Feedback Control of the Software Test Process Through Measurements of Software Reliability

João W. Cangussu*
Aditya P. Mathur†
Department of Computer Sciences
Purdue University
West Lafayette-IN 47907-1398, USA
{cangussu,apm}@cs.purdue.edu

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

Abstract

A closed-loop feedback control model of the Software Test Process (STP) is described. The model is grounded in the well established theory of Automatic Control. It offers a formal and novel procedure for using product reliability or failure intensity as a basis for closed loop control of the STP. The reliability or the failure intensity of the product is compared against the desired reliability at each checkpoint and the difference fed back to a controller. The controller uses this difference to compute changes necessary in the process parameters to meet the reliability or failure intensity objective at the terminal checkpoint (the deadline). The STP continues beyond a checkpoint with a revised set of parameters. This procedure is repeated at each checkpoint until the termination of the STP. The procedure accounts for the possibility of changes, during testing, in reliability or failure intensity objective, the checkpoints, and the parameters that characterize the STP. The effectiveness of this procedure was studied using commercial data available in the public domain and also from the data generated through simulation. In all cases the use of feedback control produces adequate results allowing the achievement of the objectives.

1 Introduction

Software reliability is a well known metric of software quality [7]. Reliability $R(t)$ of a software product is the probability of its failure free operation during the time interval $[0 - t)$ for a given operational profile. [13] Failure intensity of software is the number of failures experienced per unit time of execution and is denoted by $\lambda(t)$. A variety of models have been proposed to estimate $R(t)$ using estimates of $\lambda(t)$ that are derived from failure data obtained during a test of the software product. The test is conducted using inputs generated in accordance with an operational profile which is a probability distribution on the input space of the software product. [13]

Proposed here is a novel approach, based on feedback control, that allows a software test manager to make use of in-test reliability or failure intensity measurements to control the progress of the test so as to meet the quality objective by the deadline. The quality objective is specified as a reliability metric at some time prior to the start of the test process and is to be met by a deadline. Measurements of reliability or failure intensity at checkpoints during the process are compared with those estimated using the proposed model and the difference is fed back to a controller. The controller then computes a finite set of possible changes that the test manager could select from to alter the test process so as to possibly meet the quality objective by the deadline. Two key parameters that the controller allows to be manipulated are the size of the workforce dedicated to the test process and the quality of the test process.

The strength and novelty of our approach lies in the (a) use of a controller in the feedback loop that is based on well proven techniques of feedback control in various domains of engineering and (b) the flexibility allowed to the test manager in that the suggested changes may or may not be ignored and the quality objectives revised in-process. Feedback allows the model to adjust itself and learn about the process over time. The ability to ignore recommendations from the model allows the test manager to postpone alterations to the process due to one or more of a variety of reasons such as budgetary constraints, market pressure, etc. Even in the case where recommendations from the model are ignored throughout the test process, the model is useful in that it predicts when the quality objective will likely be met if no change is made to the process. In a case study

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

conducted using a similar model, indeed the test manager ignored all recommendations only to find that the project was completed 12 weeks later than the date predicted by the test manager and only 2 weeks later than that predicted by the model. [3]

The remainder of this paper is organized as follows. Section 2 describes `Model S` and its application to the control of the STP. The method used to evaluate the model is described in Section 3. Results from the evaluation exercises and their analyses appear in Sections 4 and 5 respectively. A discussion of this work related to its applicability in practice is in Section 6.

2 A State Model of the STP

We first review the context in which feedback control is being used and then describe a state variable model of the STP. The mathematical background required for a complete understanding of the material in this section is found in standard texts on automatic control [8, 12]. The estimation of $R(t)$ using the failure intensity $\lambda(t)$, is discussed in the literature [13, 17]. The proposed control procedure does not favor or recommend any specific model for the estimation of reliability. In the remainder of this paper we refer to the proposed model as `Model S`.

2.1 Feedback Control of the STP

Consider the system test phase of a software product. We assume that at the start of this phase a Test Manager is given a quality objective for the end product and the deadline by which this objective must be met. The quality of a software product can be specified using one or more of quality metrics two of which being its reliability and the number of remaining errors. In `Model S` we use failure intensity as the quality metric since reliability can be directly derived from it.

We further assume that to plan and monitor the progress of the STP, the test manager divides the entire phase into a sequence of $n > 0$ checkpoints denoted by $\text{cp}_1, \text{cp}_2, \ldots, \text{cp}_n$ with cp_1 being the first checkpoint after testing has begun and cp_n the deadline. The realization of the ultimate quality objective is distributed over these checkpoints. Thus, checkpoint cp_i is specified as a pair (t_i, λ_i^d), where t_i is some time prior to the deadline and λ_i^d is the desired failure intensity of the product at checkpoint cp_i.

The use of feedback control during the STP is illustrated in Figure 1. The figure shows four key components that participate in the control process. These are the `Actual STP` which consists of the test engineers, tools, documentation, etc., a `State Model` of the STP which is `Model S`,

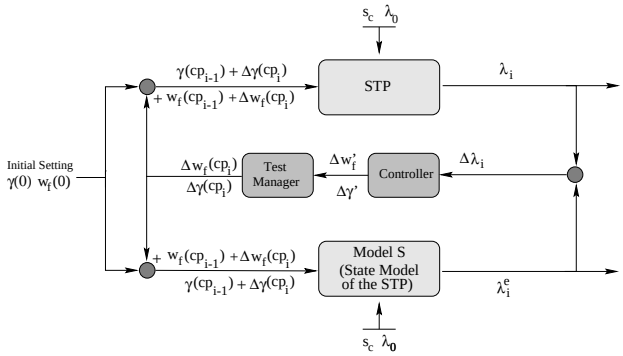

Figure 1. Closed loop control of the software test process using reliability measurements. $\lambda_i(t)$ is the estimate of the failure intensity at checkpoint cp_i. $\lambda_i^e(t)$ is the expected failure intensity of the product computed by `Model S` at checkpoint cp_i.

a `Controller`, and the `Test Manager`. The management decides how many testers to employ to test the product. This number is the initial value of w_f. The Test Manager estimates the quality γ of the test process. The complexity of the software under test is computed as a convex combination of several well known quality metrics such as the number of function points, lines of code, and the number of data flows [3]. An initial estimate of λ_0 is made of the software product ready to enter the test phase.

