

Feedback and Adaptive Control for Software Testing

Kai-Yuan Cai¹

Department of Automatic Control
Beijing University of Aeronautics and Astronautics
Beijing 100083, China
kycai@buaa.edu.cn

João W. Cangussu

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

Ray A. DeCarlo

Department of Electrical and Computer Engineering
Purdue University
West Lafayette-IN 47907-1398, USA
decarlo@ecn.purdue.edu

Aditya P. Mathur²

Department of Computer Science
Purdue University
West Lafayette-IN 47907-1398, USA
apm@cs.purdue.edu

and

Scott Miller

Department of Computer Science
Purdue University
West Lafayette-IN 47907-1398, USA
s.miller@cs.purdue.edu

Abstract

Although qualitatively understood, a quantitative understanding of software testing

¹ Supported by the National Natural Science Foundation of China (60233020), the “863” Programme of China (2001AA113192) and the Aviation Science Foundation of China (01F51025).

² Supported in part by SERC and NSF

remains a challenging part of the software development process. The motivation for utilizing feedback and adaptive control mechanisms for software testing is to formalize, quantify, and optimize the feedback and learning mechanisms in the software test process and to make it more manageable, predictable, and thus more cost-effective with repeatable performance. In this paper we describe two approaches for feedback control of the software test process. One approach is based on a deterministic state variable model, and the other models the software test process as a controlled Markov chain. In this context adaptive software testing becomes the software testing counterpart of adaptive control. We also review related work and identify several open problems and future research directions. The new technology described in this paper is dramatically different from existing software technology in that the new technology rests on the quantification of feedback signals and enforces the closed-loop philosophy.

Keywords

Software testing, adaptive software testing, feedback control, adaptive control, software cybernetics

1 Introduction

Software testing is an integral part of any software development process. The test process comes in many flavors depending on the size of the software project, available budget, and the desired or acceptable quality of the end product. A variety of techniques and tools exist and are applied during the software test process. These include techniques and tools for test design, test automation, test application, test optimization, and test adequacy assessment. Application of one or more of these techniques and tools generally leads to improved quality but also entails cost. A significant amount of research effort has been spent in the development and evaluation of techniques and processes used in software testing. This has led to a myriad of excellent techniques useful in almost all phases of software testing, such as in unit test, integration test, system test, and regression test.

Despite the availability and utilization of such techniques, various important parameters of the test process and its end product, remain difficult to predict and control. Numbers of software defects or overall reliability are not directly incorporated into various software test data adequacy criteria [21]. Test case selections depend heavily on a tester's intuition and empirical rules of thumb. Various feedback mechanisms are seldom formalized, quantified or optimized. The question of how to achieve quantitatively an *a priori* reliability goal is seldom addressed.

In this work the focus is on questions related directly to the control of the test process. Section 2 explains why feedback and adaptive control is applied for software testing.

Section 3 describes a deterministic state model of the software test process. Section 4 describes the controlled Markov chains (CMC) approach to software testing, the adaptive software testing and the idea of software cybernetics. Sections 5 reviews related work. Sections 6 and 7 discuss open problems and future directions, respectively. Concluding remarks are contained in Section 8.

2 Why Feedback and Adaptive Control

To motivate the problem of control, consider the following scenario extracted from a commercial test process at week 14.

The system test phase for product P started at time $t=0$. The objective of the test phase is to (a) test P, find errors, and remove any errors found so as to bring its reliability to R and (b) the test process is to be completed at time $t=t_c>0$. It is assumed that testing and debugging take place concurrently. The progress of the test phase is reviewed at time $t=t_1<t_c$. The review reveals that the quality of the current version of P is $R_1<R$. The review team asks the following question: "Will the originally stated objectives be met if the test process continues in its current manner? And, if not then what changes, measured quantitatively, should be made to the process so that the objectives are met."

Not all software projects are expected to set up the objective in as precise terms as in this scenario. However, projects that deal with critical applications such as those found in health care devices and defense equipment are likely to encounter similar scenarios. The question stated in this scenario has two components: part (a) is the *predictive* component and part (b) is the *control* component. These questions could be answered using several techniques. For example, a test manager might use past experience augmented with data collected during the current system test phase to arrive at a suitable answer. In the study described in Section 3, we describe a different approach that uses a predictive model to answer the first part of the question and a *controller* in a feedback loop to answer the second part. For the specific scenario described above, our model's answer to the first part of the question was: "The originally stated objective will not be met. Several possibilities, and their impact, were suggested by the controller as an answer to the second part of the question. One possible change was to increase the size of the work force by 1.5."

