An Overview of Software Cybernetics

Kai-Yuan Cai[*] [1], João W. Cangussu[$], Ray A. DeCarlo[&] *and* Aditya P. Mathur [2]

[*] *Department of Automatic Control*
Beijing University of Aeronautics and Astronautics
Beijing 100083, China
kycai@buaa.edu.cn
[$] *Department of Computer Science*
University of Texas at Dallas
Richardson - TX 75083-0688, USA
cangussu@utdallas.edu
[&] *Department of Electrical and Computer Engineering*
Purdue University
West Lafayette-IN 47907-1398, USA
decarlo@ecn.ptwdue.edu
Department of Computer Science
Purdue University
West Lafayette-IN 47907-1398, USA
apm@cs.purdue.edu

Abstract

Software cybernetics explores the interplay between software and control and is motivated by the fundamental question whether or not and how software behavior can be controlled. In this paper we formulate the underlying motivations and ideas of software cybernetics and review various existing research topics in this emerging area, including feedback mechanisms in software processes, bisimulation and controllability, adaptive software. software synthesis, software test process control. and adaptive testing. We identify software rejuvenation and performance control. sojiware fault-tolerance, logical foundation for control systems, and communication complexity for control systenis as potential research topics Several on-going research projects are also summarized.

[1] Cai was 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] Mathur was supported in part by SERC and NSE

1. Introduction

Certainly, the introduction and utilization of the notion of computability and the advent of software technology constitute one of the great achievements of scientific human history [1, 2]. Computing and software technologies significantly impact economical development, the entertainment industry, and scientific activities everywhere. Indeed, the current age of information depends critically on computing and software technologies. This argument is further strengthened by the establishment of various inter-disciplinary areas such as computing mathematics, computational fluid mechanics, and computer simulation and by the new and growing internet-based areas of e-commerce [3], virtual laboratories [4], and e-science [5]. This essential importance of software technologies raises a number of fundamental questions: What computable specifications or services can software systems implement [6, 7]? What is the nature of software behavior [8]? How can software behavior be formalized [9, 10]? How dependabie are software systems [11]?

The emerging inter-disciplinary area of software cybernetics is motivated by the fundamental question of

whether or not and how software behavior can be controlled. Software behavior includes the behavior of software development processes, software maintenance processes, software evolution processes as well as that of software execution itself. Various related works have been reported in scattered conference proceedings, research journals and edited books under different umbrellas such as adaptive software [12], feedback control of the software test process [13], and adaptive testing [14], among others. Although common or similar ideas underlie these works, they need a more rigorous and quantitative formulation and some unifying framework which is precisely the purpose of the new area of software cybernetics. As such, it is necessary to establish a reference point as to the state of the art of this area. We then ask the question: which research topics have the greatest potential for resolution using the emerging tools of software cybernetics? Our purpose is to provide some snap shots in answer to this question.

The aim of this paper is three-fold. First, the paper will formulate the underlying motivations and ideas of software cybernetics. Section 2 addresses this issue. Second, the paper will review various related and existing research topics in the area of software cybernetics. This will be done in Section **3.** Third, in Section **4,** several research topics that have not been approached but may have great potential in the future are identified. In addition, Section **5** will summarize several on-going research projects of the present authors. Concluding remarks are contained in Section 6.

2. Software Cybernetics Concepts and Definitions

The emerging area of software cybernetics explores the interplay between software processes and control [15-17]. Nobert Wiener's Cybernetics is concerned with control and communication in the animal and the machine [20]. If we treat software as a part of the machine, then a simple interpretation of the term "software cybernetics" is that it is concerned with the control and communication in software. However as will be discussed in the rest of this paper, there is potential for the principles and theories of software to play a role in control, we should not stick to Wiener's view of cybernetics and should better interpret software cybernetics as the interplay between software and control. In general, software cybernetics addresses issues and questions on: (i)-the formalization and quantification of feedback mechanisms in software processes and systems; (ii)-the adaptation of control theory principles to software processes and systems; (iii)-the application of the principles of software theories and engineering to control systems and processes; and (iv)-the integration of the theories of software engineering and control engineering.