At checkpoint $\text{cp}_i, i > 0$, the failure intensity λ_i of the software product is estimated and compared against the expected value λ_i^e computed by the State Model. The error signal $\Delta\lambda_i = \lambda_i^e - \lambda_i$ is input to a controller. Using this error signal, s_c, w_f, and γ, the controller computes a set of possible changes $\Delta w_f'$ and $\Delta\gamma'$ that could be made to w_f and γ, respectively, in order for the STP to meet the quality objective on or before the deadline. The computed changes are made available to the Test Manager who may or may not choose to ignore them. Though Figure 1 gives the impression that only a single pair ($\Delta w_f'$ and $\Delta\gamma'$) of values is output by the controller, in reality the controller outputs a finite set of such pairs from which the Test Manager could select. The STP resumes with a workforce of $(w_f + \Delta w_f)$ and a process quality of $(\gamma + \Delta\gamma)$, where Δw_f and $\Delta\gamma$ denote the actual changes made by the Test Manager.

The STP model and the Controller work together to form a feedback control loop in the STP. Unforeseen disturbances in the STP are accounted for by updating estimates of the model parameters. The control loop offers the Test Manager opportunities to make alterations to the STP at any checkpoint. These alterations could come in many forms such as a change in the number of testers, or in the quality of the test process, or in the test objectives themselves, or

any combination of these. Thus, the inherent uncertainty and the variability of the STP is well accounted for in the feedback control loop.

2.2 The State Model

A linear model of the STP is based on three assumptions with consequent equations presented below [2]. These assumptions are based on an analogy of the STP with a physical process typified by a spring-mass-dashpot system. The widespread use of differential equations to model so many different types of systems [6, 12] combined with the fact that most of such models were developed using analogies to physical systems with assumptions similar [9, 16] to ours further justify the choice of a second order linear model.

Assumption 1: The magnitude of the rate of decrease of failure intensity of a software product is proportional to the net applied effort during the test phase and inversely proportional to the complexity of the product,

$$\ddot{\lambda} = \frac{e_n}{s_c} \quad \Rightarrow \quad e_n = \ddot{\lambda}(t)\, s_c \tag{1}$$

The first assumption is justified as follows. When the same metric or combination of metrics is used to compute software complexity for two different programs under test, it is reasonable to expect that more effort will be necessary to test the more complex program and consequently decrease failure intensity (λ).

The net applied effort (e_n) is the balance of all the effort applied during the test phase. This results from the difference of the effective effort applied by the test team minus any "frictional" forces that decrease the applied effort. Since λ represents failure intensity, its first derivative $\dot{\lambda}$ is the failure intensity reduction velocity (v_e). Consequently, $\ddot{\lambda}$, which denotes the rate of change of $\dot{\lambda}$, is an acceleration. Thus, the concepts of velocity and acceleration have counterparts in the test phase.

Assumption 2: The magnitude of the effective test effort is proportional to the product of applied work force and the failure intensity, i.e., for an appropriated ζ,

$$e_{et} = \zeta\, w_f\, \lambda(t) \tag{2}$$

Assumption 2 can be understood with another analogy. In a spring the restoring force is determined by the spring stiffness and by how much the spring is extended beyond its natural length. Increasing the spring stiffness or the extension increases the restoring force. The effective test effort can be interpreted in an analogous way. The failure intensity is analogous to the spring length. At the beginning of the test phase λ is larger than it is towards the end. Hence, the effective effort decreases as λ decreases. The work force can be related to the spring stiffness. The larger the work force, the greater the restoring force, i.e., the effective effort. Thus spring stiffness is analogous to w_f and spring extension to failure intensity (λ). In Eqn. 2, ζ remains constant over a period and must be calibrated for the project under analysis.

Assumption 3: The resistance to a decrease in the failure intensity is proportional to the the velocity of failure intensity and inversely proportional to the overall quality of the test phase, for an appropriate constant ξ,

$$e_r = \xi\, \frac{1}{\gamma}\, \dot{\lambda}(t) \tag{3}$$

This assumption implies that the faster one tries to reduce the failure intensity the more likely one is to make mistakes which slows the entire process. A physical dashpot can be used to explain this behavior. The coefficient of viscosity of the liquid inside the dashpot is $\frac{1}{\gamma}$. Therefore, a small coefficient of viscosity is analogous to the test phase being conducted in a smooth and careful way and, thus, the number of new errors inserted is small. Larger coefficient of viscosity is analogous to the test phase in which more errors are introduced than would be introduced under normal circumstances. The velocity component in the dashpot is analogous to the failure intensity reduction velocity ($\dot{\lambda}$). Thus, the overall quality of the test phase, denoted by (γ), and the rate at which failure intensity is decreasing, determines the failure intensity reduction resistance effort which is analogous to the damping force generated by the dashpot. In Eqn. 3, ξ is merely a constant of proportionality.