From the above scenario we can further elaborate why feedback and adaptive control are applicable for software testing. First, software testing activities consist of planning, test case generation, test environment development, execution, test results evaluation, problem reporting/test log, and defect tracking [4]. Feedback is ubiquitous in these activities, although it may be in an informal manner. For example, in test case generation, tester's experience with similar software projects may help to determine which test suite is "best" for a new software project. Tester may also have better

understanding of the capability of a test suite in the late phase of software testing by examining the software testing data in the early phase of software testing and thus select better test cases. In problem reporting/test log, all software testing activities are documented. The used test cases and the phenomena, expected or unexpected, correct or incorrect, are recorded. The problem reporting/test log forms the basis for later debugging and fixing the “bugs” which were observed as failures during testing. They also help software tester to regression test the faulty software components which were returned to software developer for debugging.

Second, learning is also ubiquitous in software testing. By learning we mean that tester’s understanding of the software under test, the capability of a test suite and the achievable testing goals is improved during software testing. As more and more test cases are applied and defects are detected and removed, tester will be able to analyze the defect distributions with respect to defect types and program modules. The tester can judge which modules are more or less defect-prone. Learning can be treated as a special form of feedback compensation and it is easy to understand that feedback and learning play an essential role in software testing.

Third, in order to make software test processes more repeatable, manageable and predictable, it is necessary to formalize the underlying mechanisms of feedback and learning. Without formalization, software testing activities would continue to depend heavily on human intuition, rules of thumb and lack theoretically rigorous foundation. Software testing would continue to be an art with only qualitative understanding.

Fourth, in order to optimize software testing activities, the possible closed-loop feedback-based software testing strategies must be in comparison with open-loop software testing strategies to decide which kinds of software testing strategies behave better. In this way an inverse question of software testing can be addressed: given a testing goal, how to synthesize a software testing strategy to achieve the goal in an optimal manner.

Fifth, feedback and optimization are two central themes in the control community and there are mature theories of feedback and adaptive control [1, 23]. In feedback control the history of inputs and outputs of the controlled object is used to determine updated control forces to be applied to the controlled object. A controller implements the updates. The controller and the controlled object constitute a closed-loop feedback control systems. In adaptive control the history of inputs and outputs are further used to identify the potential parametric and/or structural changes occurring in the controlled object. The corresponding controller adjusts its energies accordingly to accommodate these changes and to keep the control system behaving properly. The maturity of the theories and the great successes of feedback and adaptive control in various engineering fields suggest that the principles of feedback and adaptive control are ready to be adapted to the new area of software testing.

Finally, previous work has shown that feedback and adaptive control markedly improve the software testing process. Specifically, after proper quantitative modeling feedback control can be applied to the management of software test processes while adaptive control principles lead to quantitative approach termed adaptive software testing. Indeed, the principles of feedback and adaptive control can be further extended to the broad area of software cybernetics that explores the interplay between software and control. Sections 3 and 4 will explain this novel extension in more details.

In summary, feedback and adaptive control is seen as the next feasible and workable in the evolution of software testing.

3 A Deterministic State Model of the Software Test Process

The software industry is constantly affected by delays in the release of products. Additionally, the quality of these products, in terms of an estimated number of remaining defects, is generally unknown. These problems are not only associated with the techniques used to develop the software product but also with the lack of control and prediction techniques to help managers regulate the development process. A solution for this problem would be the existence of a control mechanism to regulate the software process similar to what is available in other engineering disciplines. A mathematical model capturing the dominant behavior of the process is required to achieve this goal. This is reinforced by Lehman's statement: "Mastery of feedback mechanisms demands adequate quantitative models individually calibrated against real world software evolution environments." [35]. The fact that software development is not a physical process may appear to be a barrier but the existence of many mathematical models in economics, biology, and many other disciplines added to the successful preliminary results in the modeling of the test process serve to void such assumptions.

The linear deterministic model of the STP (software test process) is based upon three assumptions. These assumptions and the corresponding equations are presented below [16]. They are based on an analogy of the STP with the physical process typified by a spring-mass-dashpot system and also by Volterra's predator-prey model [38]. A description and justification of this analogy and the choice of a linear model can be found elsewhere [16]. The model has been validated using sets of data from testing projects and also by means of an extremal case evaluation and by a sensitivity analysis [17].

Assumption 1 The rate at which the velocity of the remaining errors changes is directly proportional to the net applied effort (e_n) and inversely proportional to the complexity (s_c) of the program under test, i.e.,

$$\dot{r}(t) = \frac{e_n(t)}{s_c} \Rightarrow e_n(t) = \dot{r}(t)s_c \quad (3.1)$$

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

$$e_f(t) = \zeta(s_c)w_f r(t) \quad (3.2)$$

where $\zeta(s_c) = \frac{\zeta}{s_c^b}$ is a function of software complexity. Parameter b depends on the characteristics of the product under test.

