incomplete product. The first factor accounts for installation problems, learning how to use/test the product, development of test cases, etc. The second factor represents that new functionality is still being added to the product. After the initial “flat” part of an S-shaped learning curve has passed and no more major functionality is being added to the product the curve changes from a linear to an exponential decay. The two cases studies presented here and several others conducted at Sun Microsystems provide a strong support for our belief. In this case we notice that Project  $\Omega$  has prematurely en-

tered the Alpha gate and that 23 additional days would have been necessary to fulfill the requirements to enter the gate.

A 70% defect reduction is the established requirement to enter the Beta gate. From the estimate of  $R_0$  and the predictions using data from the first 60 days, the entrance to the Beta gate should have been at week 11.4. However, as more data became available, the estimate of  $R_0$  improved and the entrance point changed to 18.4 and finally to 16.8 weeks as shown in Figures 3 and 4, respectively. Comparing the actual entrance point to the Beta gate at week 16 to the estimated 70% defect reduction at the end of week 16, we notice that the Beta gate has been entered prematurely.

A defect reduction of near 90% is specified as one of the requirements to release a product at Sun Microsystems. The predicted entrance point for this gate evolved from 14.8 to 24.6 to 24.2 days as additional data became available (see Figures 2, 3, and 4). We notice from Figure 4 that the actual Revenue Release entrance occurred at week 23 and it was indeed very close to the expected quality goal established at the beginning of the test process.

Overall, it appears that the accuracy in the estimation of  $R_0$  increased from 46% to 100% with the availability of additional data. As a significant improvement in the accuracy occurred prior to the end of the test process, it gave the test manager the capability to assess the quality of the product released and to determine the customer support subsequent to product release.



**Figure 4. Results, using data from the first 100 days of testing, from the application of the CDM Model to the test process of Project  $\Omega$ .**

Observations on defect data from the field reveal that, after releasing the product, the defect detection rate follows the same exponential decay as in the Beta testing. This data is plotted in Figure 4. After an initial period, the decay rate

slows down suggesting that not many new combinations of the features of the product are being exercised. This scattered data is not presented in the plot since it spans over a very large period of time. However, adding their value to the last data point is a strong indication that the prediction made at the end of the test process are extremely accurate.

### 4.3 Project $\Theta$

The application resulting from Project  $\Theta$  provides an integrated heterogeneous SAN management environment based on storage management standards that support assisted provisioning. Version 2.0 offers topological views which provide graphical representation of the logical and physical characteristics of a customer SAN. Health and diagnostic information is presented to the customer about the different views of their SAN. Configuration, extended diagnostics and asset reporting capabilities also exist using the topology view as a graphical entry to the functionality. Asset reporting, fault management, and set-based configuration are available from the topology views based on context sensitive selections. Fault detection, fault isolation and service advice are provided for all supported SAN elements. Storage administrators can do operations across sets of heterogeneous arrays, such as volume creation and storage pool creation. The operations include the ability to search for arrays from a group of many that can fulfill particular needs, such as the ability to offer a particular service level.

#### 4.3.1 Parameters

As in Project  $\Omega$ , KLOC is used as a complexity metric for Project  $\Theta$ . The nominal value of  $s_c$  is 50% larger for Project  $\Theta$  when compared to Project  $\Omega$ . Notice that the actual nominal values are used in the model to make the predictions. An initial test team of five members started testing Project  $\Theta$ . Two testers were added subsequently to the team at the end of the fourth week. Therefore,  $w_f = 5$  is used for  $t = 1, 2, \dots, 20$  and  $w_f = 7$  is used for  $t > 20$ .

The same factors used for the computation of the quality of the test process  $\gamma$  for Project  $\Omega$  were used for Project  $\Theta$ . Since both projects are related and the majority of the members of the test team are the same, a considerable increase in the experience of the team was perceived. An increase in the test plan adequacy also resulted from the previous experience with Project  $\Omega$ . Also, it was decided by the manager that the test plan adequacy has a larger impact on the project than the automation level which led to changes in their respective weights. Overall, an increase from  $\gamma = 0.65$  to  $\gamma = 0.8$  is detected from Project  $\Omega$  to Project  $\Theta$ . The values and weights used in the computation of  $\gamma$  are shown in Table 2. No changes in the quality

of the test process were perceived and the value of  $\gamma$  was considered constant over the entire period.