Combining Eqs. 1, 2, and 3 and organizing in a State Variable format ($\dot{x} = Ax + Bu$) produces the following system of equations.

$$\begin{bmatrix} \dot{\lambda}(t) \\ \ddot{\lambda}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-\zeta\, w_f}{s_c} & \frac{-\xi}{\gamma\, s_c} \end{bmatrix} \begin{bmatrix} \lambda(t) \\ \dot{\lambda}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{s_c} \end{bmatrix} F_d \tag{4}$$

$$\begin{bmatrix} \lambda(t) \\ \dot{\lambda}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda(t) \\ \dot{\lambda}(t) \end{bmatrix} \tag{5}$$

`Model S` shown in Figure 1 consists of Eqs. 4 and 5. A solution to these equations, with appropriate estimates of parameter values, generates $\lambda_i^e(t)$ in Figure 1. Using the relationship in Eq. 6, one computes $R_i^e(t)$.

$$R(\tau) = e^{-\lambda(t)\tau} \tag{6}$$

2.3 Estimation of model parameters

w_f is relatively easy to compute as it is defined to be the number of testers testing the product. The value of w_f must be adjusted for any part-time and temporary personnel. Parameters s_c and γ are computed by applying a convex combination [10] of available metrics. The remaining

parameters are estimated through the use of System Identification [11] techniques [5]. An important characteristic of our approach is that estimates of all parameters are updated at each checkpoint thereby improving their accuracy with the passage of time. Changes in the test environment, such as in the workforce and the quality of the STP, are accounted for as and when they occur.[3]

2.4 Computing $\Delta w'_f$ and $\Delta\gamma'$

In a feedback control system, the largest eigenvalue of the system determines the slowest rate of convergence and dominates how fast the output variables converge to their desired values. Therefore, in order to control $\lambda(t)$ so that it reaches its desired value by the deadline t_n, we must adjust the largest eigenvalue appropriately. Given $\lambda(T)$, the failure intensity at time T, and $\lambda(T + \Delta t)$, the desired failure intensity at time Δt later, we use the following equation to determine the amount by which to adjust the largest eigenvalue ς_{max}.[1]

$$\lambda(T + \Delta t) \quad = \quad \lambda(T)\, e^{-\varsigma_{max}\, \Delta t} \tag{7}$$

We know the values of $\lambda(T)$, $\lambda(T + \Delta t)$ and Δt at checkpoint $\text{cp}_i, i > 0$ and hence we solve Eq. 7 and find the value of ς_{max}. The eigenvalues of a system are defined by the roots of the characteristic polynomial ($\Pi_A(\varsigma) = det[\varsigma I - A]$). Computing the characteristic polynomial of our model leads to

$$\begin{aligned} det[\varsigma I - A] &= det \begin{bmatrix} \varsigma & -1 \\ \frac{\zeta\, \hat{w_f}}{s_c} & \varsigma + \frac{\xi}{\hat{\gamma}\, s_c} \end{bmatrix} \\ &= \varsigma^2 + \frac{\xi}{\hat{\gamma} s_c}\varsigma + \frac{\zeta \hat{w_f}}{s_c} \end{aligned} \tag{8}$$

where $\hat{\gamma} = \gamma + \Delta_\gamma$ and $\hat{w_f} = w_f + \Delta_{w_f}$. Using Eq. 8 we compute the variations in the work force and in the quality of the process necessary to meet the desired quality objective by the deadline.

3 Evaluation of `Model S`

Two studies were undertaken to evaluate the performance of `Model S`. One study, referred to as `Case Study I`, is based on data available in the public domain. The other study was conducted using simulation. A sensitivity analysis of our model [4] based on tensor product [1] indicates its closeness to reality and sensitive to changes in parameters.

[1]The symbol λ is used in math literature to represent the eigenvalues of a system. λ is also used in the Software Reliability to represent failure intensity. To avoid confusion and keep uniformity with Software Reliability standards we decided, in this paper, to represent the eigenvalues of system by the symbol ς.

3.1 `Case Study I`

In one study, hereafter referred to as `Case Study I`, the failure data was obtained from the Data & Analysis Center for Software (DACS), Software Reliability Dataset, System Code 1.[14] The project is a real time command and control application with 21,700 delivered object code instructions, as described in DACS web-page. A total of 136 failures were found during the system test operations phase. The check points for this case study were assumed to occur at regular intervals of 10 days during system test. The observed failure intensity of the product at cp_i is computed as the average failure intensity for the period between cp_{i-1} and cp_i. An execution time model is used in this case, but it is not a requirement. The expected failure intensity is generated by `Model S` with $s_c = 21.7$, $w_f = 3$, $\gamma = 0.6$, $\zeta = 1.3$ and $\xi = 55.79$. The value for s_c represents the number of KDOCI (Kilo Delivered Object Code Instructions) since it is the only complexity metric directly derived from the information available about the project. The values for w_f and γ were set arbitrarily due to lack of data. This does not pose a problem when the alterations in w_f suggested by the feedback controller are considered as percentages. Thus, for example, an increase of 1.5 in w_f represents a 50% increase in the actual work force. A similar argument applies to the quality of the process (γ). The expected behavior was intentionally set to have a faster decrease in failure intensity than the actual project hence justifying the values chosen for ζ and ξ.