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

$$e_r(t) = -\xi \frac{1}{\gamma} \dot{r} \quad (3.3)$$

Combining Equations (3.1), (3.2), and (3.3) in a force balance equation and organizing it in a state variable format ($\dot{x} = Ax + Bu$) [38] produces the following system of equations

$$\begin{bmatrix} \dot{r}(t) \\ \dot{r}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\zeta w_f & -\xi \\ \frac{1}{s_c^{(1+b)}} & \frac{1}{\gamma s_c} \end{bmatrix} \begin{bmatrix} r(t) \\ \dot{r}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ \frac{1}{s_c} \end{bmatrix} F_d(t) \quad (3.4)$$

F_d above is included in the model to account for unforeseen disturbances[15] such as hardware failures, personnel illness or any event that slows down or even interrupts the continuation of the test process.

Along with the model in Equation (3.4) an algorithm, based on system identification techniques [37], has been developed to calibrate the parameters of the model [16]. Using data available and an estimation of \dot{r} the error difference for a specific period of time $D^i = [r(i) \ \dot{r}(i)]^T - [r(i-1) \ \dot{r}(i-1)]^T$ is computed. It can be shown that

$D^i = MD^{i-1}$, where $M = e^{AT}$, T being the time increment between two consecutive

measurements of data, and A the A-matrix of Eq. (3.4). Therefore, matrix M can compute from

$$R_1 = MR_2 \Rightarrow R_1 R_2^{-R} \quad (3.5)$$

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

The Spectrum Mapping Theorem [23] shows that the eigenvalues of M , say λ_1^M and λ_2^M , have the following relation with the eigenvalues of matrix A : $\lambda_1^M = e^{\lambda_1 T}$ and $\lambda_2^M = e^{\lambda_2 T}$. Therefore $\lambda_1 = \frac{1}{T} \ln(\lambda_1^M)$ and $\lambda_2 = \frac{1}{T} \ln(\lambda_2^M)$. The eigenvalues are the roots of the characteristic polynomial of the A -matrix, as in Eqn. (3.4), which is

$$\Pi_A(\lambda) = \det[\lambda I - A] = \lambda^2 + \frac{\xi}{\hat{\gamma} s_c} \lambda + \frac{\zeta \hat{w}_f}{s_c^{(1+b)}} \quad (3.6)$$

Since ξ and ζ are the only unknowns at this time, they can be estimated by matching the roots of $\Pi_A(\lambda)$ to λ_1 and λ_2 computed above.

The fast convergence presented by the algorithm increases the model applicability and accuracy [14]. Finally, a parametric control procedure is used to compute required changes in the model in order to converge to desired results according to time constraints [16].

The deterministic model has already been successfully applied to two distinct projects. The model was first used on the error data reported by Knuth on the development of $T_E X$ [31]. Though applied to a finished process, the experiment showed the model had produced results very similar to the ones presented by the error data. The results were encouraging for the application of the model in an ongoing software test process in a large industrial project. The project was the translation of four million lines of Cobol to SAP/R3 code. Using data collected from the first 14 weeks the model was used to predict a delay of 10 weeks to achieve the specified goals with no changes in the process. The feedback application predicted that an increase of 1.5 in the work force would be necessary to finish the project within the deadline. No actions were taken by the manager and the process finished with a delay of 12 weeks showing an accuracy of 94.5% on the prediction of the total time to complete the project [16].

4 The CMC Approach, Adaptive Software Testing, and Software Cybernetics

The controlled Markov chains (CMC) approach to software testing treats software testing as an (optimal) control problem as shown in Figure 4.1. The software under test serves as the controlled object and is modeled as a controlled Markov chain. The software testing strategy serves as the corresponding controller and is synthesized by using the theory of controlled Markov chains to optimize the underlying performance criterion. The software under test and the corresponding testing strategy constitute a closed-loop feedback control system.

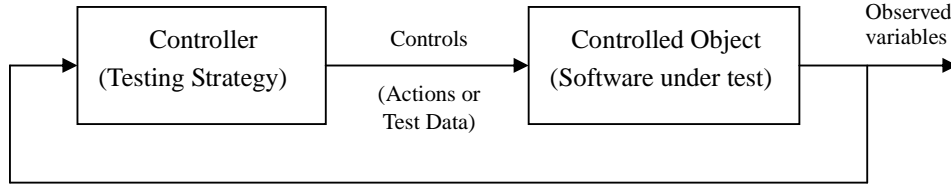


Figure 4.1 Software Testing as an Optimal Control Problem

A controlled Markov chain is five-tuple $\langle S, A, \{A(y)|y \in S\}, Q, W \rangle$ [24, 28], where S and A are given sets, called the state space and the control (action) set, respectively. We assume that S is discrete. $\{A(y)|y \in S\}$ is a family of nonempty subsets $A(y)$ of A , with $A(y)$ being the set of feasible controls or actions in the state $y \in S$. Q is a transition law such that

$$Q(B|y, a) = \Pr\{Y_{t+1} \in B | Y_t = y, A_t = a\}, B \subset S$$

where Y_t denotes the system state at time t , and A_t the action (control) applied at time t which is discrete. Finally, W is a cost-per-stage (or one-stage cost) function. If the system is in the state $Y_t = y \in S$ at time t and the control $A_t = a \in A(y)$ is applied, then two things happen: (1) a cost $W_y(a)$ is incurred, and (2) the system moves to the next state Y_{t+1} according to the transition law Q . Once the transition into the new state has occurred, a new control is chosen and the process is repeated.