**Table 2. Quality factors and associated values and weights used in the computation of the quality of the test process  $\gamma$  for the test process of Project  $\Theta$ .**

	Weight $w$	Value $v$	$w \times v$
Test team experience	0.5	0.9	0.45
Test plan adequacy	0.4	0.8	0.32
Automation level	0.1	0.3	0.03
$\gamma =$			0.8

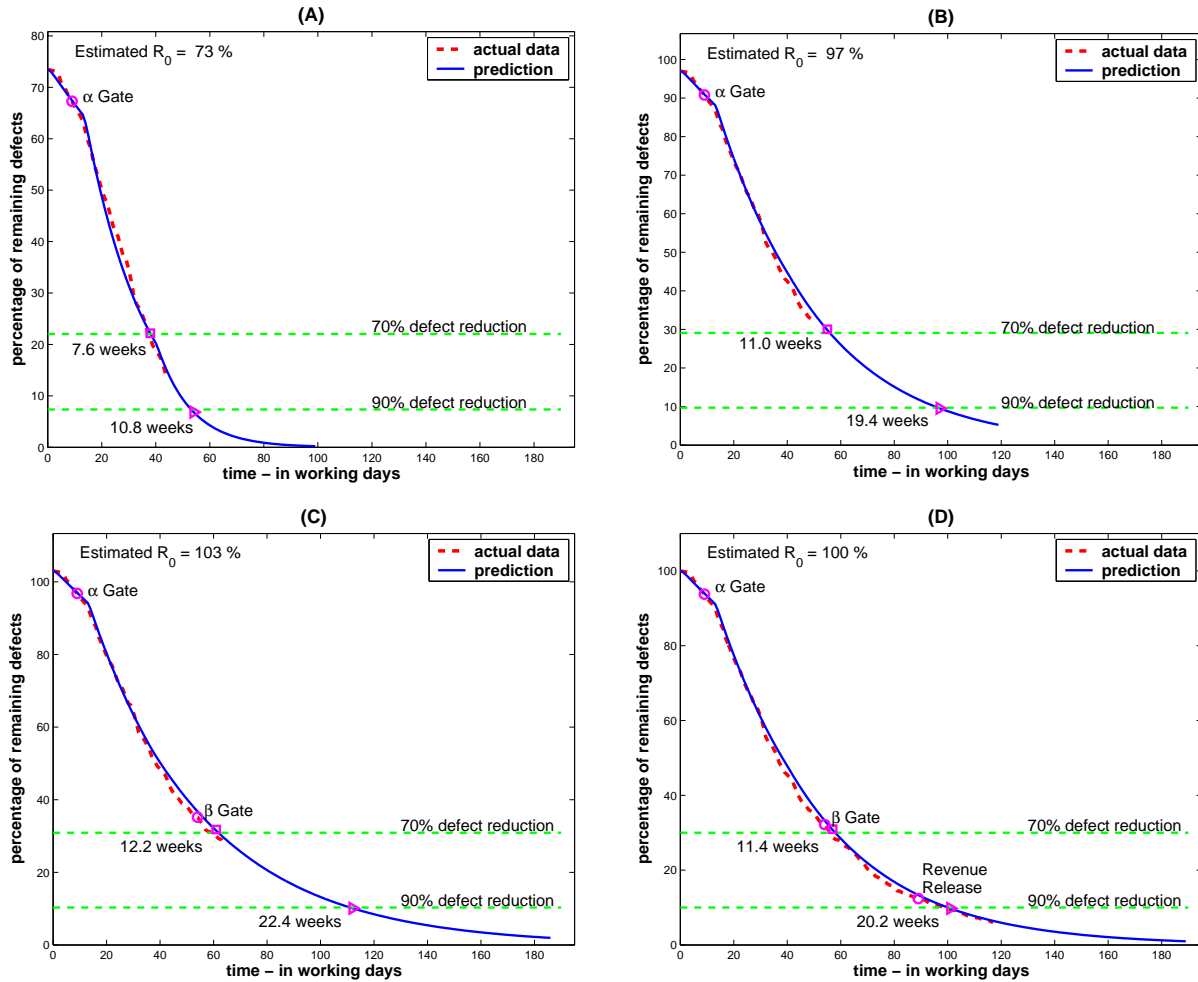
Given that Project  $\Omega$  and Project  $\Theta$  are related, the same value  $b = 1.12$  was used in both projects. As stated earlier, the proportionality constants  $\xi$  and  $\zeta$  are computed when (re)calibrating the model due to changes in the process environment. Recalibration also occurs when additional data becomes available. In addition to the recalibration to increase accuracy, the entrance to the Beta and the Revenue Release gates are also used as recalibration points. The addition of two more members to the test team at the end of the fourth week is not used for recalibration since it did not affect the perceived quality of the test process.

#### 4.3.2 Results

The results from the application of the CDM Model to Project  $\Theta$  are shown in Figure 5. Figure 5(A) shows the predictions using 45 days of defect data. As can be noticed, the alpha gate was entered at the end of week 2 and a 70% defect reduction was believed to be achieved at week 7.6 and a 90% defect reduction at 10.8 weeks. The estimated  $R_0$  at this point was 73%. The use of data from five additional days (Figure 5(B)) raised the accuracy of the prediction of  $R_0$  to 85%. Using the 14% error between the first two predictions to compensate the second one lead to the 97% value shown in Figure 5(B). In this case, the prediction of the Beta and Revenue Release entrances are 11 and 19.4 weeks, respectively.