3.2 `Case Study II`

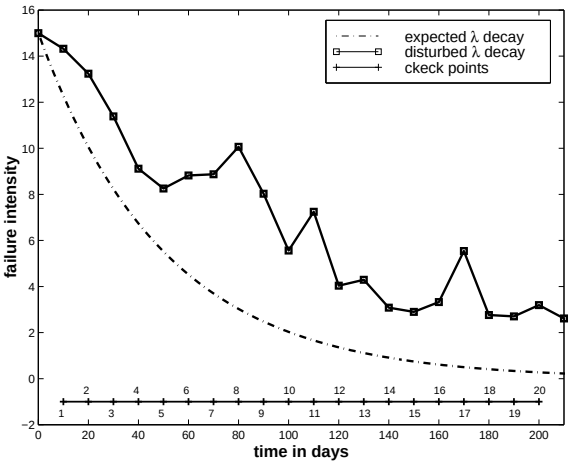

Figure 2. Expected and disturbed decay of failure intensity (λ) at each check point for a specific software test process.

In the second study simulation was used to generate data to further evaluate `Model S`. First, the expected decay of the failure intensity is computed at each check point (cp_i) using the equation

$$\lambda_{\text{cp}_i} = \lambda_0 e^{-\theta_e \text{cp}_i} \quad (9)$$

where λ_0 is the initial failure intensity and θ_e the expected failure intensity decay. Solving Eq. 9 at each check point leads to the expected failure intensity for a specific project. The corresponding reliability is computed by $R_i(\tau) = e^{-\lambda_i \tau}$ [13]. An example of the expected failure intensity decay and the corresponding reliability can be observed in Figures 2 and 3. The unit of time in both figures is days. The use of `Model S` with the values of the parameters set to $s_c = 5$, $w_f = 3$, $\gamma = 0.6$, $\zeta = 0.7$ and $\xi = 63$ produces results similar to those observed in Figure 2. As described below, Eq. 9 was used to simplify the introduction of disturbance into the simulated test process. Similar results can be obtained from `Model S` by producing a random input(F_d).

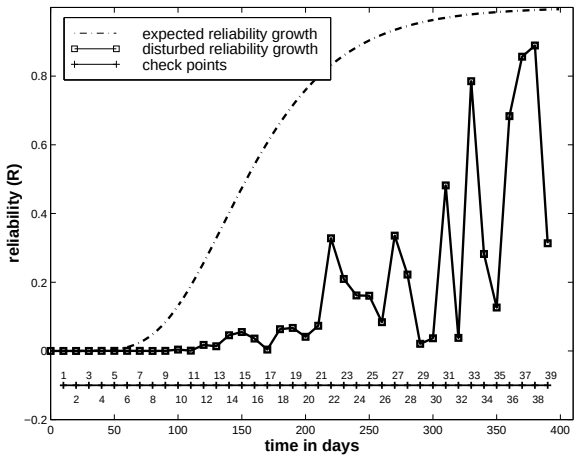

Figure 3. Expected and disturbed reliability (R) growth at each check point for a specific software test process.

The expected values generated as above are then perturbed to produce deviations from the expected behavior. The deviations are produced by changes in the expected failure intensity decay parameter. If θ_e is the expected decay, the new values for the disturbed failure intensity decay parameter are generated by $\theta_d(\text{cp}_i) = \alpha(\text{cp}_i)(\theta_e\ \vartheta)$, where $\alpha(\text{cp}_i) = 0.8, \ldots, 1.2$ and ϑ is a constant between $0.6\ and\ 0.8$. The results of such perturbation in the failure intensity decay and the corresponding reliability are shown in Figures 2 and 3. We assume a negative perturbation, i.e., a delay in the test process. A less likely positive perturbation can also be observed and was not investigated. We believe that from a control point of view a delay in the test process is a more useful case to be handle than a speed up in the process.

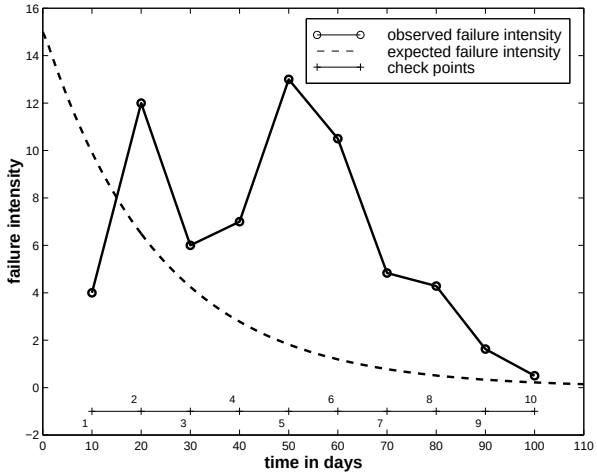

Figure 4. Observed failure intensity for `Case Study I` and the results obtained from the application of `Model S`.

After the detection of a deviation from the expected behavior, `Model S` is used to calculate possible changes in w_f and γ to correct such deviations. The test manager must decide on the number of check points (Δt in Eq. 7) that can be tolerated for the process to converge to the expected behavior and then feedback can be used as described in Section 2.4. A test manager has two alternatives to select from:

1. Ignore the recommendations of the controller and wait until the next check point to re-evaluate the process. This is a feasible situation when the deviation from the expected behavior is not large; when the accuracy of the collected data is not within safe limits; or when budget constraints prevent the implementation of the recommendations.

2. Implement the recommendations partially or fully. If the test manger decides to fully implement recommendations of the controller, whether or not there will be a delay in the application of the solutions recommended by the controller. For example, suppose the controller finds that the size of the test team should be increase by 2. In the absence of any delay, i.e. two testers are available for work during the time following the current checkpoint, feedback can be applied directly. If there is a delay, such as when the Human Resource Department needs two weeks to hire a tester, `Model S` is used to predict the behavior for the next two weeks when the controller re-evaluates the process to provide

a set of possible solutions. The same considerations are valid if the test manager decides to apply partially the recommendations. The difference is that an extension of the deadline is predicted by the model.