The CMC approach to software testing is aimed to formalize, quantify, and optimize the feedback mechanisms in software testing and address such a question: given a testing goal (e.g., reliability objective, cost constraint), how to synthesize an optimal testing strategy that achieves the goal. In partition testing, an action may be interpreted as selection of an equivalence class and a test case thereof. The cost incurred may conclude the cost of selecting the test case and the CPU time of

executing the test case. Depending on the different interpretations of state, action, and cost, many software testing problems can be formulated in a single framework of CMC. For example,

- (1). Software testing for optimal reliability growth [6]. Software state is defined as the number of defects remaining in the software under test, and the testing goal is to detect and remove all the remaining software defects by using at least expected number of test cases.
- (2). Optimal software testing within given number of tests [13]. Software state is defined in terms of the number of defects remaining in the software under test as well as the number of remaining tests allowed to be applied. The testing goal is to detect and remove defects as many as possible within the given number of tests.
- (3). Optimal stopping of software testing [7]. Software state is defined as the number of defects remaining in the software under test, and the testing goal is to minimize the expected sum of the cost of software testing and the cost incurred by defects exposed in field operations.
- (4). Software testing for optimal reliability assessment [12]. Software state is defined as a vector, representing the numbers of test cases selected from various equivalence classes and the corresponding numbers of observed failures, and the testing goal is to minimize the variance of an unbiased software reliability estimator.

In treating software testing as an optimal control problem the related software parameters such as software defect detection rates and the initial number of software defects are assumed known. However these software parameters are actually unknown. Adaptive software testing alleviates this problem by estimating software parameters on-line, using the testing data collected during software testing as shown in Figure 4.2. Adaptive software testing is the counterpart of adaptive control in software testing. Two case studies, one is on the Space program written in C language [10], and the other on a software data collection program written in C++ language [11], show that adaptive software testing can really work in practice and may significantly outperform the random testing strategies in terms of the average number of test cases used to detect and remove given number of defects and the corresponding variance, which may be reduced up to 35% on average and 30%, respectively.

Adaptive software testing defines a topic of the emerging area of software cybernetics that explores the interplay between software and control [5, 9]. In general, software cybernetics addresses

- (1). How to formalize and quantify feedback mechanisms in software processes and systems;
- (2). How to adapt control theory principles to software processes and systems;
- (3). How to apply the principles of software theories and engineering to control systems and processes; and
- (4). How to integrate the theories of software engineering and control engineering.

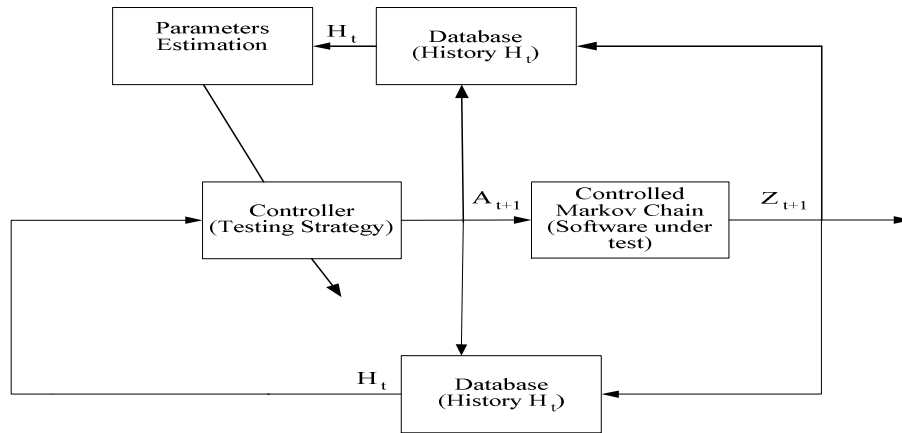


Figure 4.2 Diagram of Adaptive Software Testing

5 Related Work

Lehman is one of the first who attach full importance to the feedback mechanisms in software engineering [33, 34]. He classifies software as S-type, P-type software and E-type software. An E-type software system solves a problem or implements a computer application in the real world and thus the validity or effectiveness of it depends on the environment it is embedded. Lehman argues that E-type software systems are change or evolution prone and there are various laws of software evolution which are closely related to the underlying feedback mechanisms in the E-type software systems.

Discrete event simulation techniques are commonly used to model and evaluate the SDP [27]. However, the control capability of Software Process Simulation is restricted to answer "what if" questions and therefore does not present a self regulation mechanism. Optimization can be achieved using simulation, but the number of possible combinations of changes in the parameters of the model can reach high values making simulation a very costly technique, although it is capable of determining results similar to the sensitivity techniques used in the state model [17].

