

# A control-theoretic approach to the management of the software system test phase

Scott D. Miller<sup>a,\*</sup>, Raymond A. DeCarlo<sup>b</sup>, Aditya P. Mathur<sup>a</sup>, João W. Cangussu<sup>c</sup>

<sup>a</sup> *Purdue University, Department of Computer Science, 250 N. University Street, West Lafayette, IN 47907-2066, United States*

<sup>b</sup> *Purdue University, School of Electrical and Computer Engineering, 465 Northwestern Avenue, West Lafayette, IN 47907-2035, United States*

<sup>c</sup> *University of Texas at Dallas, Department of Computer Science, Richardson, TX 75083-0688, United States*

Available online 22 May 2006

## Abstract

A quantitative, adaptive process control technique is described using an industrially validated model of the software system test phase (STP) as the concrete target to be controlled. The technique combines the use of parameter correction and Model Predictive Control to overcome the problems induced by modeling errors, parameter estimation errors, and limits on the resources available for productivity improvement.

We present an example of the technique applied to data from the execution of the STP of a commercial software development effort at a large software manufacturer. The example shows that the control technique successfully achieves the schedule and quality objectives despite uncertainty in the estimation of the model parameters.

© 2006 Elsevier Inc. All rights reserved.

*Keywords:* Feedback control; Model predictive control; Software cybernetics; Software process control; Software process modeling; System test phase

## 1. Introduction

We define the term software system test phase (STP) to refer to a set of activities intended to assess the *performance* and *interoperability* of the completed features of an application with respect to its requirements. By *software cybernetics* we refer to the set of techniques through which a software system may be adapted to meet changing goals in an evolving environment (Cai et al., 2003); in the present case the software system is a model intended to capture the progress of the STP. Lastly, we use *control* in the sense of the collection of quantitative techniques in the field of Control Engineering which provide methods for determining the best set of system adjustments which drive the behavior of the controlled system to within desired limits. The specific control method evaluated in this paper is derived from

the techniques in the area of Model Predictive Control (MPC) (Camacho and Bordons, 2004).

The subsequent sections detail a method for guiding the performance of the STP to a desired level in the presence of a changing environment, and indeed, fixed or varying quality and duration objectives.

The need for effective software process management (and control) is evidenced by nearly 50 years of literature on the topic and the high cost associated with large-scale software development. In the discipline of Control Engineering, there is also a large body of literature dedicated to techniques for eliciting a desired behavior from systems whose dominant behavior is captured mathematically. To date, little work has been done to draw the well developed control techniques from the engineering disciplines into the domain of software development. Software cybernetics, the name given to work directed toward that aim, is applied in this paper to a small piece of the management activities associated with software development—namely, the STP—such that this work may be considered a step toward

\* Corresponding author. Tel.: +1 765 496 7574; fax: +1 765 494 0739.

E-mail addresses: [millersd@cs.purdue.edu](mailto:millersd@cs.purdue.edu) (S.D. Miller), [decarlo@ecn.purdue.edu](mailto:decarlo@ecn.purdue.edu) (R.A. DeCarlo), [apm@cs.purdue.edu](mailto:apm@cs.purdue.edu) (A.P. Mathur), [cangussu@utdallas.edu](mailto:cangussu@utdallas.edu) (J.W. Cangussu).

importing the aims of software cybernetics into the field of software process control.

We are concerned with a model of the STP that aims to support the management tasks of (i) resource allocation; (ii) schedule/duration updates with respect to planning and budgeting of the STP; (iii) estimation of future support staff/hours and (iv) customer satisfaction with respect to the budgeting and planning of future maintenance and support activities for a project. The specific model to which we apply our technique is the CDM model first described in Cangussu et al. (2002) (where CDM refers to the authors of that model).

We define the management control objectives as (i) reduce, to the extent possible, the error in the estimate of the initial number of defects present in the project at the beginning of the STP; (ii) determine a set of quantitative changes to return a deviating STP to within the desired operational limits; (iii) incrementally update (i) and (ii) as additional process data becomes available for model re-calibration and error correction.

The concrete case upon which the goals and assumptions are validated is a testing scenario at large software manufacturer, detailed elsewhere (Cangussu et al., 2004), where the product under test is a Storage Area Network (SAN) management environment based on storage management standards. The product provides a graphical view of the physical, logical, and topological characteristics of the network, and allows the user to query for health, diagnostic, and configuration information. The environment provides reporting and fault management capabilities in a context sensitive manner via a graphical interface. Other features of the SAN management environment allow administrators to perform operations, e.g. volume and storage pool creation, across heterogeneous arrays under system management, and provide search capabilities based on various criteria, such as available service levels.

For this project, the STP was part of a testing regimen composed of unit and integration testing; functional testing (i.e. product matches the specification) and system testing. The management goals are set in terms of the target release gates defined as:

- (1)  $\alpha$ -gate: product has all intended functionality.
- (2)  $\beta$ -gate: more than 70% defect reduction (DR) from the initial number of defects.
- (3) Revenue Release: more than 90% DR; target date is narrowly fixed and must be met.

We note that the  $\beta$ - and Revenue Release gates are determined in terms of a software quality metric—percent reduction of initial defects—and will focus our attention on the task of reaching the Revenue Release on time.

### 1.1. Contributions

Following are the key contributions of the work reported here.

- (1) A Model Predictive Control (MPC) technique for the CDM model that provides sub-optimal process control adjustments to drive the performance of the STP within desired operational limits. The technique adapts robustly to changes in process parameters and changes in the process environment (Camacho and Bordons, 2004).
- (2) A new calibration (parameter identification) algorithm for the CDM model which is more resilient to variation in the training data than that reported in Cangussu et al. (2002).

We note that Contribution 1 is a significant improvement over the partial eigenvalue assignment technique in Cangussu et al. (2002) in that (i) a cost function is used to select from the (potentially large) set of control adjustments that attempt to minimize the cost function over the (finite) control horizon; (ii) The MPC algorithm updates its suggested process adjustments at successive time instants thereby overcoming unforeseen changes in the process and its environment and (iii) The new controller can incorporate more of the information which may be known a priori to the test manager, such as workforce availability, expected down-time, etc.

### 1.2. Organization

The remainder of this paper is organized as follows: Section 2 briefly ties our work to the existing body of literature. In Section 3 we summarize the existing CDM model of the STP that is the primary focus of our work. Some common problems in modeling, relevant to the current work, are addressed in Section 4. Section 5 defines the process management goals captured in the CDM model and Section 6 lays out the MPC-based control algorithm. Section 7 touches on some fortuitous features of our approach and how they may be exploited to implement some practical restrictions on the control suggestions generated by the algorithm. Section 8 suggests a method for selecting the parameters of the optimization cost functional. An example in Section 9 presents the results obtained by applying the proposed technique to the data obtained from the STP phase of the SAN project. Section 10 summarizes the data required and the overall process of applying the technique in industrial practice. Details of parameter calibration essential for the application of the CDM model, are discussed in Section 11. Our conclusions appear in Section 12.

## 2. Related work

The literature in Software Engineering spans nearly five decades in many sub-disciplines. Here, we isolate those disciplines most closely related to the current work and elucidate the similarities while highlighting the differences.

### 2.1. Software testing

While we are aware of the vast body of literature on Test Generation and Test Selection, such issues are out-of-scope for the CDM model. Improvements in these techniques will likely be reflected in the CDM model through the scalar *test process quality* parameter, leading most likely to an increased *defect removal rate*. Interestingly, the ideas of Software Cybernetics have been applied to issues in Software Testing as well (Cai, 2001, 2002; Cai et al., 2005).

### 2.2. Software process control

Literature in Software process control has focused on the task of monitoring and adjusting software processes so that they remain within acceptable limits of operation, or that they return quickly to within acceptable tolerance limits once a deviation outside such limits is detected. Among the approaches are *statistical process control* (e.g. Card, 1994; Jalote and Saxena, 2002; Dalal et al., 1993), defined by the philosophy that “...as long as you perform a process consistently it will demonstrate consistent results, in measures like productivity, error rate, and cycle time from project to project” (Card, 1994), discrete event and hybrid simulations and the familiar “what-if” mechanism for determining process change (Martin and Raffo, 2000a,b), and *process simulation* for the purpose of process comprehension (Andersson and Karlsson, 2001; Andersson et al., 2002; Kellner et al., 1999; Martin and Raffo, 2000b) where the idea is that a better process understanding will lead to better process improvement decisions.

We highlight that our approach uses neither the “what-if” trial-and-error-based approach for process change determination, nor are our assumptions anchored in the mandatory reuse of prior processes (except for the atypical case where the CDM model is initially trained from past-project data). We calibrate and re-calibrate our model to the data available from the ongoing project and provide rigorously derived quantitative control suggestions to the project management as an indication of those changes which they could implement to achieve the desired behavior specified to the controller; a “how-to” approach. In reference to this goal, our work is related to the majority of the software process control literature, though we believe it to be unique in its method derived from Model Predictive Control and novel in its approach to the correction of errors in the parameter estimates.

### 2.3. Software process modeling

Starting with software life cycle models, then progressing to software process models, and finally to software process simulation (Scacchi, 2002); thus is the chronology found in the software process modeling literature. Scacchi distinguishes two categories of process models: *prescriptive* and *descriptive* (Scacchi, 2002); prescriptive models are

used to determine, “what character should I build into my process?” (e.g. Cass et al., 2000), while descriptive models answer the question, “what is the character of the process that I have?” (e.g. Boehm et al., 2000; Cangussu et al., 2002). Let us focus on *descriptive* models as this is the class under which the CDM model falls.

Noting that the CDM model has a simulation—whose output represents productivity—we further limit the comparison to those descriptive models which predict progress, and have an associated simulation.

Abdel-Hamid and Madnick (1989, 1991) are credited with an early attempt at applying techniques from the field of system dynamics to the modeling of software processes. Their work treats the phases of software development as black boxes, and models their interaction. Our work looks explicitly into one particular phase, the STP, and brings its behavior into focus.

Another related work is that of Iida et al. (1996), in which a model is developed to study the effect of concurrent (in contrast to serial) development of multiple feature sets. This is a rate-based system dynamics model of an aspect of the software development process which uses an approach analogous to ours, though the model itself is targeted to a different domain (i.e. completion of coding vs. our focus on completion of software testing).

We note that much of the work in simulation-based software process control deserves mention under the category of process modeling as well. While this has been by no means a complete survey of the vast area of software process modeling and simulation, we note that the CDM model is a novel descriptive model of the STP which may be simulated to predict the defect removal behavior of the test environment to which it is calibrated. To our knowledge, it is unique in character.

### 2.4. Software cybernetics

Software cybernetics is an emerging field concerned with the fusion of techniques for adaptation and control with the issues that arise in all aspects of Software Engineering (Cai et al., 2003). Given this broad definition, the literature contains contributions in software cybernetics applied to related but different areas as varied as Adaptive Test-Set Selection (Cai, 2001, 2002; Cai et al., 2005) and Optimal Schedule Generation (Padberg, 2005).

To our knowledge, there has not been another attempt to apply techniques of adaptation and control to a predictive model for the purpose of guiding management decision making during the execution of the STP.