The simulation results presented in Section 4 account for all the alternatives presented above. Though variation in parameters is also present in the simulation runs, in this paper we restrict ourselves to the discussion of simulation results using the parameter values presented earlier in this section.

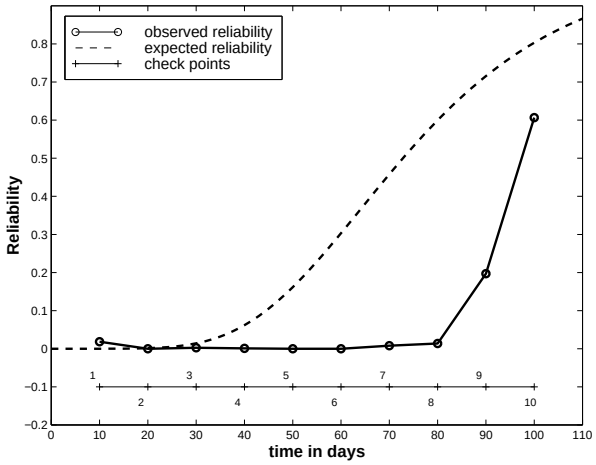

Figure 5. Observed reliability for `Case Study I` and the results obtained from the application of `Model S`.

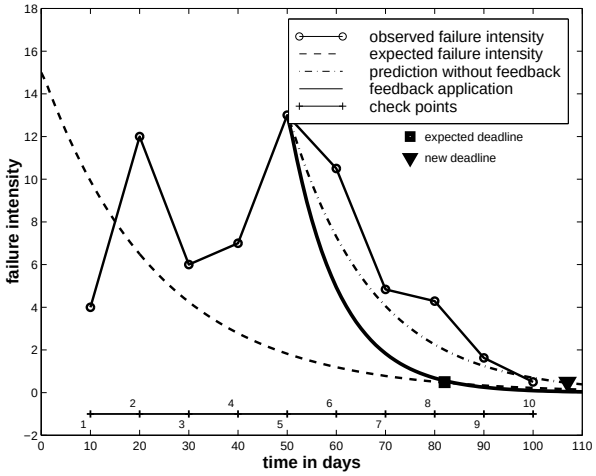

Figure 6. Prediction of `Model S` with no changes in the software test process and the result of the feedback application for the software test process for `Case Study I`.

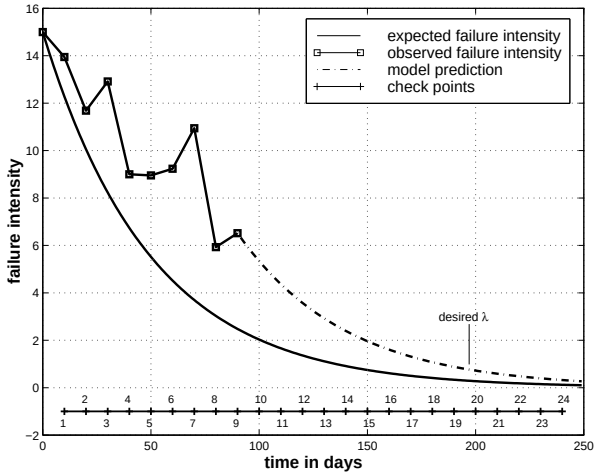

Figure 7. Prediction from `Model S` for the decrease of failure intensity in a disturbed test process without the application of feedback.

4 Results

4.1 Results from `Case Study I`

Figure 4 depicts the expected and observed failure intensities for `Case Study I` and Figure 5 does the same for the reliability values. As can be noticed, the observed values diverge from the expected ones. Though deviations are observed early in the process, we assume that the test manager has decided to ignore them until check point 5 (i.e. 50 days from the start of the test process). Figure 6 shows the impact of no changes in the process and the revised expected deadline. Figure 6 also shows the impact of feedback to achieve the desired failure intensity on or before the deadline.

4.2 Simulation Results

A total of two hundred simulation runs were carried out. In this section we present the results of one simulation run to exemplify the behavior of `Model S`.

Though deviations from the expected behavior are observed from the beginning of the process, in the simulation runs, actions are taken only at check point 9, i.e. $cp_9 = 90$ days from the start of the project. It was decided to postpone the application of the model until cp_9 to make it more interesting. There is no time restriction to the application of `Model S` but the process is more sensitive to changes at initial periods as described in a sensitivity analysis [4]. Therefore, the choice of postponing the use of the model does not bias the results in any way.

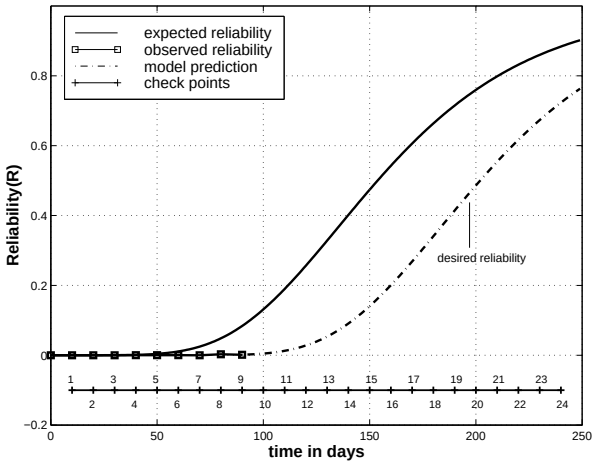

Figure 8. Prediction from `Model S` for reliability growth in a disturbed test process without the application of feedback.