Though none has achieved overall acceptance [3, 44], Processes Programming Languages are another alternative to model and emulate a software development process and a myriad of them are available. An example of a process language suitable for software processes is Little-JIL [18]. The language is appealing due to its graphical nature and understandability for non-programmers while still presenting a well defined semantic. The design of Little-JIL makes an explicit distinction between coordination and other language features such as resource management and communication facilitating its understanding and use. The language also presents a control paradigm for process control flow [44]. Though being an innovative control flow approach, Little-JIL, as any other process language, does not provide a closed

loop feedback solution for the control of the SDP.

Feedback mechanisms are adopted in both the antirandom testing and the adaptive random testing. In the antirandom testing a new test case is selected so that it has maximum distance to the test cases that has been selected [39]. In the adaptive random testing the failure patterns or the distance between selected test cases and non-selected test cases are also used to improve the effectiveness of the simple random testing [21].

The emerging adaptive software systems demonstrate that feedback mechanisms can be used to improve the system behavior and performance. An example is an adaptive user interface that provides individualized, just-in-time assistance to user by recording user interface events and frequencies, organizing them into episodes, and automatically deriving patterns [36]. The theories of feedback control including supervisory control may play an important role in building adaptive software systems [29, 32].

The software project schedule problems of staffing, project completion time and cost can be formulated in the setting of Markov decision processes and treated as an optimal decision problem, where the history of decision-making is essential for making next decisions [41].

A ConnectedSpace is a collection of one or more smart devices which are controlled via embedded software and capable of connecting to a local network or the Internet. It is shown that the theory of supervisory control of discrete event systems can be used to synthesize the required safety controller that guarantees the safe operation of the ConnectedSpace under consideration [42, 43].

Synthesis of reactive software is one more topic that the supervisory control can play a role. The software operating environment serves as a controlled object that is modeled as a discrete event system, and the reactive software to be synthesized serves as the corresponding controller. The theory of supervisory control is then used to synthesize the reactive software so that the required software state properties of, e.g., reachability and invariance are guaranteed [40, 45].

6 Open Problems

Nature of the software test process. Halstead argued that software systems obeyed physics-like laws [26], and Lehman argued there were laws of software evolution [33, 34]. From the partial-repeatability perspective software systems and processes obey certain laws that are not fully repeatable [8]. A natural question is that whether the software test process obeys certain laws that are irrespective of software systems and human intention. We note that there may be the counterpart of Newton laws in

software testing [16, 17], and software defect curves, or the relationships between the estimate of the number of defects remaining in the software under test and the number of observed failures as software testing proceeds, often demonstrates a trapezoidal pattern [2]. The open problem is what the essential features of laws the software test process obeys in comparison with conventional physical laws and how many laws of software test process there may be.

Measurement. Though many advanced and useful metrics are available to measure different aspects of the development process, there is still no agreement of a single metric to be used in all situations. Different companies or groups inside one company use different metrics. Therefore the techniques used to quantify the features of the SDP need to be flexible enough to accommodate the use of distinct metrics while useful enough to properly characterize the features what does not constitute a straightforward task. Measurement is a key aspect of the closed loop feedback application since what cannot be measure cannot be controlled. The problem is how to develop a satisfactory measurement framework for the software test process.

Validation. Defect and any project related data are proprietary and considered very sensitive for almost all the companies doing software development. The availability of data is crucial for the proper validation of any model. Even when a large set of data is available, validation is not a simple task due to the diversity of scenarios represented by the data. The task becomes even more challenging due to the difficult of obtain data from companies. Hopefully, good results and the availability of friendly tools hiding the complexity of the models will start breaking this barrier. The problem is what criteria should be followed in validating a given model.

Performance analysis. Although simulation and case studies show that the adaptive testing strategies may use fewer test cases to detect more defects and behave more stable in performance [6, 10, 11], formal performance analysis is required to enforce the simulation and empirical results. A difficult and open problem is how much errors in parameter estimation can be tolerated to make a correct decision of test case selection. Different parameter estimation methods may lead to different performance of adaptive software testing.

Direct adaptive software testing. The adaptive software testing described in Section 4 is a form of indirect adaptive software testing since parameters are estimated and updated on-line and the estimates are used for on-line decision-making. Direct adaptive software testing makes on-line decisions of test case selection directly without invoking parameter estimation. The problem is how to develop a proper direct adaptive software testing strategy.

Debugging. Testing and debugging are often intermingled in complex ways. Therefore we prefer to treat these two as distinct though related component processes of the STP. Testing is concerned primarily with test design, test set up and execution,

analysis of test results, and reporting of failures. Debugging is concerned with determining the cause of failures and repairing the program under test. How to model and control testing and debugging processes in an integrated manner is an open problem.