## 3. State model of the STP

Our work treats the problem of managing the STP as a feedback control problem. This approach requires two components: (i) a quantitative state model of the process to be controlled (in our case the STP) and (ii) a state-model-based control law which minimizes a cost function/

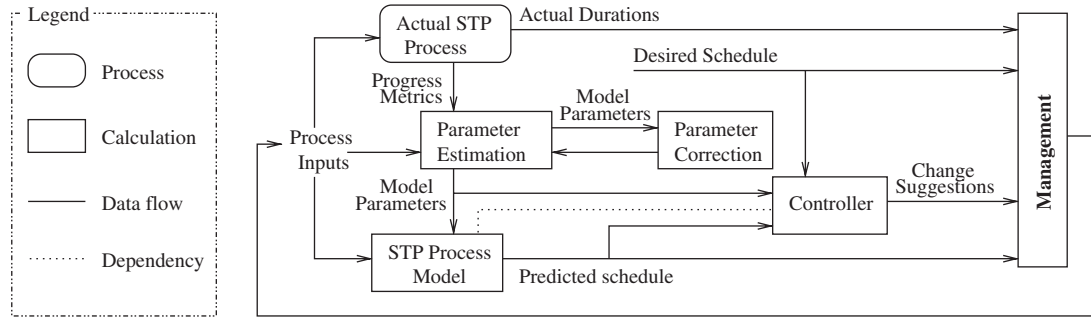


Fig. 1. Usage scenario for the STP model.

performance index tailored to management objectives for the STP.

In this section we review the formal state-based model of the STP, proposed in Cangussu et al. (2002), which is called *the CDM model*. Fig. 1 illustrates the usage of the CDM model in conjunction with an error correction algorithm which is detailed in Cangussu et al. (2004) and summarized in Section 4.2; it is this scenario upon which we build our algorithm based on model predictive control in Section 6.

The CDM model is composed of three assumptions built via analogy with physical systems. The first is based on Newton's second law:  $F = ma$ , where, in the context of the STP, force maps to the net effort ( $e_n$ ) applied by the workforce, mass corresponds to the complexity of the software ( $s_c$ ), and acceleration ( $\ddot{r}$ ) is understood to be the derivative of the rate (i.e. velocity) at which defects are removed

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

The second analogy is drawn with the Lotka-Volterra predator-prey model (Lotka, 1925; Luenberger, 1979; Volterra, 1926), and represents the effective effort ( $e_f$ ) applied by the workforce. Here, *effective* refers to that effort which has been productive in detecting and removing defects. At any given time, each member of the testing workforce can encounter only as many defects as are present in the software under test. Thus the workforce size ( $w_f$ ), multiplied by the remaining defects ( $r(t)$ ), serves to enumerate the potential encounters between testers and defects. The term  $\zeta/s_c^b$  is understood as the probability that the testing activities cause an encounter to actualize; this probability is inversely proportional to a power of the software complexity ( $s_c^b$ ), where  $b$  reflects *economies of scale* in the COCOMO (Boehm et al., 2000) family of models. The factor  $\zeta$  is calibrated to project data yielding the equation

$$e_f(t) = \frac{\zeta}{s_c^b} \times w_f \times r(t) \quad (2)$$

The third assumption posits a notion termed *error reduction resistance* which is analogous to friction. In Eq. (3) below, the error reduction resistance ( $e_r$ ) is proportional to the rate of defect removal ( $\dot{r}$ ) which implies that the faster one works, the more likely they are to make mistakes, suffer from fatigue, etc. Also,  $e_r$  is inversely proportional to

the *quality* factor ( $\gamma$ ) of the test process—both in terms of the workforce and the testing environment. The constant of proportionality ( $\xi$ ) is calibrated to project data

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

Writing the force balance equation (equilibrium equation), one observes that

$$e_n(t) = -e_f(t) + e_r(t) \quad (4)$$

Substituting Eqs. (1)–(3) into Eq. (4) yields the differential equation model

$$s_c \ddot{r} = -\frac{\zeta w_f}{s_c^b} r(t) - \frac{\xi}{\gamma} \dot{r}(t) \quad (5)$$

By defining the state variables to be  $r(t)$  (the number of remaining defects at time  $t$ ) and  $\dot{r}(t)$  (the defect removal rate at time  $t$ ) Eq. (5) yields the following non-linear state-based model for the STP:

$$\begin{bmatrix} \dot{r} \\ \ddot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{\zeta w_f}{s_c^b(1+b)} & -\frac{\xi}{\gamma s_c} \end{bmatrix} \begin{bmatrix} r \\ \dot{r} \end{bmatrix} \quad (6)$$

We note that the values of  $\zeta$  and  $\xi$  evolve over the course of the STP. At periodic *checkpoints*, these parameters are re-estimated; hence Eq. (6) can be thought of as a piecewise constant linear model provided that the inputs ( $w_f$  and  $\gamma$ ) and calibration factors ( $\xi$  and  $\zeta$ ) are held constant between checkpoints. Otherwise, the model becomes non-linear and time-varying.

#### 4. Problems of modeling

In this section we briefly address some of the common problems in modeling as the motivation to take a software cybernetic approach to management of the STP.

##### 4.1. Errors in the model

According to Box and Tiao (1992) (also commonly attributed to W. E. Deming) “All models are wrong, some models are useful”. The task of constructing a model is that of taking an infinite-dimensional real process, capturing its

dominant aspects, which seem most relevant, and rendering them into a formalism such as a state model or set of differential equations. This stripping away of the non-essential, uninteresting, or secondary aspects leads to the notion that there is an infinite-dimensional *actual model* which is equivalent to a finite-dimensional *analytical model* plus some *modeling error*. For the test process manager, this means that the output from the predictive models is inaccurate; management must then decide whether the degree of predictive error is sufficient to render the prediction useless. In terms of the CDM model, one may capture this relationship as  $STP = CDM + \epsilon$ ; where  $\epsilon$  denotes the difference between the incomputable “perfect” model (STP) and the constructed, approximate model (CDM). Here, it is obvious that  $\epsilon$  is neither known nor computable, however it may be bounded, and those bounds may be computable or known a priori.

For example, consider the evolution, during the STP, of the calibration parameters  $\zeta$  and  $\xi$  as mentioned in Section 3. Though the model periodically updates these parameters at the checkpoints, in the interim they are treated as constants. Stated more formally,  $\xi_i = \xi(t_i)$  is the value of the parameter at time  $t_i$  which denotes the time that the  $i$ th checkpoint is reached. The exact nature of the evolution that the calibration parameters undergo is unknown; they may be constant (i.e.  $\xi_i = \Xi$ , for some  $\Xi$ ) or time-varying (i.e.  $\exists(i,j)$  s.t.  $\xi_i \neq \xi_j$ ), however experience suggests that they are not constant. The assumption that a parameter is piece-wise constant is a common modeling simplification used to preserve analytical viability (McClure, 1975).

To mitigate the predictive error induced by model inaccuracy, the test process manager may opt to search for (more complex) models shown to have less significant modeling error, or they may select models which make use of one or more algorithms for correction (e.g. parameter correction). Section 6 outlines a software cybernetic approach to dealing with the modeling error in the CDM model by the construction of a model-predictive feedback control technique which incorporates model re-calibration and parameter correction.

#### 4.2. Errors in the parameters

Parameter estimation is confounded by the modeling error as well; typically the parameters are tuned using least-squares fit of the available data to the model equations. Thus a tuning algorithm will choose parameters which cause the (incorrect) analytical model (in our case, CDM) to provide the best fit to the available (correct) data produced by the actual process (STP, which is  $CDM + \epsilon$ ).

Implicitly, the parameter estimates compensate for the modeling error ( $\epsilon$ ). If the parameters denote physically meaningful quantities (e.g. defects remaining in the software), then the estimated values will be inaccurate and may even fall beyond reasonable physical limits leading to incorrect analysis and response (e.g. premature release of a software product from the STP).

In the CDM model, the training data is used to calibrate the parameters  $\zeta$  and  $\xi$ , as well as  $r_0$ —the initial number of defects in the code. It was noticed that early in the execution of the STP the relative lack of available data induced error in the estimation of  $r_0$ , which significantly impacted the performance of the CDM model (Cangussu et al., 2003). In Cangussu et al. (2004), the following linear error corrector is proposed for  $r_0$  in the CDM model.

Let  $p$  denote an arbitrary parameter in the CDM model. We then define

$$e(t_k) = \frac{p(t_k) - p(t_{k-1})}{p(t_k)} \quad (7)$$

to be the normalized difference between the most recent successive estimates of  $p$  (i.e.  $p(t_{k-1})$  and  $p(t_k)$ ), relative to the most recent estimate, where  $p(t_k)$  denotes the estimate made using all data up to time  $t_k$  (i.e. at the  $k$ th checkpoint). Implicitly,  $p(t_k)$  is assumed to be the more accurate than  $p(t_{k-1})$ , hence the normalization to  $p(t_k)$ .

A linear error corrector for  $p$  is then given by

$$p^{\text{new}}(t_k) = (1 + e(t_k))p^{\text{old}}(t_k) \quad (8)$$

where it is clear that Eq. (8) applies the relative difference between the most recent calculations of  $p$  again to the current calculation. In an environment where the available data for parameter tuning increases with time, this technique has elicited more rapid convergence of tuned parameters to their final values than successive model re-tuning alone (Cangussu et al., 2004).

## 5. Control objectives

The objectives we define for the system illustrated in Fig. 1 are twofold: (i) quantitatively estimate the progress of the STP for evaluation purposes and (ii) provide information to management in terms of process changes required for the STP to meet its schedule and quality objectives. Here, the schedule and quality objectives are assumed to be defined in terms of the proportion of defects remaining at various offsets in calendar time.

The CDM model identifies two parameters as being readily under the control of the project management:  $w_f$  and  $\gamma$ . These parameters are used to exert control over the STP process in system objective (ii) in order to meet the schedule and quality objectives as per Fig. 2.

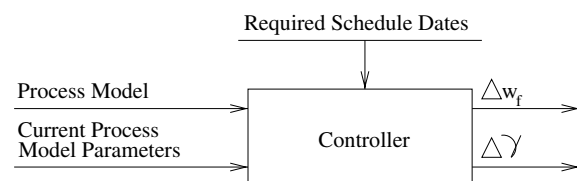


Fig. 2. Black box representation of the control problem.

We provide a MPC-based alternative to the partial eigenvalue assignment control technique in Cangussu et al. (2002) toward the following goals: (i) better tolerate the modeling error referred to in Section 4.1; (ii) incorporate the parameter corrections described in Section 4.2; (iii) allow for the specification of limits on the control variables to limit the control suggestions to that set which the manager is capable of implementing; (iv) allow more flexibility and finer temporal granularity as to when process changes are to be implemented and (v) provide a methodology that may be applied to other non-linear process models.

For the test manager, this new controller would then incorporate any specified limits on the resources available to the manager, and would provide suggestions which could be interpreted as “Make an addition of X team members for the remainder of the STP, and bring on Y temporary team members for 5 weeks. Increase the quality of the process by Z after the temps have left”.

## 6. A model predictive control approach

Model Predictive Control (MPC) is a methodology that, for a given process model, solves a finite horizon ( $N$  time unit) optimal control problem based on a desired performance to be elicited from the process over the finite horizon. The resulting control, however, is implemented only over the first  $j < N$  time units. At  $j$ , the optimization is repeated over the new horizon  $[j, j + N]$ . This technique is motivated by the scenarios such as the following: (i) the process model is inaccurate; (ii) numerical computations introduce significant error in the control calculation; (iii) the process goals change as a result of management discretion, in the presence of process disturbances and (iv) additional modeling error is introduced to simplify the complexity of the control calculation (e.g. model discretization).

This section presents an algorithm based on MPC for the CDM model. We have chosen to build our technique on MPC based on its origin in environments where a practical need for explicit parameter re-calibration and revision of the suggested control inputs was necessitated by frequent environment changes, known model inadequacies, and uncertainty in measurements.