Simulation Case I

The results of applying `Model S` to a disturbed test process are presented in Figures 7 and 8. In this case, it is assumed that the deviation in the test process is observed but no action is taken by the test manager. The model is used to predict the new deadline to meet the failure intensity requirements. It can be observed that it will take 47 extra days to reach the desired λ with no changes in the process.

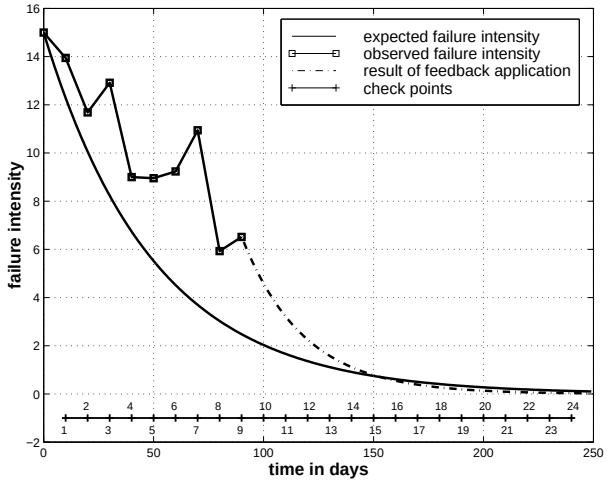

Figure 9. Result in the decrease of failure intensity from the immediate application of feedback control in a disturbed test process.

Simulation Case II

For this case, we assume that the deviation in the test process is observed and action taken immediately. This often implies that resources are promptly available to increase the size of the test team or to improve the quality of the test process. The eigenvalue of the system must set to -0.0357 in order to correct the deviation at cp_9 and arrive at the desired result within 60 days ($\Delta t = 60$). The feedback alternative provided by the model to achieve this goal, by increasing the work force, is shown in Figures 11. The model also provides all the possible combinations of increasing w_f, γ and altering the deadline. However, a discussion of these alternatives is beyond the scope of this paper. The result of properly increasing the work force (Δw_f) or increasing the overall quality of the test process ($\Delta\gamma$) or any combination of them are shown in Figures 9 and 10.

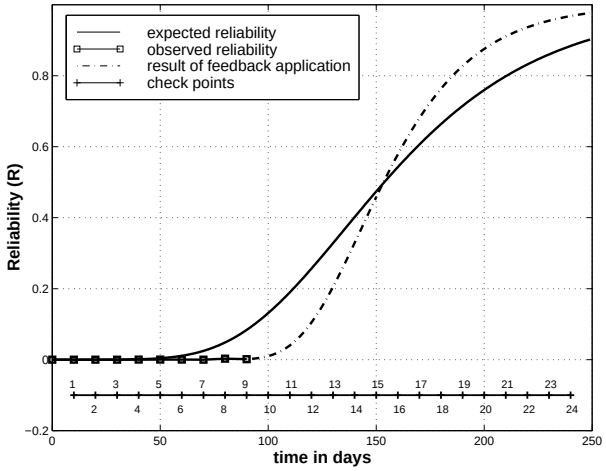

Figure 10. Result in the reliability growth from the immediate application of feedback control in a disturbed test process.

Simulation Case III

The results of simulation case III can be observed in Figures 12 and 13. In this case, the deviation is observed at check point 9 (cp_9) but resources are not promptly available and a delay in improving the process performance is expected. A three week delay is assumed. The delay can be due to the hiring process in case the test manager choose to increase the size of the test team or due to the acquisition of a better testing tool and the training process associate with it. Under such conditions, `Model S` is used to predict the expected behavior for the subsequent three weeks and then feedback is applied to compute the alternatives to obtain the desired results prior to the deadline. To achieve this goal, the eigenvalue of the system needs to be -0.0415. A

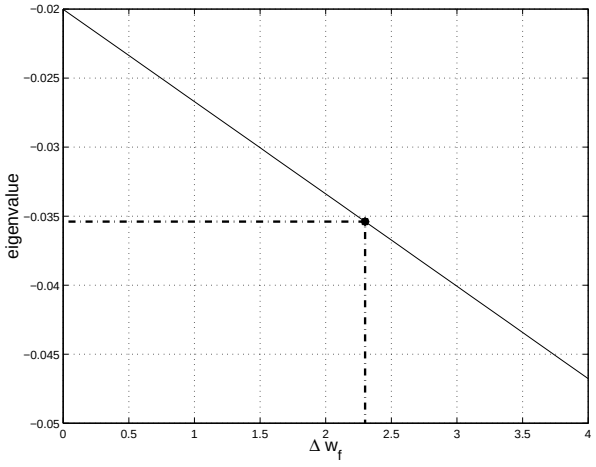

Figure 11. Eigenvalue relationship for changes in Δw_f for simulation case II.

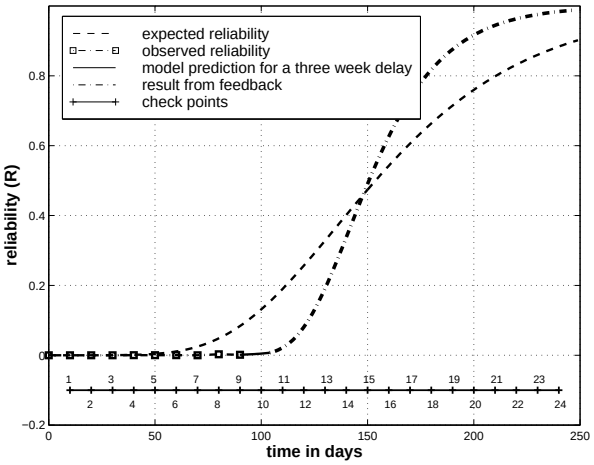

Figure 13. Result in the reliability growth from the application of feedback control in a disturbed test process after a three week delay.

increase of 3.2 in the work force (Δw_f) is associated with this eigenvalue. The quality of the test process ($\Delta\gamma$) can not be increase as much as necessary to place the eigenvalue of the system in the desired position. Therefore, case a 3.2 increase in the work force is not available, a combined solution of increasing both w_f and γ should be applied. Again, `Model S` provides all possible alternatives for changes in Δw_f and $\Delta\gamma$, but an analysis of these results is beyond the scope of this paper.