Effects. What are the different ways in which feedback can be applied into the software test process? What is the cost and benefit associated with these different ways of changing an existing test process? These problems are open and challenging.

Applicability. Are the techniques of feedback and adaptive control general enough so that they can be applied to any phase of the software test process? Or to any type of software process? Or, the techniques of feedback and adaptive control, and the associated models, differ in their applicability to the software test process depending on the test phase and how the process is organized? These problems are open should be addressed in future investigation.

7 Future Directions

Expansion of the STP State Model

The State Model for the Software Test Process has shown to be adequate to capture the dominant aspects of the test process [16]. However, the sensitivity analysis has also shown that other important aspects of the process are necessary to make the model more complete and precise [17]. These aspects are the communication overhead and the learning curve.

It is well known that communication overhead can and will, most likely, cause a delay in the realization of a project. Therefore, the inclusion of this aspect in the state model for the software test process seems to be required in order to improve its accuracy. In the absence of this feature, the model does not account for additional frictional forces when communication increases due to an approaching deadline or due to the increase in the size of the work force.

When the test process starts test teams, in general, are not familiar to the product they are testing. The adaptation to the product and the process itself is dependent on many parameters. Are the testers familiar with the language used to implement the product? Are they familiar to the testing tool being used? Do they have experience with similar projects? These and many other questions should be answered to determine how fast the test team will adapt to the product/process. This adaptation is related to a learning curve delaying the realization of the process. Also, the late inclusion of new members to the test team will slow down the learning curve. All these aspects must be included in the state model as pointed out by the sensitivity analysis [17].

Another aspect that needs consideration is the development of a Stochastic Model for the STP. Despite the difficulty in creating mathematical models, plausible accurate models have been derived for physical and non-physical systems [30]. However, the difficulty level arises when modeling non-physical system due to the fact that observation/measurement of these processes is not accurate, more specifically; they are subject to disturbances and noisy data. Under these circumstances, a stochastic rather than a deterministic model appears to be an alternative solution to represent the process. The software test process presents such characteristics and seems to be suitable for a stochastic approach.

The deterministic model seems to be appropriated to control relatively well defined test processes, where the level of unpredictability is not critical. However, in practice, test processes are subject to a variety of external disturbances. In addition, collected data is often noisy and prediction of the intermediate states of the process may be compromised depending on the level of inaccuracy of the data. Under such circumstances, one alternative would be the use of stochastic [19, 25] model by adding two stochastic processes $\eta(t)$ and $\varphi(t)$ to the state model ($\dot{x}(t) = Ax(t) + Bu(t)\eta(t)$ and $y(t) = Cx(t) + \varphi(t)$). The inclusion of noise components represents the major difference between the deterministic and the stochastic model. The influence of randomly external disturbance is accounted for by the noise sequence $\eta(t)$ whereas the inaccuracy of the collected/measured data is represented by the noise sequence $\varphi(t)$.

Expansion of the CMC Approach

The CMC approach has been shown widely applicable for software testing. It can deal with the testing problems for optimal reliability improvement, optimal reliability assessment, and optimal stopping of testing. Recall that software testing may be conducted for assessment of capability of test suite and this is particularly true for mutation testing. It is not clear that how the CMC approach can deal with mutation testing so that the otherwise heavily burden of mutation testing can be reduced.

Another problem in software testing is the oracle problem, and in some circumstances there may not be absolute oracle for the software under test [22]. How to adapt the CMC approach to the software testing problem in the absence of test oracle is topic for future investigation.

Software testing is subject to testing resource constraints (e.g., financial constraints, release constraints). How to adapt the CMC approach to deal with testing resource constraints is one more topic for future investigation.

Combination with conventional software technology

Conventional software technology is dominant in software engineering practice, where various feedback mechanisms are not formalized, quantified or optimized. For example, white-box testing techniques are popular in unit testing, and functional testing or black-box testing is widely applied in integration and system testing. It is unlike for feedback and adaptive control theory to stand alone in software engineering practice. Feedback and adaptive control theory is not aimed to throw out conventional software technology. Rather, it is aimed to improve them. So, we may talk about adaptive structural testing, adaptive process improvement, and so on.

Development of solid theoretical foundation

Solid theoretical foundation is badly needed for feedback and adaptive control for software testing, although two complimentary approaches have been developed as described in Sections 3 and 4. Many theoretical problems need further investigation. For example, what is the scope of software testing scenarios that the linear state model is applicable? From the control perspective, what are the relationships about controllability, observation, and testability? Under what circumstances the adaptive software testing certainly outperforms the random testing? What is the on-line computational complexity of the adaptive software testing?