**Algorithm 1** (*A model-predictive control algorithm for the CDM model*).

```

1 Let  $t_R$  be the Revenue Release date.
2 Let Checkpoints be an ordered set of dates  $\{c_1, c_2, \dots, c_N\} < t_R$ .
3 for  $i = 1$  to  $N$  do
4   Train CDM to all data up to  $c_i$ 
5   Linearize CDM about the nominal trajectory at  $c_i$ 
6   Discretize the linearized model
7   Solve the optimal control problem over  $[c_i, t_R]$  with the linearized
   discretized model.
8   Implement the resulting control until  $c_{i+1}$ 
9 end

```

For example, consider the discussion in Section 4.1 regarding parameter and modeling error, and the fact that at any checkpoint an arbitrary change in our estimate of  $r_0$  may result from model re-calibration. Algorithm 1 provides the steps which are discussed in detail below. The algorithm builds on concepts in Dutka and Grumble (2004) combined with those of MPC; specifically, given a reference trajectory ( $x^{\text{ref}}$ ), we solve a linear quadratic optimal control problem over the interval between the current checkpoint ( $c_i$ ) and the deadline ( $t_R$ ). The resulting control values ( $w_i, \gamma$ ) over  $(c_i, c_{i+1})$  guide the management’s choice until the next checkpoint. Our algorithm makes no assumptions regarding how the STP management chooses to use the control values. At the next checkpoint ( $c_{i+1}$ ) the model parameters are re-calibrated and the process is repeated. To illustrate this, one could picture the feedback loop in Fig. 1 being executed each time a checkpoint is reached, including the parameter calibration process using all process data generated up to the checkpoint.

For each checkpoint, the following steps of Algorithm 1 are performed:

### 6.1. Retrain the model

This line instructs the user to execute the parameter estimation routines using all available process data up to the current checkpoint  $c_i$ . This includes using any parameter correction techniques that are available.

### 6.2. Linearize the model about the nominal trajectory

Using the parameter values calculated at line 4, we linearize the CDM model about the re-calculated nominal trajectory of the STP.

Let Eq. (6) be restated as

$$\dot{x} = A(u)x = f(x, u) \quad (9)$$

where  $x = [r, \dot{r}]^T$  and  $u = [w_i, \gamma]^T$ . Let the nominal input  $u^* \equiv [w_i^*, \gamma^*]^T$  reflect the previously implemented control, and the test manager’s current plans regarding workforce allocation and process quality improvement over the remainder of the project. Then  $x^*$ , the solution to the differential equation  $\dot{x} = f(x, u^*)$ , is the ‘nominal trajectory’ or *expected behavior* of the test process, given  $u^*$ .

For perturbations about the nominal behavior, the perturbed state  $x$  satisfies

$$\dot{x} = \dot{x}^* + \Delta \dot{x} = f(x^* + \Delta x, u^* + \Delta u)$$

$$\approx f(x^*, u^*) + \left. \frac{\partial f}{\partial x} \right|_{x^*, u^*} \Delta x + \left. \frac{\partial f}{\partial u} \right|_{x^*, u^*} \Delta u$$

for sufficiently small  $\Delta x$  and  $\Delta u$ ; where

$$\left. \frac{\partial f}{\partial x} \right|_{x^*, u^*} = \begin{bmatrix} 0 & 1 \\ -\frac{\zeta w_i^*}{s_c(1+b)} & -\frac{\xi}{\gamma^* s_c} \end{bmatrix} \quad (10)$$

$$\left. \frac{\partial f}{\partial u} \right|_{x^*, u^*} = \begin{bmatrix} 0 & 0 \\ -\frac{\zeta \gamma^*}{s_c(1+b)} & \frac{\xi \gamma^*}{\gamma^{*2} s_c} \end{bmatrix} \quad (11)$$

Thus, by subtracting out  $\dot{x}^* = f(x^*, u^*)$ , we arrive at the linearized delta-model

$$\dot{\Delta x}(t) = \bar{A}(t)\Delta x(t) + \bar{B}(t)\Delta u(t) \quad (12)$$

where  $\bar{A}(t) = \partial f / \partial x|_{x^*(t), u^*(t)}$  and  $\bar{B}(t) = \partial f / \partial u|_{x^*(t), u^*(t)}$  as defined in Eqs. (10) and (11).

### 6.3. Discretize the linearized model

Let  $k$  denote a time index in discrete time, and let  $t_k$  denote the equivalent time on the continuous time scale. We then define the time-step  $T = t_{k+1} - t_k$  as the duration spanned between discrete time values. For example, our work assumes that  $T = 1$  day.

Given these preliminaries, and assuming that  $A(t) = A_k$  is constant over the time interval  $[kT, (k+1)T]$ , the discretized version of the linearized model of Eq. (12) is given by the following system of difference equations (Chen, 1999; DeCarlo, 1989):

$$\Delta x((k+1)T) = e^{\bar{A}_k T} \Delta x(kT) + \left[ \int_{kT}^{(k+1)T} e^{\bar{A}_k((k+1)T-\tau)} \bar{B}(\tau) d\tau \right] \Delta u_k$$

This result is derived from the fact that  $\frac{d}{dt} e^{At} = A e^{At} = e^{At} A$  (Chen, 1999). By pre-multiplying both sides of Eq. (12) by  $e^{-\bar{A}_k t}$  and simplifying, we arrive at the equation

$$\frac{d}{dt} \left( e^{-\bar{A}_k t} \Delta x(t) \right) = e^{-\bar{A}_k t} \bar{B}(t) \Delta u(t) \quad (13)$$

Integrating both sides of Eq. (13) over the time interval  $[kT, (k+1)T]$ , and using  $(e^{-\bar{A}_k t})^{-1} = e^{\bar{A}_k t}$  (Chen, 1999), we arrive at the system of difference equations above, which may be re-written as

$$\Delta x((k+1)T) = \tilde{A}(kT) \Delta x(kT) + \tilde{B}(kT) \Delta u(kT) \quad (14)$$

where the input perturbation

$$\Delta u_k = \Delta u(kT) = [\Delta w_f(kT), \Delta \gamma(kT)]^T$$

is again assumed constant over the interval  $[kT, (k+1)T]$ , and

$$\tilde{A}(kT) = e^{\bar{A}_k T}; \quad \tilde{B}(kT) = \left[ \int_{kT}^{(k+1)T} e^{\bar{A}_k((k+1)T-\tau)} \bar{B}(\tau) d\tau \right]$$

### 6.4. Solve an optimal control problem with the discretized model

This begins a Model Predictive Control step. In MPC, one solves an optimal control problem over a (typically) finite horizon with respect to a given cost functional. In the case of a discrete-time MPC problem, the cost functional is typically a summation, over the optimization hori-

zon, of quadratic forms in the state and input vectors representing (i) the cost of deviation from the desired state trajectory balanced against (ii) the cost incurred by the effort required to correct the deviant trajectory (i.e. to reach the desired error reduction).

For our algorithm, the optimization horizon is from the present checkpoint to the Revenue Release date; thus each successive iteration of the algorithm optimizes over a smaller horizon.

The solution of the optimal control problem yields the control sequence  $\Delta u \equiv [\Delta w_f, \Delta \gamma]^T$  (where the lack of a time-index indicates a vector of elements; one for each time-index) such that the cost functional is minimized over all possible changes to the STP process. Thus the control input—which is the best set of changes to be made in  $w_f$  and  $\gamma$  (considering the cost functional) per time-step over the control horizon—is combined with the nominal input sequence to provide the suggested parameters for the STP:  $u(t) \equiv u^*(t) + \Delta u(t)$ . In our scenario, this input sequence is the set of process changes necessary to best meet the quality and schedule requirements associated with the Revenue Release gate (i.e. to optimize between budget and schedule adherence).

The technique we have chosen to solve the control problem is known as Constrained Linear Quadratic Tracking, which can be stated as a convex multi-parametric Quadratic Programming problem (Bemporad et al., 2000; Borrelli, 2002, 2003; Goodwin et al., 2005; Kerrigan and Maciejowski, 2000).

Allowing the subscript to denote the time index, we define our cost functional as

$$J(x, u) = \frac{1}{2} \sum_{\tau=1}^N \| (x_{\tau}^{\text{ref}} - x_{\tau}^* - \Delta x_{\tau}) \|_{Q_{\tau}}^2 + \frac{1}{2} \sum_{\tau=0}^{N-1} \| (u_{\tau}^* + \Delta u_{\tau}) \|_{R_{\tau}}^2 \quad (15)$$

where  $Q$  and  $R$  are positive definite matrices representing the cost of *state deviation* and the cost of applying *control input*, respectively. Eq. (15) is composed of two summations; the first computes the per-step cost of deviation from the desired trajectory ( $x^{\text{ref}}$ ) by the controlled trajectory ( $x^* + \Delta x_{\tau}$ ). The second summation accumulates the per-step cost of the entire input to the system ( $u_{\tau}^* + \Delta u_{\tau}$ ). Note that both costs are quadratic, with respect to  $Q$  and  $R$ , respectively.

To acquire the form of a quadratic program, the formula for the discretized linearized state model (Eq. (14)) is substituted in place of the  $\Delta x_{\tau}$  term in the cost functional (Eq. (15)) for each  $\tau$  in the summation. This substitution is repeated until all  $\Delta x_{\tau}$  terms have been reduced to a  $\Delta x_0$  dependence. Next, the definition  $\|a\|_M^2 = a^T M a$  is used to remove the norms. After collecting terms and simplifying, we arrive at the following form:

$$\Delta u^T H \Delta u + \Delta u^T v \quad (16)$$

where

$$H = \widehat{B}^T \widehat{Q} \widehat{B} + \widehat{R}; \quad v = \widehat{B}^T \widehat{Q} (\widehat{A}x_0^* - \widehat{x}^{\text{ref}} - \widehat{x}^*) + \widehat{R}u^*$$

$$\widehat{A} = \begin{bmatrix} \widetilde{A}_0 \\ \widetilde{A}_1 \widetilde{A}_0 \\ \widetilde{A}_2 \widetilde{A}_1 \widetilde{A}_0 \\ \vdots \end{bmatrix}; \quad \widehat{B} = \begin{bmatrix} \widetilde{B}_0 & & & \\ \widetilde{A}_1 \widetilde{B}_0 & \widetilde{B}_1 & & \\ \widetilde{A}_2 \widetilde{A}_1 \widetilde{B}_0 & \widetilde{A}_2 \widetilde{B}_1 & \widetilde{B}_2 & \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

$$\widehat{Q} = \text{diag}(Q_1, Q_2, \dots, Q_N); \quad \widehat{R} = \text{diag}(R_0, R_2, \dots, R_{N-1})$$

$$\widehat{x}^{\text{ref}} = [x_1^{\text{ref}T} \quad x_2^{\text{ref}T} \quad \dots \quad x_N^{\text{ref}T}]^T; \quad \widehat{x}^* = [x_1^{*T} \quad x_2^{*T} \quad \dots \quad x_N^{*T}]^T$$

This is a multi-parametric Quadratic Program (mp-QP), which can be readily solved by standard quadratic program solvers such as MATLABs *quadprog* (Coleman et al., 1999). The convexity of the program guarantees a globally optimal solution with respect to the linearized and discretized model.

To mitigate the error induced by the linearization and discretization, we iterate the optimal control calculation according to Algorithm 2.

In our experience with the CDM model, further iterations after the second cease to induce a noticeable difference in the computed (iterated) control suggestion.