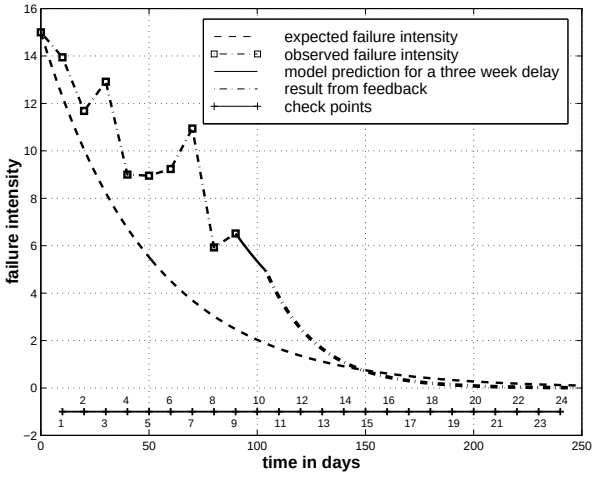

Figure 12. Result in the decrease of failure intensity from the application of feedback control in a disturbed test process after a three week delay.

5 Analysis

In a mature development environment the accuracy of the data collected is likely to be high due to the use of data and experience from past similar projects and due to a standardize development process. Such environments are likely to be found in companies at Levels 4 or 5 of the Capability Maturity Model (CMM) [15]. The accuracy of the data and the standardize development process makes the application of `Model S` more trustworthy but is not a requirement for the application of the model.

5.1 Analysis of `Case Study I`

As observed in Figure 6 the time needed to achieve the desired level of failure intensity (λ_f) increases to 107 days if no changes are made in the process at cp_5. At this point, the maximum eigenvalue of the system must be set to -0.1018 to make the test process converge to λ_f within 32 days. The increase in w_f needed to achieve this goal is presented in Figure 14. This goal can not be achieved by making changes only to γ, i.e., even if $\Delta\gamma$ is set to its maximum value it is not possible to complete the project by the expected deadline. A multidimensional solution of combined changes in w_f and γ is presented in Figure 15.

5.2 Analysis of the Simulation Results

An analysis of the results of the three simulation cases presented in Section 4.2 is given in this section.

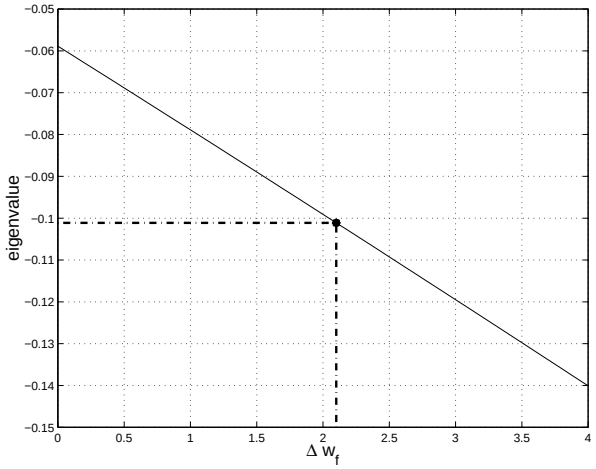

Figure 14. Eigenvalue relationship for changes in Δw_f for `Case Study I`.

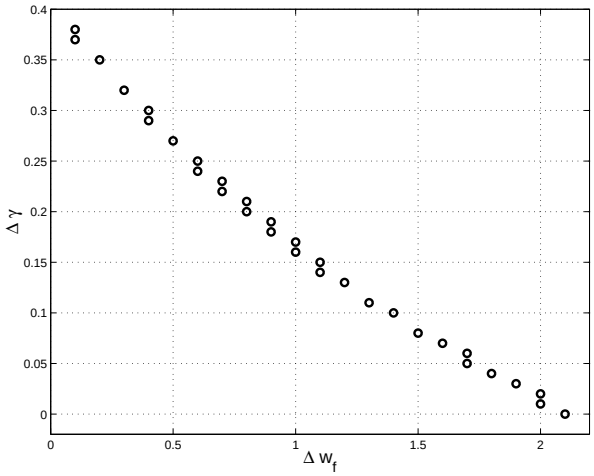

Figure 15. Alternatives of combined changes in Δw_f and $\Delta \gamma$ to achieve the desired results within the deadline for `Case Study I`.

Analysis of Simulation Case I

Though presenting a deviation from the expected failure intensity decay, changes in the process are not allowed for simulation case I and `Model S` is used to predict the behavior for the remaining period. This fact imposes a delay of 47 days to achieve the desired failure as observed in Figure 7. In this case the new deadline for the project has to be extended to day 197.