Large-scale commercial case studies

The success or failure of the feedback and adaptive control for software testing will eventually be judged on the basis of large-scale commercial practice, although the theoretical soundness and laboratory case studies can justify the applicability of feedback and adaptive control for software testing. The feedback and adaptive control should eventually lead to substantial improvements on software test process management and new effective and efficient software testing techniques. It should make software test process more repeatable, predictable and manageable. It should simplify the software test process and make it more cost-effective.

8 Concluding Remarks

The existing mainstream technology for software testing mainly follows the open-loop or what-if philosophy, although feedback and learning is ubiquitous in the software test process. This results in a significant gap between software testing practice and human expectation. The software test process is, from a control perspective, still ill-understood. The motivation of formalizing, quantifying and optimizing the feedback and learning mechanisms in software testing is to adopt the principles and theories of feedback and adaptive control to the software testing and

make the software test process more repeatable, manageable, predictable and thus more cost-effective. To this end, two complimentary approaches have been developed. One approach is based on a deterministic state variable model, and the other models the software test process as a controlled Markov chain. Feedback control is adopted in both approaches. Additionally, the adaptive software testing adopts learning to the software test process and serves as a counterpart of adaptive control in software testing. The technology described in this paper follows the closed-loop philosophy.

Besides the two approaches mentioned above, in the preceding sections we also review the related work, identify open problems and future research directions. The approaches described in this paper should be expanded to deal with more scenarios. They should be in cooperation with existing software technology to improve the software test process. Solid theoretical foundation should be developed for them and large-scale commercial practices are required. Following the approaches and philosophy described in this paper, we can expect that the new and broad area of software cybernetics is emerging to explore the interplay between software and control.

References

1. K.J.Astrom, B.Wittenmark, *Adaptive Control*, Addison-Wesley Publishing Company, 1989.
2. C.Bai, K.Y.Cai, T.Y.Chen, "An Efficient Estimation Method for Software Defect Curves", *Proc. the 29th IEEE Computer Software and Application Conference*, 2003, pp534-539.
3. S.Bandinelli, A.Fuggeta, S.Grigolli, "Process Modeling in-the-large with Slang", *Proc. 2nd International Conference on the Software Process*, IEEE Computer Society Press, 1993, pp75-83.
4. A.Bertolino, "Software Testing", in: A.Abran, et al, (editors), *Swebok: Guide to the Software Engineering Body of Knowledge (Trial Version)*, IEEE Computer Society Press, 2001, Chapter 5.
5. K.Y.Cai, "On the Concepts of Total Systems, Total Dependability and Software Cybernetics", (unpublished manuscript), Centre for Software Reliability, City University, London, Draft version, October 1994; revised version, July 1995.
6. K.Y.Cai, "Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics", *Information and Software Technology*, Vol.44, 2002, pp841-855.
7. K.Y.Cai, "Optimal Stopping of Multi-project Software Testing in the Context of Software Cybernetics", *Science in China (Series F)*, Vol.46, No.5, 2003, pp335-354.
8. K.Y.Cai, L.Chen, "Analyzing Software Science Data with Partial Repeatability", *Journal of Systems and Software*, Vol.63, 2002, pp173-186.
9. K.Y.Cai, T.Y.Chen, T.H.Tse, "Towards Research on Software Cybernetics", *Proc.*