The reference trajectory ( $x^{\text{ref}}$ ) is given as the exponential function which decays from  $r_0$  at time  $t_0$  to  $r_{\text{goal}}$  at time  $t_R$ . This is motivated by the common knowledge that defect detection exhibits exponential decay over time given a testing technique (Horgan and Mathur, 1996).

**Algorithm 2** (Iterative mitigation of the linearization and discretization error).

```

1 Let I be the number of iterations
2 Let  $\Delta u_{\text{iter}} = \mathbf{0}$ 
3 for  $i = 1$  to  $I$  do
4   Generate the optimal control input as per Algorithm 1  $\rightarrow \Delta u$ .
5   Update the iterated control suggestion:  $\Delta u_{\text{iter}} \leftarrow \Delta u_{\text{iter}} + \Delta u$ 
6   Update the nominal input:  $u^* \leftarrow u^* + \Delta u$ 
7   Update the nominal trajectory using the non-linear, continuous time
   model, the current state, and the updated nominal input  $\rightarrow x^*$ 
8 end

```

### 6.5. Implement the control suggestions

Lastly, the calculated control inputs are available to the management for application to the *actual process*. To determine the impact of applying the control suggestions to the actual process, we apply these to the *non-linear, non-discretized* model until the next checkpoint is reached. Following the technique of MPC, one then re-solves, at the next checkpoint, the optimal control problem over the discretized linearized model using the data from the actual process.

## 7. Benefits of optimal control with constraints

The techniques for solving mp-QP problems allow the specification of additional constraints (Bemporad et al., 2000; Borrelli, 2002, 2003; Goodwin et al., 2005; Kerrigan and Maciejowski, 2000) specified as  $L\Delta u \leq w$ . These constraints may take, among others, the following forms (Bemporad et al., 2000; Goodwin et al., 2005; Kerrigan and Maciejowski, 2000):

$$\Delta u_k^{\min} \leq \Delta u_k \leq \Delta u_k^{\max} \quad (17)$$

$$\delta_k^{\min} \leq \Delta u_k - \Delta u_{k-1} \leq \delta_k^{\max} \quad (18)$$

That is, we may place restrictions on the amount of change allowed in the control input ( $\Delta u_k$ ) at a given time step ( $k$ ) and indeed, by Eq. (18), we can specify limits on the difference allowed between consecutive control inputs. We note that because these limits are provided externally, this feature can be used to specify additional factors that influence the ability of the manager to control the STP, such as delays in the ability to bring additional resources to the STP, or to impose a preference on the granularity at which changes are allowed in the control parameters.

For example, consider a scenario where the manager of the STP has budget for four additional workforce members. However, it will be three weeks before they can be brought on-board. In this scenario, one could set the workforce component of  $\Delta u_k^{\max}$  to zero for those time indices ( $k$ ) corresponding to the three weeks, and subsequently to 4 for the remainder of the prediction/control period thereby disallowing the control calculation to suggest an addition of workforce within the 3-week period; subsequent to the three week period, the control calculation would be limited to an addition of four additional workforce members. Let us further assume that the manager desires to make only a single process change per week. To acquire this behavior from the control calculation, one may set  $\delta_k^{\max} = \alpha$  and  $\delta_k^{\min} = -\alpha$  for a large 2-vector  $\alpha$  (i.e. components as near infinity as is practical) for those  $k$  which are even multiples of 5 (assuming a 5-day work week), thereby allowing an arbitrary difference between the last control suggestion and the current. For all other periods, one sets them to zero (meaning the control suggestions on these days must be exactly what it was on the previous day).

Note that Eqs. (17) and (18) may be represented as the following set inequalities:

$$\begin{aligned} \Delta u_k &\leq \Delta u_k^{\max} \\ -\Delta u_k &\leq -\Delta u_k^{\min} \\ (\Delta u_k - \Delta u_{k-1}) &\leq \delta_k^{\max} \\ -(\Delta u_k - \Delta u_{k-1}) &\leq -\delta_k^{\min} \end{aligned}$$

which is mirrored in the following matrix formulation,  $L\Delta u \leq w$ :

$$L = \begin{bmatrix} I_N \\ -I_N \\ D \\ -D \end{bmatrix}; \quad \Delta u = \begin{bmatrix} \Delta u_0 \\ \Delta u_1 \\ \vdots \\ \Delta u_{N-1} \end{bmatrix}; \quad w = \begin{bmatrix} \Delta \hat{u}^{\max} \\ -\Delta \hat{u}^{\min} \\ \hat{\delta}^{\max} \\ -\hat{\delta}^{\min} \end{bmatrix}$$

where

$$D = \begin{bmatrix} I_2 & & & & & \\ -I_2 & I_2 & & & & \\ & -I_2 & I_2 & & & \\ & & & \ddots & \ddots & \\ & & & & & \ddots \end{bmatrix}; \quad \Delta \hat{u}^{\max} = \begin{bmatrix} \Delta u_0^{\max} \\ \Delta u_1^{\max} \\ \vdots \\ \Delta u_{N-1}^{\max} \end{bmatrix}; \quad \Delta \hat{u}^{\min} = \begin{bmatrix} \Delta u_0^{\min} \\ \Delta u_1^{\min} \\ \vdots \\ \Delta u_{N-1}^{\min} \end{bmatrix}$$

$$\hat{\delta}^{\max} = \begin{bmatrix} \delta_0^{\max} \\ \delta_1^{\max} \\ \vdots \\ \delta_{N-1}^{\max} \end{bmatrix}; \quad \hat{\delta}^{\min} = \begin{bmatrix} \delta_0^{\min} \\ \delta_1^{\min} \\ \vdots \\ \delta_{N-1}^{\min} \end{bmatrix}$$

where  $I_N$  and  $I_2$  are the  $N \times N$  and  $2 \times 2$  identity matrices, respectively.

## 8. Choosing the cost matrices

This section provides an overview of the issues to be addressed in selecting the cost matrices (i.e.  $Q$  and  $R$ ) for the cost functional in Eq. (15), and proposes a technique for generating reasonable cost matrices.

The technique we propose for the selection of the cost matrices arises with regard to the existence of constraints on the control values (e.g. in the CDM model, the process quality  $\gamma$  may not range above 1.0). In selecting the cost matrices with respect to constraints on the control inputs, we propose that one should first equalize the cost of reaching the limits. For simplicity, we consider the case of diagonal cost matrices.

For example, consider limits on  $\Delta w_f$  and  $\Delta \gamma$  of 5 and 0.18, respectively. For an input-cost matrix  $R = \text{diag}(r_1, r_2)$ , and control input vector  $\Delta u = [\Delta w_f, \Delta \gamma]^T$ , the quadratic form  $\Delta u^T R \Delta u = \Delta w_f^2 r_1 + \Delta \gamma^2 r_2$ . By pushing  $\Delta u$  to its limits, we arrive at the maximal cost  $25r_1 + 0.0324r_2$ .

Balancing the contribution to the maximal cost made by each component of the input,

$$\frac{25r_1}{0.0324r_2} = 771.60 \frac{r_1}{r_2} = 1$$

we see that to balance the cost of maximizing the control inputs, we see that weight  $r_2$  should be  $\approx 772$  times larger than  $r_1$ . This provides a ‘level field’ as the starting point for making adjustments *by ratio* to reflect the management’s experience. For example, if a 10% quality improvement costs twice as much as a 10% workforce increase, then  $r_2$  could be increased by a factor of 2.

The other aspect to consider is the balance between the cost incurred by implementing control (via the  $R$  matrix) and the cost incurred by deviating from the desired trajectory  $x^{\text{ref}}$  calculated by the  $Q$  matrix. The technique suggested here is to select a  $Q$  that is large and scale it down

if the resulting control is infeasible. We note that the component of  $Q$  which assigns cost to the derivative term is likely to be small compared to that which assigns cost to the magnitude term.

For example, consider assigning  $Q = \text{diag}(1, 0.001) \times \|R\|^2$ , which weights the magnitude term of the  $Q$  matrix much higher than the derivative term, and sets a proportion between the  $Q$ -cost (cost of state deviation) and  $R$ -cost (cost of input changes) to prefer input changes over state deviations. We note that in the case where reaching the deadline on time is more important than the cost incurred in doing so, that the mp-QP solvers will likely include an interface for specifying state constraints (i.e. requirements on both the initial- and final-state) which could be employed to force a desired final state.

## 9. An illustrated example

This section presents an example usage of the technique presented in Section 6. The data set is the same used in Cangussu et al. (2004). A brief description of the nature of the software product upon which the study is conducted is given in Section 1.

The execution of the STP that generated the study data lasted 120 days over which a number of defects were discovered and removed (95% by estimation of the testing team).

Our example will impose an artificial deadline and quality goal onto this STP execution and determine the changes that would have been required within the test environment in order to meet our accelerated deadline and more ambitious quality goal. As with a live STP, we re-calibrate the CDM model at checkpoints within the STP to incorporate newly available data, and, in theory, improve the model predictions and consequent control suggestions.

To simulate the increasing availability of parameter tuning data, we calibrate the CDM model using successively larger contiguous subsets of the total data starting at day 1. We have chosen to set the checkpoints 10-days apart (i.e. two 5-day work weeks) after the 30th day. Thus each successive re-calibration of the CDM model incorporates 10 additional days of training data. This is reflected in Figs. 4–6 by the vertical line marking the end of the data collection period the data from which is used to calibrate the model parameters which shape the plot. The data required to train the model is the number of defects detected and eliminated per day of the STP. Such data is likely to be available in the reporting capability of any industrial-strength defect tracking system.

As the data set only describes the execution of the STP in the absence of control, our example makes the assumption that the dynamics of the STP, modulo the effect of the inputs, are not significantly altered by the implementation of the control suggestions in the STP. Stated differently, we assume that the changes in the dynamics of the STP that would have been elicited by implementing the control suggestions are adequately described by the action of the input

variables on the CDM model. We note that this assumption is made mandatory not as a consequence of the model, but rather as a consequence of using data from a completed project to illustrate a technique which is intended to be used in an on-line manner.

Fig. 3 shows the actual defect detection/removal data, normalized to the total number of defects discovered through the entirety of the STP. The plot marks the point at which the software manufacturer decided to release the project (i.e. the ‘actual’ Revenue Release) with a circle on the ‘Normalized Defect Removal’ trajectory, to illustrate the errors found subsequently, and thus the accuracy of the Revenue Release date selection.

For our example we have chosen a deadline at 70 days, and a quality objective of 90% defect reduction. We have chosen the limits of  $5 \leq w_t \leq 12$  and  $0.70 \leq \gamma \leq 0.98$ , leading to cost matrices  $Q$  and  $R$  set as  $R = \text{diag}(1.00, 772.00)$  and  $Q = \text{diag}(1, 0.001) \times \|R\|^2$  where the choice of values in the  $R$  matrix indicates that the cost of a 10% workforce increase is roughly equivalent to a 10% improvement in the test process quality, and the resulting norm of the  $Q$  matrix will generate a strong bias for reaching the reference trajectory regardless of the input cost required. Section 8

describes the mechanism for selecting these parameters. We have further made the choice that the control suggestions should only change once per week, and thus have used the values of  $\delta^{\max}$  and  $\delta^{\min}$  as per the example in Section 7. The remainder of the parameters are taken from the actual study in Cangussu et al. (2004).