Analysis of Simulation Case II

At check point 9 (cp_9) the observed failure intensity is 6.52 and the desired one is 2.53. Therefore, the test manager must take some action to correct the deviation. Assume the test manager has decided in favor of a passive, i.e., has decided to correct the deviation in six check points and thus reach the desired failure intensity only by the deadline (cp_{15}). The eigenvalue needed to achieve this goal is -0.0357 and an increase of 2.3 in the size of the work force ($\Delta w_f = 2.3$) is one solution, as depicted in Figure 11, assuming that there are no changes in γ. The analysis of simulation case II also cloncludes that it is not possible to achieve the desired results by increasing the quality of the test process while retaining the work force at its current value. γ has a 0.6 value for the process under consideration. Therefore, in this case, $\Delta\gamma$ ranges from 0 to 0.4 and even its maximum value does not produce the desired eigenvalue. Alternatives to increase both w_f and γ ($\Delta w_f > 0$ and $\Delta\gamma > 0$) to place the system eigenvalue at -0.0357 are presented in Figure 16.

Analysis of Simulation Case III

The difference between simulation case II and III is a three week delay to implement the changes in the STP. At cp_9 the failure intensity value is 6.52 and must drop to 2.53 within 60 days. Due to the delay, feedback can not be directly applied in this case. `Model S` is then used to predict the failure intensity value of 4.9 for three weeks after cp_9. Feedback is then applied so as to set the system eigenvalue at -0.0415 and make the process converge from 4.9 to 2.53 in 45 days. The work force needs to be increased by 3.2, though once again it is not possible to achieve the desired results by only increasing γ. The set of alternatives to increase γ and w_f and place the system eigenvalue at -0.0415 is not presented here but the results are similar to the ones in Figure 16. In general we can conclude that: the longer the delay, the larger the changes needed to make the system converge to the desired value. Indeed, under certain circumstances it can slow down a test process instead of accelerating it as is indicated by the sensitivity analysis reported elsewhere [4].

6 Discussion

`Model S` is a state model for the control of the test process with failure intensity as the control variable. With little modification, the approach can be used to control a test process based on the estimated reliability or failure intensity (λ) of the product. Any one or more models for the computation of reliability could be used. `Model S` relates the effort and quality of the process to the failure intensity/reliability level and provides mechanisms to correct de-

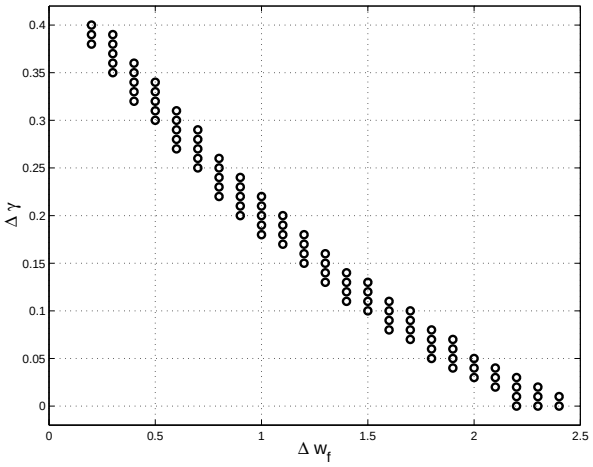

Figure 16. Alternatives of combined changes in Δw_f and $\Delta\gamma$ to achieve the desired results within the deadline for simulation case II.

viations in the process. A set of solution to correct such deviations is provided by the model. The use of `Model S` enhances the controllability and predictability of the STP emerging as a powerful tool to be used by test managers.

The results from the simulation runs and the case study using data in public domain offer evidence in support of the applicability of the model. Certainly additional experiments and case studies are needed to further study the model behavior, results presented here are encouraging.

A sensitivity analysis of our model [4] suggests changes to account for frictional forces related to the complexity of the product under test. Such changes will make the model behavior more accurate and more sensitive to changes in the s_c parameter.

References

[1] John W. Brewer. Matrix calculus and the sensitivity analysis of linear dynamic systems. *IEEE Transactions on Automatic Control*, 23(4):748–751, August 1978.

[2] João W. Cangussu, Raymond DeCarlo, and Aditya Mathur. A state variable model for the software test process. In *Proceedings of 13th International Conference on Software & Systems Engineering and their Applications(ICSSEA)*, Paris-France, December 2000.

[3] João W. Cangussu, Raymond DeCarlo, and Aditya Mathur. A formal model for the software test process. Technical Report SERC-TR-176-P, Purdue University-SERC, March 2001.

[4] João W. Cangussu, Raymond DeCarlo, and Aditya Mathur. Sensitivity analysis of a state variable model of the software test process. In *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference (SMC 2001)*, Tucson-Arizona, September 2001.

[5] João W. Cangussu, Raymond DeCarlo, and Aditya 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, September 2001.

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

[7] Willa K. Ehrlich, John P. Stampfel, and Jar R. Wu. Application of software reliability modeling to product quality and test process. In *Proceedings of ICSE*, pages 108–116, 1990.

[8] Graham C. Goodwin, Stefan F. Graebe, and Mario E. Salgado. *Control system design*. Prentice Hall, Upper Saddle River, N.J., 2001.

[9] F. Lanchester. Aircraft in warfare, the dawn of the fourth arm. Constable, London, 1916.

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

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

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

[13] John Musa. *Software Reliability Engineering*. McGraw-Hill, 1999.

[14] John D. Musa. Software reliability data. Data & Analysis Center for Software, January 1980.

[15] Marck C. Paulk and et al. Capability maturity model for software. Technical report, Software Engineering Institute, Carnegie Mellon University, 1993.

[16] P. A. Samuelson. Interactions between the multiplier analysis and the principle of acceleration. *Rev. Economic Statistics*, 21(7):75–78, May 1939.

[17] Nozer D. Singpurwalla and Simon P. Wilson. *Statistical Methods in Software Engineering: Reliability and Risk*. Springer-Verlag, New Your, NY, USA, 1999.