- 7th *IEEE International Symposium on High Assurance Systems Engineering*, 2002, pp240-241.
10. K.Y.Cai, B.Gu, H.Hai, Y.C.Li, "Adaptive Software Testing with Fixed-Memory Feedback", working paper, 2003, pp32-39.
 11. K.Y.Cai, T.Jin, "A Case Study of Adaptive Software Testing", in preparation, 2003.
 12. K.Y.Cai, Y.C.Li, K.Liu, "How to Test Software for Optimal Software Reliability Assessment", *Proc. the 3rd International Conference on Quality Software*, IEEE Computer Society Press, 2003, pp32-39.
 13. K.Y.Cai, Y.C.Li, W.Y.Ning, "Optimal Software Testing in the Setting of Controlled Markov Chains", *European Journal of Operational Research*, accepted for publication, 2003.
 14. J.W.Cangussu, "Convergence Assessment of the Calibration Algorithm for the State Variable Model of the Software Test Process", *Proc. the IASTED International Conference on Software Engineering*, 2003, pp10-13.
 15. J.W.Cangussu, R.A.DeCarlo, A.P.Mathur, "Effect of Disturbances on the Covergence of Failure Intensity", *Proc. the 13th International Symposium on Software Reliability Engineering*, 2002, pp12-15.
 16. J.W.Cangussu, R.A.DeCarlo, A.P.Mathur, "A Formal Model for the Software Test Process", *IEEE Transactions on Software Engineering*, Vol.28, No.8, 2002, pp782-796.
 17. J.W.Cangussu, R.A.DeCarlo, A.P.Mathur, "Using Sensitivity Analysis to Validate a State Variable Model of the Software Test Process", *IEEE Transactions on Software Engineering*, Vol.29, No.5, 2003, pp430-443.
 18. A.G.Cass, B.S.Lerner, F.K.McCall, L.J.Osterweil, S.M.Sutton, A.Wise, "Little-jil/juliette: A Process Definition Language and Interpreter", *Proc. 22nd International Conference on Software Engineering*, 2000, pp754-757.
 19. G.Chen, G.Chen, S.H.Hsu, *Linear Stochastic Control Systems*, CRC Press, 1995.
 20. M.H.Chen, M.R.Lyu, W.E.Wong, "Effect of Code Coverage on Software Reliability Measurement", *IEEE Transactions on Reliability*, Vol.50, No.2, 2001, pp165-170.
 21. T.Y.Chen, T.H.Tse, Y.T.Yu, "Proportional Sampling Strategy: a Compendium and Some Insights", *Journal of Systems and Software*, Vol.58, 2001, pp65-81.
 22. T.Y.Chen, T.H.Tse, Z.Q.Zhou, "Fault-based Testing without the Need of Oracles", *Information and Software Technology*, Vol.45, 2003, pp1-9.
 23. R.A.DeCarlo, *Linear Systems: a State Variable Approach with Numerical Implementation*, Prentice-Hall, 1989.
 24. C.Derman, *Finite State Markovian Decision Processes*, Academic Press, 1970.
 25. M.S.Grewal, A.P.Andrews, *Kalam Filtering: Theory and Practice Using MATLAB* (2nd edition), John Wiley & Sons, 2001.
 26. M.H.Halstead, *Elements of Software Science*, Elsevier, 1977.
 27. G.A.Hansen, "Simulating Software Development Processes", *IEEE Computer*, Vol.29, January 1996, pp73-77.
 28. O.Hernandez-Lerma, *Adaptive Markov Control Processes*, Springer-Verlag, 1989.

29. G.Karsai, J.Sztipanovits, "A Model-based Approach to Self-Adaptive Software", *IEEE Intelligent Systems*, May/June 1999, pp46-53.
30. H.Khalil, *Nonlinear Systems*, Prentice-Hall, 1996.
31. D.E.Knuth, "The Errors of TeX", *Software Practice and Experience*, Vol. 19, No. 7, 1989, pp 607-685.
32. M.M.Kokar, K.Baclawski, Y.A.Eracar, "Control Theory-Based Foundations of Self-Controlling Software", *IEEE Intelligent Systems*, May/June 1999, pp37-45.
33. M.M.Lehman, "Software Engineering, the Software Process and their Support", *Software Engineering Journal*, Vol.6, No.5, 1991, pp243-258.
34. M.M.Lehman, "Laws of Software Evolution Revisited". In: *Software process technology*, (C.Montangero ed.), Springer-Verlag, 1996, pp108-124.
35. M.M.Lehman, "Process Modeling – Where next?", *Proceedings of International Conference on Software Engineering*, 1997, pp 549-552, Boston, Massachusetts.
36. J.Liu, C.K.Wong, K.K.Hui, "An Adaptive User Interface Based on Personalized Learning", *IEEE Intelligent Systems*, March/April 2003, pp52-57.
37. L.Ljung, *System Identification: Theory for the User*, Prentice-Hall, 1987.
38. D.G.Luenberger, *Introduction to Dynamic Systems: Theory, Models and Applications*, John Wiley & Sons, 1979.
39. Y.K.Malaiya, "Antirandom Testing: Getting the most out of Black-box Testing", *Proc. the 6th International Symposium on Software Reliability Engineering*, 1995, pp86-95.
40. H.Marchand, M.Samaan, "Incremented Design of a Power Transformer Station Controller Using a Controller Synthesis Methodology" *IEEE Transactions on Software Engineering*, Vol.26, No.8, 2000, pp729-741.
41. F.Padberg, "Scheduling Software Projects to Minimize the Development Time and Cost with a Given Staff", *Proc. Asia-Pacific Software Engineering Conference*, 2001, pp187-194.
42. B.Sridharan, A.P.Mathur, K.Y.Cai, "Using Supervisory Control to Synthesize Safety Controllers for ConnectedSpaces", *Proc. the 3rd International Conference on Quality Software*, IEEE Computer Society Press, 2003, pp186-193.
43. B.Sridharan, A.P.Mathur, K.Y.Cai, "Synthesizing Distributed Controller for Safe Operation of ConnectedSpaces", *Proc. the IEEE International Conference on Pervasive Computing and Communication*, 2003, pp452-459.
44. S.M.Sutton, Jr., L.J.Osterweil, "The Design of a Next-Generation Process Language", *Proc. the 5th Symposium on the Foundation of Software Engineering*, ACM SIGSOFT, September 1997, pp142-158.
45. X.Y.Wang, Y.C.Li, K.Y.Cai, "On the Polynomial Dynamical System Approach to Software Development", *Science in China (Series F)*, accepted for publication, 2003.