The data in the following plots is normalized to a percentage of the total defects found during the entirety of the STP. The calibration of the CDM model includes methods for estimating the initial number of defects ( $r_0$ ) from the defect removal productivity (Cangussu et al., 2002). All plots begin from the current estimate of  $r_0$  and proceed according to the dynamics of the CDM model. Thus Fig. 4a shows the initial number of defects as nearly 160%—this means that the estimate of  $r_0$  at this point was around 160% of the total number of defects that would later be discovered.

In Figs. 4–6 we apply our technique to the data using our modified parameter calibration routine (Section 11.2). Fig. 4a is an example of poor calibration of the CDM model. Note that the original estimate of  $r_0$  is 160% of the number of defects that will eventually be found. The reference trajectory plots the ideal route to 90% defect reduction at day 70, and the ‘new’ trajectory illustrates the expected trajectory of the STP if the controls had been implemented. Note that given the large number of defects between the nominal and reference trajectory, the controller was not able to drive the STP to the desired trajectory—this is because the limits on the control parameters were reached.

There is now a choice as to how to proceed. Given the number of defects predicted at day 30, the management may believe the prediction, or may reject it due to inconsistency with prior experience. We will extend the example in both directions, starting with the former. Fig. 4b uses the assumption that the control suggestions from the prior checkpoint were implemented. Note that the new estimate of  $r_0$  is a little over 80% of the total defects that will be discovered. The nominal trajectory at this point contains the controls that the manager was planning to implement given the prediction at the 30-day checkpoint. It is apparent from the current prediction that these controls are no longer deemed necessary, thus the controller suggests that the

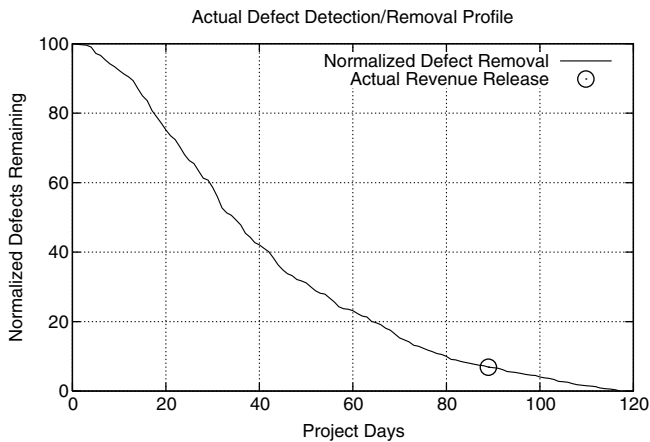


Fig. 3. Remaining defects: Actual defect detection/removal profile based on total number of defects found in the STP. Normalized to the total number of defects found during the STP.

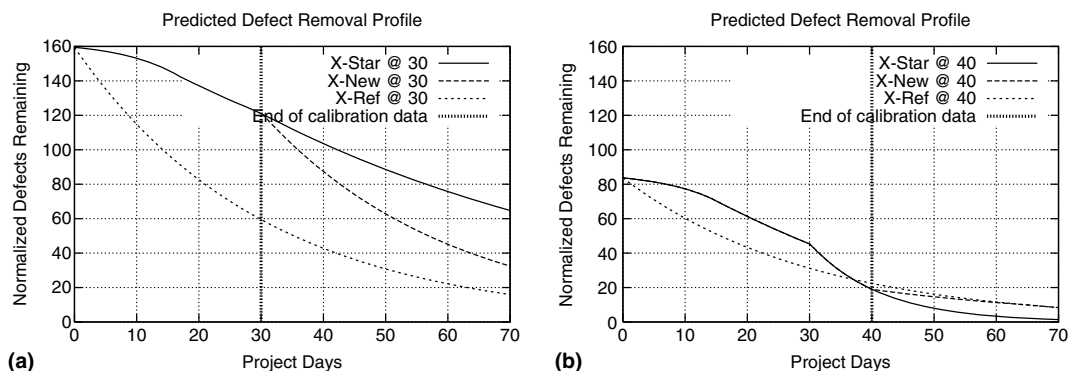


Fig. 4. Prediction and estimated control impact: (a) calibrated to 30 days data; (b) calibrated to 40 days data, believing day 30 prediction.

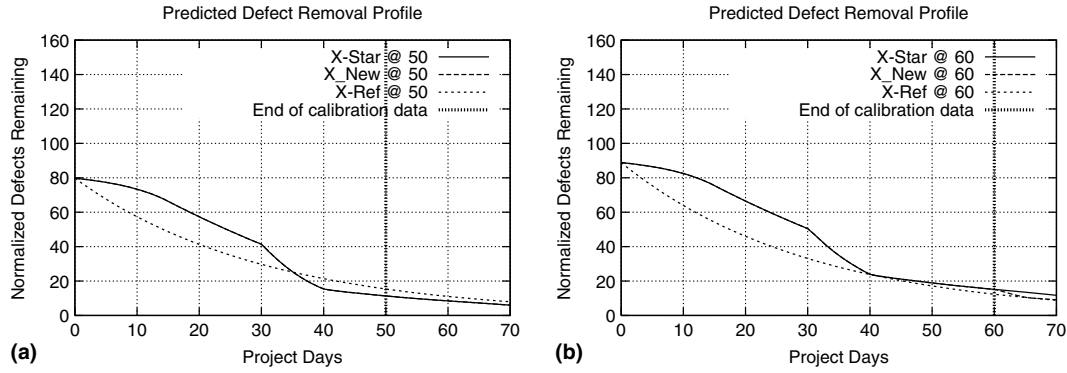


Fig. 5. Prediction and estimated control impact: (a) calibrated to 50 days data; (b) calibrated to 60 days. Assumes belief in day 30 prediction.

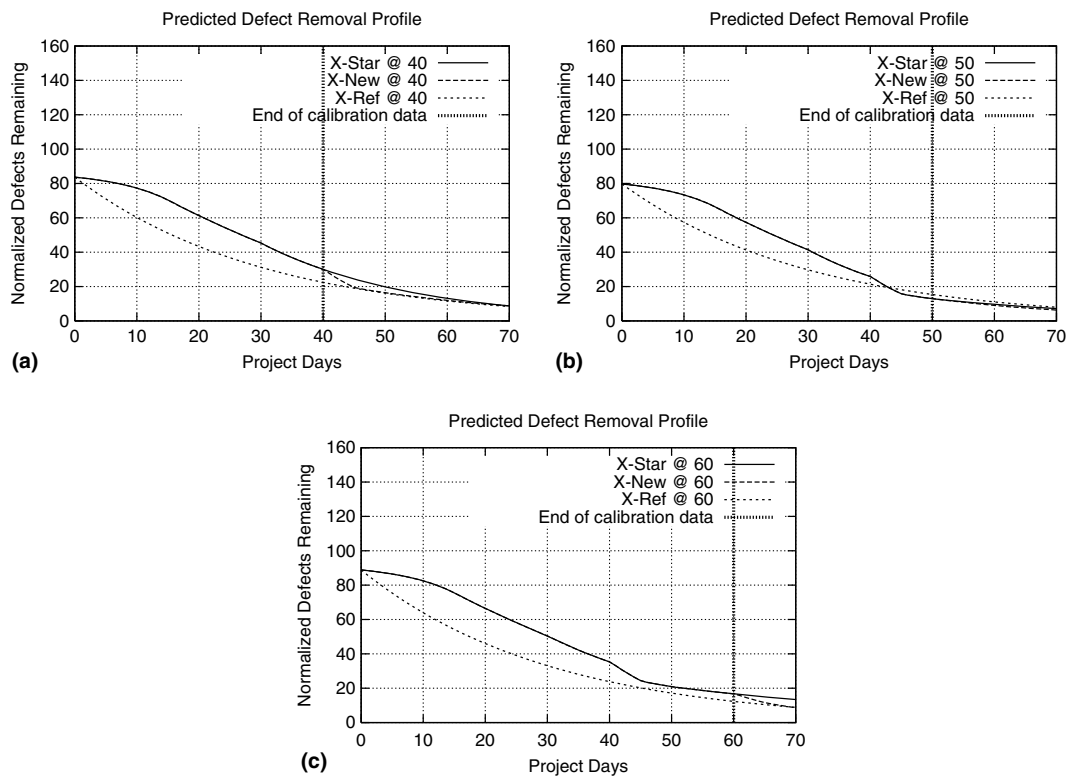


Fig. 6. Prediction and estimated control impact: (a) calibrated to 40 days data; (b) calibrated to 50 days; (c) calibrated to 60 days data. Assumes rejection of the day 30 prediction.

manager pull the resources and the controlled trajectory exhibits a reduction in the defect detection/removal rate. It is worthy of note that at this point that the controller has pushed the control parameters to their lower limits because the two weeks of implementing the previous control suggestions have pushed the nominal trajectory of the STP well below the reference trajectory.

The remaining checkpoints at 50- and 60-days exhibit a similar situation. Fig. 5a shows that the initial defect estimate is relatively unchanged from day 40, and that the controller is still trying to remove resources from the STP in order to compensate for the bad judgment at day 30. Fig. 5b shows an increase in the 60 day  $r_0$  estimate from

that made at day 50, and consequently, that there is a small increase in the control parameters is necessary to track the desired trajectory.

Returning now to the second scenario where the test manager rejects the prediction at 30 days, we see a different controller reaction. In Fig. 6a, even though the nominal trajectory is on-track to reach the reference trajectory at day-70, the control calculation determined that it is more cost effective to bring additional resources to the STP and then release them later, down to near the minimum workforce that we have allowed. Fig. 6b shows that the new estimate of  $r_0$  causes the CDM model to predict a trivial achievement of the goal at day 70, and thus it

reduces the resources on the project to the lower limits we have set. Lastly, Fig. 6c again shows an increase in the  $r_0$  estimate, which requires the controller to bring resources back to the project in order to meet the 70-day goal.

## 10. Applying the approach in practice

Here, we describe the data required to apply this technique to an actual STP in progress. There are two sets of data required: (i) that data required by the CDM model for calibration and simulation and (ii) that data required by the controller in the form of constraints and cost factors required by the control algorithm.

The CDM model has 7 parameters: (i) The workforce size ( $w_f$ )—the average number of people working at a given time; (ii) The quality of the test process ( $\gamma$ )—a real number in the range  $[0, 1]$ ; (iii) The complexity of the software to be developed ( $s_c$ ); (iv) The software type being developed ( $b$ )—this is the same factor from the COCOMO family of models; (v) The defect detection constant of proportionality ( $\zeta$ ); (vi) The quality impact constant of proportionality ( $\xi$ ) and (vii) An estimate of the number of defects introduced into the software ( $r_0$ ). The first four are calculated or measured as per the instructions in Cangussu et al. (2002), the last three are estimated from the project data by the Parameter Identification algorithms in the subsequent sections. It should be noted that the data required from the project by the Parameter Identification algorithms is an array of the number of defects removed per unit time (our example used days).

The remaining pieces of data required to put the system into use are: (i) The schedule and quality objectives, that is, a list of dates and the desired proportion of defects that is to be remaining in the software once each date is reached—this data determined the desired/reference trajectory; (ii) The weight matrices ( $Q$  and  $R$ ) that calibrate the controller's cost function/performance index—the computation of which is given in Section 8; (iii) The control constraints to be placed on the system—any maximum or minimum values on  $\Delta w_f$  and  $\Delta \gamma$  from within which the controller must select its control suggestions.

Once these data have been acquired, one runs the parameter identification algorithm which calibrates the CDM model to the existing data. Then, we use the expected values of  $w_f$  and  $\gamma$  (the 'nominal inputs') to run the CDM model and gain a prediction which we will call the 'nominal trajectory'. If the nominal trajectory is unsatisfactory i.e. fails to meet the quality and schedule objectives, it is plugged into the control algorithm along with a reference trajectory and the nominal inputs. Assuming that the cost matrices and control constraints have been supplied, the control algorithm will generate a suggestion for changes in the nominal inputs based on the cost function.