The actual entrance to the Beta Gate happened at week 11 (55 days) as shown in Figure 5(C). At this point, the use of 60 data points with a 3% compensation error brought the predictions of  $R_0$  to 103%, the 70% defect reduction to 12.2 weeks, and the 90% defect reduction to 22.4 weeks. The 3% compensation error came from a 100% prediction when compared to the previous prediction of 97%.

Though actual data from 20 weeks appear in Figure 5(D), only data from 80 days are used for prediction purposes. When compared to Figure 5(C), Figure 5(D) shows a change of -0.8 weeks to achieve 70% defect reduction and a -2.2 weeks change to achieve a reduction of 90% in the



**Figure 5. Results from the application of the CDM Model to the test process of Project  $\Theta$ . 45 days of testing data are used in part (A), 50 days in part (B), 60 days in part (C), and 80 days in part (D).**

total number of defects. The actual release of the product occurred at week 18 as shown in Figure 5(D). The predictions without compensation using 60 and 80 data points produced exactly the same values. Due to this convergence in the prediction, a compensation error was not used in the last one.

### 4.3.3 Analysis

Though not as acute as in Project  $\Omega$ , a 10-day premature entrance to the Alpha gate was detected for Project  $\Theta$ . This improvement is, most likely, due to the experience acquired during the development of Project  $\Omega$ . With regard to the entrance to Beta gate, the predictions evolved from 7.6 to 11 to 12.2 and back to 11.4 weeks. The actual entrance was at week 11 which indicates an improvement in the accuracy of the predictions. Even though an accu-

rate prediction was made only very close to the actual Beta gate entrance, the results from the model helped managers assess whether or not the gate entrance requirements were met. Prior to the use of the CDM Model such assessment during the testing of the product was done mostly based on the manager's experience.

The predictions made for the release of the product with a 90% defect reduction started at 10.8 weeks and were updated later to 19.4, 22.4, and 20.2 weeks. The large offset in the first prediction is related to the offset in the estimates of  $R_0$ . As the accuracy in the estimates of  $R_0$  increased, so did the predictions to achieve the desired defect reduction. As can be seen in Figures 5(B), (C) and (D), the predictions were accurate when using 50 or more data points in the computation. This allowed for an early assessment of the quality of the product at the release time. The last estimate at  $t = 80$ , two weeks prior to the released, revealed that an

88% defect reduction would be achieved at the release of the product and that two more weeks would be necessary to achieve the desired reduction. As in Project  $\Omega$ , managers can infer the effort needed for customer support after the release of the product.

The improvement in the predictions for Project  $\Theta$  when compared to Project  $\Omega$  are largely due to the initial “linear” behavior of the respective test processes. These periods are with reference to the first 40 days for Project  $\Omega$  and first 20 days for Project  $\Theta$ . In addition to have a larger “linear” period, Project  $\Omega$  also presents a lower decay rate over this period than Project  $\Theta$ . The calibration algorithm is based on a least square approach that is more sensitive to the initial slower and longer decay period of Project  $\Omega$  as compared to the faster and shorter decay period of Project  $\Theta$ . Further improvement could be achieved for the predictions of Project  $\Theta$  if the error in the first prediction of Project  $\Omega$  was used to compensate the first prediction for Project  $\Theta$ . However, though the projects are related, many other factors can influence the error in the predictions and, in this study, we have decided not to use the information from previous projects.

## 5 Estimated versus Actual Number of Defects

In this work we refer to the actual total number of defects as the sum of all defects found during the test phases plus the field defects found over a large period of time. Two interpretations are possible in this case. The first, used here, is that this number is a good approximation to the total number of defects and that only a few defects are expected to be found in the future. The second interpretation is that the customers are using the product in a limited way and some of the functionalities of the product are rarely used or not combined in a complex manner. This would lead to a saturation effect [15] that limits the detection of additional defects giving the impression that are no more defects to be found. Though we lack data to confirm that the second option is *not* the case for both projects presented here, the developers and testers tend to believe in the first of the two interpretations.

## 6 Related Work

There exist several techniques for the estimation of number of defects [16] and they are often used in commercial development environments [17]. We note three major differences in the approach used here and what we find in the literature. First, we estimate defects in the context of a state variable model described in Section 2 of the software development process [6]. Thus, the model parameters, model

calibration, and the use of the estimates obtained, differs from the techniques used by Biffi [16] and by Onoma and Yamura [17]. Second, we use an incremental corrective approach to defect estimation and prediction. Third, we have integrated the estimation and prediction process and the use of its estimates to control the system test process.