Assuming the change suggestions are accepted, one makes the process changes that correspond to the new nominal input. At the next checkpoint, the process is

repeated by running the parameter identification algorithm to incorporate any new data from the process, and by using the updated nominal inputs to generate an updated nominal trajectory using the newly calibrated CDM model.

## 11. Parameter identification

The calculation of the parameters  $w_f$ ,  $s_c$ ,  $b$ , and  $\gamma$  is discussed in Cangussu et al. (2002). There are two parameters,  $\xi$  and  $\zeta$ , which are calibrated to project data. We first summarize the method of their computation as given in Cangussu et al. (2002), and then proceed to offer a modified version which leverages more of the constraints implied by the model equations. In each case we cover the details of using the parameter identification technique in a scenario of recurring re-calibration over multiple training periods which are believed to encapsulate different behaviors of the STP.

### 11.1. The original calibration algorithm

The following subsection is an exposition of the details of calibration technique given in Cangussu et al. (2002) and forms the basis for the algorithm in Section 11.2. For simplicity, we will refer to this calibration algorithm as *Call*.

#### 11.1.1. Estimating $\zeta$ and $\xi$

Recall that  $x(t) = [r(t), \dot{r}(t)]^T$  is the state vector of the CDM model. Let  $d(k) = r(kT) - r((k-1)T)$  be the number of defects eliminated over the interval  $[(k-1)T, kT]$ , where  $T$  is the duration separating consecutive discrete time indices. We note that for calibration purposes,  $r(\cdot)$  and  $\dot{r}(\cdot)$  are unknown; only  $d(\cdot)$  is measurable.

Define

$$D(k) = x(kT) - x((k-1)T) = \begin{bmatrix} d(k) \\ \dot{r}(kT) - \dot{r}((k-1)T) \end{bmatrix} \quad (19)$$

as the difference between consecutive CDM model states.

From the solution of the differential equation model in Eq. (5), we know that  $r(\cdot)$  is a linear combination of two distinct exponential functions. To obtain an estimate of the "velocity" term,  $\dot{r}(\cdot)$ , we crudely approximate  $r(\cdot)$  by a single first-order exponential, and use the derivative as a coarse velocity estimate,  $\hat{r}(\cdot)$ .

Let  $\hat{r}(t) = e^{\lambda t} \alpha$  be a first-order exponential approximation of  $r(t)$ . It can then be shown that there exists a scalar factor  $m = e^{\lambda T}$  such that

$$d(k) \approx md(k-1) \quad (20)$$

by

$$\begin{aligned} md(k-1) &\approx e^{\lambda T} e^{\lambda(k-1)T} \alpha - e^{\lambda T} e^{\lambda(k-2)T} \alpha \\ &\approx r(kT) - r((k-1)T) = d(k) \end{aligned}$$

The following is Least Squares fit of  $m$  to the  $N$  available data points  $d(\cdot)$ :

$$m = [d(2) \ d(3) \ \dots \ d(N)][d(1) \ d(2) \ \dots \ d(N-1)]^{-R} \tag{21}$$

where the superscript ‘-R’ indicates the right pseudo-inverse. Having  $m$  we can derive  $\lambda$  from our assumption of the exponential form of  $m$  by  $\lambda = 1/T \log m$ . Upon calculating  $\alpha$ , one will have sufficient information to use the assumed equations for  $\hat{r}(\cdot)$  and  $\hat{i}(\cdot)$  directly.

For an estimate of  $\alpha$  we turn to the definition of

$$d(k) = r(k) - r(k-1) \approx \alpha(e^{\lambda kT} - e^{\lambda(k-1)T}) = \alpha(m^k - m^{(k-1)})$$

and thus a Least Squares approximation of  $\alpha$  to the  $N$  data-points of  $d(\cdot)$  is given by

$$\alpha = \begin{bmatrix} m-1 \\ m^2-m \\ \vdots \\ m^N - m^{N-1} \end{bmatrix}^{-L} \begin{bmatrix} d(1) \\ d(2) \\ \vdots \\ d(N) \end{bmatrix} \tag{22}$$

where the superscript ‘-L’ indicates a left pseudo-inverse.

One may now build estimates of  $D(\cdot)$  using Eq. (19), and the assumed form of  $\hat{r}(\cdot)$ . That is

$$D(k) \approx \begin{bmatrix} d(k) \\ \lambda(e^{\lambda kT} - e^{\lambda(k-1)T})\alpha \end{bmatrix} \tag{23}$$

Having an estimate for  $D(\cdot)$ , we may now address the problem of parameter identification. A simulation of the CDM model is given by

$$x(kT) = M_{p(k)}x((k-1)T) \tag{24}$$

where the function  $p(k)$  returns the training period to which the time index  $k$  belongs, and  $M_j = e^{A_jT}$  is constant over the training period  $j$ . It is worthy of mention that the  $A_j$  matrices are presumed to be the system dynamics matrix in Eq. (6).

In a manner which closely parallels the calculation of the first-order exponential fit above, it can be shown that

$$D(k+1) \approx M_{p(k+1)}D(k) \tag{25}$$

It is an approximation of the average behavior of the STP that is desired here, thus one may restate Eq. (25) as

$$D(k+1) \approx \check{M}D(k) \tag{26}$$

which, solving for  $\check{M}$  yields the Least Squares approximation

$$\check{M} = [D(2) \ D(3) \ \dots \ D(N)][D(1) \ D(2) \ \dots \ D(N-1)]^{-R} \tag{27}$$

Once  $\check{M}$  is obtained, the Spectral Mapping Theorem (DeCarlo, 1989) states that the following relationship exists between its eigenvalues and those of the average system dynamics matrix,  $\check{A}$ :

$$\lambda_i^{\check{A}} = \frac{1}{T} \log(\lambda_i^{\check{M}}) \tag{28}$$

where  $\lambda_j^P$  is the  $j$ th eigenvalue of matrix  $P$ . The eigenvalues of the  $\check{A}$  matrix are the roots of its Characteristic Polynomial,  $\Pi(\check{A})$ , which, by the equation given in Cangussu et al. (2002), yields

$$\begin{aligned} \Pi(\check{A}) &= \det[\lambda I - \check{A}] = \lambda^2 + \frac{\xi}{\gamma s_c} \lambda + \frac{\zeta w_f}{s_c^{(1+b)}} \\ &= (\lambda - \lambda_1^{\check{A}})(\lambda - \lambda_2^{\check{A}}) \\ &= \lambda^2 - (\lambda_1^{\check{A}} + \lambda_2^{\check{A}})\lambda + \lambda_1^{\check{A}}\lambda_2^{\check{A}} \end{aligned} \tag{29}$$

Thus the parameter identification problem reduces to finding  $\xi$  and  $\zeta$  to satisfy Eq. (29). This occurs with

$$\xi = -(\lambda_1^{\check{A}} + \lambda_2^{\check{A}})\gamma s_c; \quad \zeta = \frac{\lambda_1^{\check{A}}\lambda_2^{\check{A}}s_c^{(1+b)}}{w_f} \tag{30}$$

### 11.1.2. Estimating $x_0$

By taking the notion of average behavior proposed in Eq. (26) and applying it to Eq. (24), we derive a less-detailed simulation

$$x(kT) = \check{M}x((k-1)T) \Rightarrow x(kT) = \check{M}^k x_0 \tag{31}$$

A method for estimating  $x_0$  is derived by substituting Eq. (31) into Eq. (19) resulting in

$$D(k) = \check{M}^k x_0 - \check{M}^{k-1} x_0 = (\check{M}^k - \check{M}^{k-1})x_0 \tag{32}$$

which defines, analogous to Eq. (22) in the scalar case,  $N$  constraints for a Least Squares fit, given  $N$  values of  $D(\cdot)$  as

$$x_0 = \begin{bmatrix} M-1 \\ M^2-M \\ \vdots \\ M^N - M^{N-1} \end{bmatrix}^{-L} \begin{bmatrix} D(1) \\ D(2) \\ \vdots \\ D(n) \end{bmatrix} \tag{33}$$

Initially, we require that  $\dot{r}(0) \approx 0$ , a constraint not implemented by standard Least Squares fitting. Thus a Weighted Least Squares approach is suggested. The calculation of  $x_0$  is modified according to

$$x_0 = ((PW)^T(PW))^{-1}(PW)^T Z \tag{34}$$

where

$$P = \begin{bmatrix} \check{M} - I \\ \check{M}^2 - \check{M} \\ \vdots \\ \check{M}^n - \check{M}^{n-1} \end{bmatrix}; \quad Z = \begin{bmatrix} D(1) \\ D(2) \\ \vdots \\ D(n) \end{bmatrix}; \quad W = \begin{bmatrix} w_{r_0} & 0 \\ 0 & w_{v_0} \end{bmatrix}$$

and the weights are modified from Cangussu et al. (2002) to the current recommendation:

$$w_{r_0} = \frac{N + e^{-d/(2N)}}{\sigma_r}; \quad w_{v_0} = \frac{N + e^{-d/(2N)}}{\sigma_{\dot{r}}}$$

where  $d$  is the expected a priori completion date (in days, from the start) of the STP,  $\sigma_r$  and  $\sigma_{\dot{r}}$  are the standard deviation of the estimates, over time, of  $r(\cdot)$  and  $\dot{r}(\cdot)$ , respectively, which were used in estimating  $D(\cdot)$ , and  $N$  is the number of data points included in the current standard deviation computations.

### 11.1.3. Re-estimating $x_0$

Here we address the issues associated with the re-calculation of  $x_0$  once more data is available, or the adjustment of  $x_0$  as per the linear parameter corrector proposed in Section 4.2. The method of estimation is still that given in Section 11.1.2, and the primary issue is that of updating the other calibrated parameters for prior training periods to be consistent with a new  $x_0$ .

Each training period captures a different behavior of the STP, and thus data from a particular period cannot be used to re-estimate parameters from any other period. The current estimate of  $x_0$  however, is used across all of the periods. Because the CDM model is exponential, a change in the initial state will produce a change in the derivative of the response, thus the eigenvalues for the  $A$  matrices of the earlier periods must be adjusted if they are to predict a defect elimination behavior consistent with the observed data for the period. Consider the correspondence between the observed data and the prediction given Eqs. (31) and (32) under a changing  $x_0$  where  $\dot{M}$  is constant.

The question is essentially that of “how does one force the calibration routines to re-interpret the data  $d(\cdot)$  in the context of a new  $x_0$  estimate?”. The methods given in the previous subsections generate a  $\zeta$  and  $\xi$  using an implicit assumption of  $x_0$  that best explains the observed data for the period. Since we have no new data which is valid for previous periods, a subsequent execution of the given methods would produce no changes. A new method capable of including an explicit external  $x_0$  estimate is required.

Let

$$\bar{r}(k) = r_0 - \sum_{i=1}^k d(i) \quad (35)$$

be a *subjective* estimate of  $r(\cdot)$  where  $r_0$  is taken from the current estimate of  $x_0$ . Again using the first-order exponential form as an approximation, (i.e.  $\bar{r}(t) \approx e^{\lambda t} r_0$ ) it can be shown that the relation

$$\lambda = \frac{1}{t} [\log(r(t)) - \log(r_0)] \quad (36)$$

holds for all  $t$ , thus defining  $N + 1$  constraints for a Least Squares fit, where  $N$  is the number of data points for  $d(\cdot)$  that are available for the period.

From this result we can estimate  $\dot{r}(\cdot)$  by taking the derivative of  $\bar{r}(\cdot)$  yielding  $\dot{r}(t) \approx \dot{\bar{r}}(t) = \lambda e^{\lambda t} r_0$ . This provides enough information to re-estimate the values of  $D(\cdot)$  for the period, and the techniques of Section 11.1.1 may be applied to calculate values for  $\zeta$  and  $\xi$  for the period which are consistent with the current estimate of  $x_0$ .

### 11.1.4. Re-estimating $\zeta$ and $\xi$

The parameter correction algorithm of Section 4.2 may be applied to both  $\zeta$  and  $\xi$ , in addition to  $x_0$ . Because the different training periods represent different behaviors of the STP, re-estimation of  $\zeta$  and  $\xi$  in this manner should only be applied to the estimate for the current period. Those for previous periods have already been adjusted to reproduce the observed data (i.e.  $D(\cdot)$ ) for the period in the manner described in Section 11.1.3.

## 11.2. An alternative calibration algorithm

Here we present a modification of the original parameter calibration algorithm (Cal1) summarized in Section 11.1 which we will refer to as *Cal2*. This algorithm takes advantage of the  $O(N^2)$  constraints that are implied in many of the Least Squares fits in the previous Subsection, where Cal1 only makes use of  $O(N)$  of the constraints. The key observation is that many of the equations also imply multiple-step relations where they are currently only used in single-step fashion.

The first such instance is the calibration of the scalar  $m$  in the computation of the first-order approximation of  $r(\cdot)$ , where the relation between  $d(k)$  and  $d(k - 1)$  in Eq. (20) further implies

$$\begin{aligned} d(k + 1) &\approx md(k) \\ d(k + 2) &\approx md(k + 1) \approx m^2d(k) \\ &\vdots \\ d(k + n) &\approx m^n d(k) \end{aligned}$$

for all valid combinations of  $k$  and  $n$ . By substituting the exponential form of  $m$ , and rearranging the equation we arrive at

$$\lambda = \frac{1}{nT} (\log(d(k + n)) - \log(d(k))) \quad (37)$$

which defines a set of  $N(N - 1)/2$  constraints for Least-Squares fit to the data, where  $N$  is the number of values of  $d(\cdot)$  available for a given training period.

We note, given the form of the approximation  $\hat{r}(k)$ , that the derivative,  $\hat{r}'(k) = \lambda e^{\lambda k T} \alpha = \lambda m^k \alpha = \lambda \hat{r}(k)$ , and thus

$$\dot{r}(kT) - \dot{r}((k - 1)T) \approx \lambda r(kT) - \lambda r((k - 1)T) = \lambda d(k)$$

Substituting this result into Eq. (19), we arrive at an alternative approximation for  $D(\cdot)$  which exhibits variance in the derivative term in accordance with the variance in the magnitude term:

$$D(k) \approx \begin{bmatrix} d(k) \\ \lambda d(k) \end{bmatrix} \quad (38)$$

In a manner parallel to that of the scalar case above, we note that Eq. (25) also implies a multiple-step relation given by

$$D(k+n) \approx \left( \prod_{i=1}^n M_{p(k+i)} \right) D(k) \quad (39)$$

### 11.2.1. Simulation

To simulate the model calibrated under the Cal2 method, one first estimates  $M_i$  for each period. To capture the behavior of a particular training period, one must partition the data according to training period. More formally, if the training period  $i$  spans the interval  $[aT, bT]$ , then the set  $S_i = \{D(j) | j \in \mathbb{Z}, (a-1) \leq j \leq b\}$  is the data partition for training period  $i$ . For the  $N_i$  members of  $S_i$ , the relationship

$$M_i = e^{A_i T} \approx \sqrt[n]{D(k+n)D(k)^{-R}} \quad (40)$$

defines the  $N_i(N_i - 1)/2$  constraints for fitting  $M_i$ . It is known (Brookes, 2005) that a general matrix has an  $n$ th root *iff* it is both Hermitian and positive semi-definite. This means, in our real-valued case, we need to show that the radicand in Eq. (40) is symmetric positive semi-definite. We proceed by first considering the construction of the right pseudo-inverse:  $C^{-R} = C^T(CC^T)^{-1}$ . Thus

$$\begin{aligned} M_i^n &\approx D(k+n)D(k)^{-R} \\ &= D(k+n)D(k)^T(D(k)D(k)^T)^{-1} \end{aligned} \quad (41)$$

and by substituting Eq. (38),

$$M_i^n \approx \begin{bmatrix} d(k+n) \\ \lambda d(k+n) \end{bmatrix} [d(k) \quad \lambda d(k)] \times \left( \begin{bmatrix} d(k) \\ \lambda d(k) \end{bmatrix} [d(k) \quad \lambda d(k)] \right)^{-1}$$

which expands to the trivially symmetric form

$$\begin{bmatrix} d(k+n)d(k) & \lambda d(k+n)d(k) \\ \lambda d(k+n)d(k) & \lambda^2 d(k+n)d(k) \end{bmatrix} \times \begin{bmatrix} d(k)^2 & \lambda d(k)^2 \\ \lambda d(k)^2 & \lambda^2 d(k)^2 \end{bmatrix}^{-1} \quad (42)$$

It is well known that the determinant of a matrix is equal to the product of its eigenvalues; likewise, the trace of a matrix provides their sum. To determine positive semi-definiteness, we examine the determinant of each matrix in Eq. (42), and find them to be zero; indicating an eigenvalue of zero. Looking at the traces, we find them non-zero; indicating that the other eigenvalue is non-zero. The trace of the inverted matrix in Eq. (42) is the sum of squared terms; given that our matrices are real, this indicates that the non-zero eigenvalue is positive and thus, the matrix is positive semi-definite. For the non-inverted matrix in Eq. (42) we note that a necessary condition for the trace to be negative is for  $d(k+n)$  to have a sign opposite that of  $d(k)$ ; thus to ensure positive semi-definiteness, one must only construct the radicand in Eq. (40) using values of  $d(k+n)$  and  $d(k)$  which have the same sign. This implies that the set of training data for a period, i.e.  $S_i$  for period  $i$ , must also be partitioned into a positive and negative set. The Least Squares fit would then be constructed by applying Eq.

(40) to both the negative and positive set. We note here that negative values of  $d(\cdot)$  are not meaningful given the typical scenario where  $d(k)$  represents the actual count of defects eliminated over the period  $[(k-1)T, kT]$  of an actual STP execution.

We also note here that the idea of partitioning the data by period may be applied to the first-order fit used to approximate the derivative term in  $D(\cdot)$ , where the  $\lambda$  in Eq. (38) would acquire a subscript of  $p(k)$  as per Eq. (24). This addition was included when generating the results using Cal2 in Section 9.

Once  $M_i$  is calculated for each period with data,  $x_{P_i} = x(P_i)$ —the state at the beginning of the training period, where  $P_i$  is the discrete time index marking the beginning of the training period—is estimated by Eq. (43) which is analogous to Eq. (33).

$$x_{P_i} = \begin{bmatrix} M_i - 1 \\ M_i^2 - m \\ \vdots \\ M_i^N - M_i^{N-1} \end{bmatrix}^{-L} \begin{bmatrix} D(P_i) \\ D(P_i + 1) \\ \vdots \\ D(P_i + n) \end{bmatrix} \quad (43)$$

Having calculated  $M_i$  and  $x_{P_i}$  for each completed training period, we apply Algorithm 3, where the non-parametric simulation referenced there is given by

$$x(P_i + k) = M_i^k x_{P_i}$$

and, for estimating the values of  $\zeta$  and  $\xi$ , one applies Eqs. (28)–(30) to  $M_i$ .

Note that lines 4–10 define an interpolation of the data from completed training periods adjusted to the estimate of  $x_{P_c}$ .

**Algorithm 3** (Simulation under the Cal2 parameter calibration algorithm).

```

1 Let  $D$  be an empty array
2 Let  $P = \{p_1 \dots p_N\}$  be an ordered set of training periods
3 Let  $p_c$  be the current training period
4 for  $i = 1$  to  $c$  do
5   Run a non-parametric simulation for  $p_i \rightarrow X_{P_i}$ 
6   Compute the differences between the entries of  $X_{P_i} \rightarrow D_{P_i}$ 
7   Concatenate the result  $D \leftarrow D.D_{P_i}$ 
8 end
9 Calculate  $\zeta$  and  $\xi$  and  $x_{P_c}$  for  $p_c$ 
10 Cumulatively subtract the elements of  $D$  from  $x_{P_c}$  in reverse order
    $\rightarrow \{x_0 \dots x_{P_{c-1}}\}$ 
11 Run a parametric simulation from  $x_{P_c}$  over the remaining prediction horizon
    $\rightarrow \{x_{P_c} \dots x_{End}\}$ 
12 Concatenate to form the entire simulation
    $X \leftarrow \{x_0 \dots x_{P_{c-1}}\} \cdot \{x_{P_c} \dots x_{End}\}$ 

```

### 11.2.2. Pre-fit data smoothing

In fitting the model parameters to the STP data, it is sometimes the case that the variation in the data causes the Least Squares fits to fail. In Cal1, one randomly excludes data points and re-attempts the fit. For Cal2, we employ data smoothing as an alternative. While both methods have consistently led to a successful fitting of the

parameters, the smoothing approach seems (philosophically) more likely to retain the information present in the training data.

We have experimented with applying a simple smoothing technique to the  $D(\cdot)$  and  $d(\cdot)$  data prior to running the Least Squares fits, and have observed a tendency for the fit to be impacted less heavily by variation as a result. The smoothing we have experimented with is given by

$$\hat{p}(i) = \frac{1}{\mu - m + 1} \sum_{j=\mu}^m p(j) \tag{44}$$

where  $p$  is the set of data items,  $\hat{p}$  is the smoothed version,  $\mu = \max(1, i - s)$ ,  $m = \min(N, i + s)$  for a smoothing factor  $s$  and number of data points  $N$ . The typical value of  $s$  in our experimentation is 3—this value was the minimum smoothing factor which produced a discernible, i.e. visual, continuity in a plot of the differential training data over time. Smoothing with a factor of 3 was employed in the example in Section 9 for both the first- and second-order data fitting.

### 11.3. Assessment of the technique

Here we present a brief comparative analysis of the performance of Cal1 and Cal2. The left half of Table 1 is composed of the normalized error between the estimate of  $r_0$

given an algorithm and a set of training data, and the total number of defects that were found during the STP.

From Table 1, we see that, given a small training set, the accuracy of the predictions of  $r_0$  generated by the different training algorithms, are comparable. For later re-calibration, say day 60 onwards, Cal2 seems to outperform Cal1 consistently.

The right half of Table 1 presents a similar result, though it examines the prediction of the duration between the project start and the Revenue Release date; in this table, the error is given relative to the duration from the project start to the time when the manufacturer marked the Revenue release. Thus one may see this comparison as a test of which technique most closely re-creates the results of the current state-of-practice in software release control. Again, but for a few initial anomalies, Cal2 consistently outperforms Cal1 regardless of the inclusion of parameter correction in Cal1.

## 12. Conclusions and future work

The interaction between the learning system for parameter correction and the real-time parameter and goal updates possible with model predictive control provide an elegant framework for future exploration in software cybernetic systems.