Software reliability, and the myriad of models for its estimation [18], provides a good estimation of software quality based on observations of failure intensity. However, the use of software reliability to infer customer support may result in misleading conclusions. A product may have a high failure intensity with a low number of remaining defects or a low failure intensity with a high number of remaining defects. The former case could result in an over allocation of effort while the latter could have an opposite effect. A quality estimate base on the remaining number of defects is useful in deciding the release of a software product and to help with the allocation of resources to handle customer support.

## 7 Concluding Remarks

The accuracy of the estimates and the predictions of the number of remaining defects tends to improve our confidence in the algorithm used in the case study. Though estimates and prediction from the model were not used to actually fix the entry points for the three gates, the management does tend to believe that they are more accurate than the “guesswork” used otherwise. When the results of the two case studies presented here are combined with the Razorfish study [6], one gets a reliable and automated defect estimation model and a procedure for integrating it into the test planning and control process. It is our hope that the use of the calibration algorithm, and the associated state variable model, can of value to the management in the planning and execution of the system test phase of the test process for a large software system. We refer to “large” software systems because system test phases of such systems tend be long enough to provide a minimal amount of data needed for the application of the calibration algorithm. In addition, the accuracy of the estimates of residual defects at the time of Revenue gate leads us to believe that these could be used to delay the Revenue gate, if necessary, and to plan manpower in customer support and maintenance.

Several questions remain to be answered. First, note that the calibration algorithm is purely a “black-box” method. It does not use any code coverage data from the test process and from the field. This lack of coverage data is certainly a cause for concern as the estimates generated by the calibration algorithm might be as good as the data is. Thus additional defects may be found when customers use the application in ways significantly different from the testers. This suggests that the calibration algorithm ought to be modified

with the inclusion of code coverage metrics. Another question is “How does the size of a project, i.e. the length of the system test phase affect the accuracy of the estimates and hence its usefulness?”. From the studies presented here and those conducted in the past, it appears that a system test phase that runs approximately 24 weeks or more and uses about 5 or more testers can be handled well using our calibration algorithm. However, additional studies are needed to get a more reliable answer to this question. Last but not the least, the feedback control mechanism in the state model, though perceived to be a powerful tool for process control, has not been used in any of the studies reported here or in the past. While the calibration algorithm proposed seems to work well, the suggestions from the feedback control mechanism remain to be validated.

## References

- [1] B. Boehm, “Safe and simple software cost analysis,” *IEEE Software*, vol. 17, pp. 14–17, September/October 2000.
- [2] T. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*. Upper Saddle River, New Jersey: Prentice Hall Software Series, 1991.
- [3] M. M. Lehman, “Process modeling - where next?,” in *Proceedings of ICSE 19*, (Boston, Massachusetts), pp. 549–552, May 1997.
- [4] G. C. Goodwin, S. F. Graebe, and M. E. Salgado., *Control system design*. Upper Saddle River, New Jersey: Prentice Hall, 2001.
- [5] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton, and A. Wise, “Little-jil/juliette: A process definition language and interpreter,” in *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, (Limerick, Ireland), pp. 754–757, IEEE, June 2000.
- [6] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, “A formal model for the software test process,” *IEEE Transaction on Software Engineering*, vol. 28, pp. 782–796, August 2002.
- [7] 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. 29, pp. 430–443, May 2003.
- [8] D. G. Luenberger, *Introduction to Dynamic Systems: Theory, models and applications*. New York: John Wiley & Sons, 1979.
- [9] R. A. DeCarlo, *Linear systems : A state variable approach with numerical implementation*. Upper Saddle River, New Jersey: Prentice-Hall, 1989.
- [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 *Proceeding of the IASTED International Conference on Software Engineering (SE’03)*, (Innsbruck, Austria), February, 10-13 2003.
- [12] D. E. Knuth, “The errors of  $T_E X$ ,” *Software-Practice and Experience*, vol. 19, no. 7, pp. 607–685, 1989.
- [13] S. R. Lay, *Convex Sets and their Applications*. New York: John Wiley & Sons Inc., 1982.
- [14] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, “A state model for the software test process with automated parameter identification,” in *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference (SMC 2001)*, (Tucson, Arizona), pp. 706–711, October 2001.
- [15] J. R. Horgan and A. P. Mathur, *Handbook of Software Reliability Engineering*, M. R. Lyu, Editor, ch. “Software Testing and Reliability”, pp. 531–566. New York, NY: McGraw-Hill, 1996.
- [16] S. Biffl, “Evaluating defect estimation models with major defects,” *J. Syst. Softw.*, vol. 65, no. 1, pp. 13–29, 2003.
- [17] A. Onoma and T. Yamaura, “Practical steps toward quality development,” *IEEE Software*, vol. 12, pp. 68–77, September 1995.
- [18] J. Musa, *Software Reliability Engineering*. McGraw-Hill, 1999.