Table 1  
Relative error in  $r_0$  and revenue release (RR) date estimates by variation in training set size and choice of calibration algorithm

Training days	% Relative error, $r_0$			% Relative error, RR date		
	Cal1		Cal2	Cal1		Cal2
	Corrected	Uncorrected		Corrected	Uncorrected	
30	24.82	24.83	59.51	0.00	0.00	10.11
35	26.50	12.99	7.42	34.83	5.62	8.99
40	14.04	0.53	16.33	22.47	11.24	20.22
45	23.86	12.19	10.57	42.70	31.46	23.60
50	30.83	8.92	20.28	65.17	31.46	26.97
55	14.14	11.53	14.33	41.57	42.70	16.85
60	12.89	12.21	11.17	40.45	43.82	11.24
65	15.01	13.61	8.90	48.31	50.56	6.74
70	11.09	12.35	3.99	39.33	47.19	20.22
75	10.04	11.20	4.27	35.96	42.70	20.22
80	11.44	11.32	1.31	41.57	46.07	13.48
85	9.66	10.49	0.92	38.20	44.94	8.99
90	9.25	9.87	4.53	41.57	47.19	0.00
95	7.27	8.57	3.45	38.20	46.07	3.37
100	6.33	7.45	3.74	38.20	46.07	2.25
105	5.22	6.34	1.47	33.71	41.57	7.87
110	3.11	4.72	0.54	26.97	37.08	10.11
115	1.95	3.33	0.49	22.47	32.58	10.11
	All data points			All data points		
Mean	13.19	10.14	9.62	36.20	36.02	12.30
Median	11.27	10.84	4.40	38.20	42.70	10.11
STDEV	8.27	5.13	13.75	13.14	15.18	7.57
	60 days of training and beyond			60 days of training and beyond		
Mean	8.61	9.29	3.73	37.08	43.82	9.55
Median	9.45	10.18	3.60	38.20	45.51	9.55
STDEV	3.93	3.24	3.34	6.84	4.89	6.33

The example presented in Section 9 shows that the controller is capable of determining the appropriate changes required to drive the industrially validated CDM model to the specified desired behavior, despite the variation in the consecutive estimates of  $r_0$ ,  $\zeta$ , and  $\xi$ . The demonstrated predictive accuracy of the CDM model documented in Cangussu et al. (2002, 2004) implies that the control suggestions would have had a similar effect on the actual STP execution had they been applied there. While the example assumed that there were up to 5 trained, experienced, employees available to be brought onto the project, the technique allows for other scenarios including latency in human resource acquisition.

The parameter identification algorithm Cal2 exhibits a higher degree of predictive accuracy than Cal1 in the later stages of the STP with respect to the prediction of  $r_0$  and the Revenue Release date. We note that the later stages are likely to be the times when the management turns to tool support for aid in mitigating a slipping schedule. Thus we see Cal2 as an improved parameter calibration algorithm.

Our future work is directed at the application of this technique to more complex Software Engineering models which are currently under development.

All calculations for applying this technique are implemented as MATLAB scripts.

## Acknowledgements

The authors would like to thank Richard Karcich for access to the data set used in Cangussu et al. (2004).

## References

- Abdel-Hamid, T., Madnick, S.E., 1989. Lessons learned from modeling the dynamics of software development. *Communications of the ACM* 32 (12), 1426–1455.
- Abdel-Hamid, T., Madnick, S.E., 1991. *Software Project Dynamics*. Prentice Hall.
- Andersson, C., Karlsson, L., 2001. A system dynamics simulation study of a software development process. Master's thesis, Lund Institute of Technology, Lund University.
- Andersson, C., Karlsson, L., Nedstam, J., Host, M., Nilsson, B., 2002. Understanding software processes through system dynamics simulation: a case study. In: *Proceedings of the Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pp. 41–48.
- Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E., 2000. The explicit solution of model predictive control via multiparametric quadratic programming. In: *American Control Conference, Chicago, USA*, pp. 872–876.
- Boehm, B.W. et al., 2000. *Software Cost Estimation with Cocomo II*. Prentice Hall.
- Borrelli, F., 2002. Discrete time constrained optimal control. Ph.D. thesis, Swiss Federal Institute of Technology (ETH), Zurich.
- Borrelli, F., 2003. Multiparametric programming: a geometric approach. *Lecture Notes in Control and Information Sciences* 290 (January), 3–50.
- Box, G., Tiao, G., 1992. *Bayesian Inference in Statistical Analysis*. John Wiley & Sons.
- Brookes, M., 2005. The matrix reference manual. URL <<http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/intro.html>>.
- Cai, K.-Y., 2001. Optimal test profile in the context of software cybernetics. In: *Proceedings of the Second Asia-Pacific Conference on Quality Software*, pp. 157–166.
- Cai, K.-Y., 2002. Optimal software testing and adaptive software testing in the context of software cybernetics. *Information and Software Technology* 44, 841–855.
- Cai, K., Cangussu, J., DeCarlo, R., Mathur, A., 2003. An overview of software cybernetics. In: *Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice*, pp. 77–86.
- Cai, K.-Y., Jing, T., Bai, C.-G., 2005. Partition testing with dynamic partitioning. In: *Proceedings of the 29th Annual International Computer Software and Applications Conference*, vol. 2, pp. 113–116.
- Camacho, E.F., Bordons, C., 2004. *Model Predictive Control*. Springer Publication.
- Cangussu, J.W., DeCarlo, R.A., Mathur, A.P., 2002. A formal model of the software test process. *IEEE Transactions on Software Engineering* 28 (8), 782–796.
- Cangussu, J.W., DeCarlo, R.A., Mathur, A.P., 2003. Using sensitivity analysis to validate a state variable model of the software test process. *IEEE Transactions on Software Engineering* 29 (5), 430–443.
- Cangussu, J.W., Karcich, R.M., Mathur, A.P., DeCarlo, R.A., 2004. Software release control using defect based quality estimation. In: *15th International Symposium on Software Reliability Engineering*, pp. 440–450.
- Card, D., 1994. Statistical process control for software? *IEEE Software* 11 (May), 95–97.
- Cass, A.G., Lerner, B.S., Sutton Jr., S.M., McCall, E.K., Wise, A., Osterweil, L.J., 2000. Little-jil/juliette: a process definition language and interpreter. In: *Proceedings of the 22nd International Conference on Software Engineering*, pp. 754–757.
- Chen, C.-T., 1999. *Linear System Theory and Design*. Oxford University Press.
- Coleman, T., Branch, M.A., Grace, A., 1999. *Optimization Toolbox User's Guide*. The Math Works, Inc.
- Dalal, S., Horgan, J., Kettenring, J., 1993. Reliable software and communication: software quality, reliability, and safety. In: *Proceedings of the 15th International Conference on Software Engineering*, pp. 425–435.
- DeCarlo, R.A., 1989. *Linear Systems*. Prentice Hall.
- Dutka, A., Grimble, M.J., 2004. State-dependent riccati equation control with predicted trajectory. In: *American Control Conference*, vol. 2, pp. 1563–1568.
- Goodwin, G.C., Seron, M.M., De Doná, J.A., 2005. *Constrained Control and Estimation, Communications and Control Engineering*, vol. XVIII. Springer.
- Horgan, J., Mathur, A., 1996. *Software Testing and Reliability*. McGraw-Hill.
- Iida, H., Eijima, J., Yabe, S., Matsumoto, K., Torii, K., 1996. Simulation model of overlapping development process based on progress of activities. In: *Software Engineering Conference, 1996. Proceedings*, pp. 131–138.
- Jalote, P., Saxena, A., 2002. Optimum control limits for employing statistical process control in software process. *IEEE Transactions on Software Engineering* 28 (12), 1126–1134.
- Kellner, M., Madachy, R., Raffo, D., 1999. Software process simulation modelling, why? what? how? *Journal of Systems and Software* 46, 91–105.
- Kerrigan, E.C., Maciejowski, J.M., 2000. Soft constraints and exact penalty functions in model predictive control. In: *Proceedings of UKACC International Conference (Control 2000)*, Cambridge, UK.
- Lotka, A.J., 1925. *Elements of Physical Biology*. Williams & Wilkins Co.
- Luenberger, D., 1979. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley and Sons.
- Martin, R., Raffo, D., 2000a. Application of a hybrid process simulation model to a software development project. In: *Proceedings of PROSIM*, pp. 237–246.

- Martin, R., Raffo, D., 2000b. A model of the software development process using both continuous and discrete models. *International Journal of Software Process Improvement and Practice* 5 (2/3), 147–157.
- McClure, D.E., 1975. Nonlinear segmented function approximation and analysis of line patterns. *Applied Mathematics* 33, 1–37.
- Padberg, F., 2005. On the potential of process simulation in software project schedule optimization. In: *Proceedings of the 29th Annual International Computer Software and Applications Conference*, vol. 2, pp. 127–130.
- Scacchi, W., 2002. Process models in software engineering. In: Marciniak, J.J. (Ed.), *Encyclopedia of Software Engineering*, second ed. John Wiley and Sons Inc., New York.
- Volterra, V., 1926. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Atti della Roma Accademia nazionale dei Lincei. Memorie della classe di scienze fisiche, matematiche e naturali* 6 (2), 31–133.

**Scott D. Miller** received a B.S. in Computer Science from Purdue University in 2000. He is currently pursuing a Ph.D. under the direction of R.A. DeCarlo and A.P. Mathur at Purdue. His research is focused on modeling and control of software processes using state models and feedback control, and he is a student member of the IEEE.

**Raymond A. DeCarlo** received a B.S. and M.S. in Electrical Engineering from the University of Notre Dame in 1972 and 1974, and received his Ph.D. from Texas Tech University in 1976 for research centered on Nyquist Stability Theory with applications to multidimensional digital filters. He is currently a full Professor in the Electrical Engineering department at Purdue University. He has worked at the General Motors Research Laboratories, is a Fellow of the IEEE, and former Associate Editor for both *Technical Notes and Correspondence*, and *Survey and Tutorial Papers* for the *IEEE Transactions on Automatic Control*. He has

been secretary-administrator of the Control Systems Society, a member of the Board of Governors of the Society, and both Program Chairman and General Chairman for the IEEE CDC. He is a former VP for Financial Activities, and distinguished member, of the IEEE Control Systems Society, and is a holder of the IEEE Third Millennium Medal. He has written three books and has numerous journal articles, conference articles, and contributed book chapters. His research interests include modeling of computer processes, variable structure control of linear systems, non-linear systems and decentralized systems, biomedical modeling and control, diesel engine control, hybrid electric vehicle control supervisory control, hybrid and discrete event systems, numerical linear algebra as applied to control and stability problems, decentralized control of large scale systems, analog and analog-digital fault diagnosis of circuits and systems, and digital and active filter design.

**Aditya P. Mathur** received his B.S., M.S., and Ph.D. degrees in 1970, 1972, and 1977, respectively, from the Birla Institute of Technology and Science, Pilani, India. He is currently a Professor of Computer Science and Associate Dean for Graduate Education and International Programs at Purdue University. His research interests are in software testing and reliability, feedback control of software processes, and music composition. He has published over 100 research papers, written two books, and composed over 40 pieces in a variety of genres.

**João W. Cangussu** received a B.S. degree in Computer Science from the Federal University of Mato Grosso do Sul-Brazil in 1990 and a M.S. in Computer Science from the University of São Paulo at São Carlos-Brazil in 1993. He received a Ph.D. degree in Computer Science from Purdue University in 2002. He is currently an Assistant Professor in the Department of Computer Sciences at the University of Texas at Dallas. His research interests are software process modeling and control, software testing, and adaptive systems. He is a member of the IEEE and the ACM.