Here we adopt the following definitions.

Software: by software we mean the software development process, the software maintenance process, the software evolution process and all related artifacts delivered during such processes, including the delivered software system.

Control: a finite or infinite sequence of actions that are taken and applied to the controlled object to achieve a given *a priori* goal for the controlled object in a systematic and repeatable manner. Several components are identified. First, a single goal or a set of goals is given before a control is applied. Second, actions are selected from a finite or infinite set of actions. Third, a sequence of actions, rather than a single action, is taken; control is a dynamic process and the dimension of time is involved. Four, the actions are taken in a systematic manner and thus there must be cooperation between and/or constraints on the taken actions; the overall effect of the taken actions is that the given goal or set of goals is achieved. Finally, the sequence of actions is repeatable. Thus the taken actions must be interpretable in the sense that the underlying reason for taking the actions can be stated explicitly; the sequence of actions must be derivable from known theoretical foundations to guarantee that the derivation or synthesis of the control or controller is applicable to different controlled objects having different goals and/or different constraints.

Control **system:** a system that comprises at least two components: controlled object and controller. The controller delivers control signals or actions to the controlled object which force the controlled object to achieve a desired goal.

Control **of** software: here software serves as the controlled object and control **is** applied to the software. The actions taken may include the identification or estimation of internal information and parameters of software or software processes for determining whether or not the software functions properly or whether or not a software process will achieve its intended objective so that a controller may transform the software or modify appropriate parameters in a software process so that desired objectives are achieved on schedule. The control process must be quantifiable and hence repeatable under similar circumstances for similar controlled objects.

Open-loop control: control (the sequence of actions) is determined off-line without feedback; the responses of the controlled object to the applied actions are not utilized for determining subsequent actions taken on the controlled object.

Closed-loop control: By closed-loop control we mean that the behavior of the controlled object is monitored and used in an on-line feedback control strategy to enforce a desired objective, i.e., real time information is used to

determine subsequent control actions.

Feedback control: By feedback control we mean closed-loop control.

Adaptive control: an advanced form of feedback control; during the process of adaptive control, not only the actions to be taken are derived and updated on-line, but also the controller is updated on-line on the basis of the responses of the controlled object to actions taken already.

To motivate the area of software cybernetics, the question of why software needs to be and can be kept under control or why control of software is necessary and feasible needs to addressed. This can be analyzed from several different perspectives.

(1). Conventional or existing software engineering is a discipline of applying sound engineering principles to software development, maintenance and management. The ultimate goal of software engineering is to deliver quality software products timely in a cost-effective manner. Several feedback mechanisms and control activities can be observed in software systems and software engineering processes. In a fault-tolerant software system, the software execution process is continuously monitored to detect possible faults; actions are then taken to mask the effects of the occurring faults if any in order to minimize the occurrence of catastrophic software failures. This is a form of feedback control. In the software development process, various verification and validation activities such as inspection, review, testing and so on are taken to assess the work of software developers. The results are fed back to the software developers for improved performance, improved quality, etc. Feedback is ubiquitous in software engineering processes. Indeed, without such control, software and software processes would rarely achieve their specified goals.

(2). A major problem with current software engineering is that it is not sufficiently quantitative. Many ad hoc approaches and rules of thumb are extensively applied without rigorous justification or explanation based on accepted basic principles. Consequently, the success of complex software projects relies heavily on the experience of software personnel. Although the current trends of formalization and componentization have helped software engineering become more rigorous, the various feedback mechanisms and controls have not been fully identified or utilized in a systematic manner. In order for the software engineering process and system to be more quantitative, manageable and repeatable, various feedback mechanisms should be formalized, quantified and optimized. Therefore a feedback control theoretic context for software engineering would seem to be a natural evolution of the current state of the art.

(3). Although software is dramatically different from conventional controlled objects such as aircraft, chemical processes and fluid flows, existing control theories and principles can be applied. By treating the software test process as a controlled object and the process manager as a controller, the management of software testing becomes a feedback control problem. By treating the operating environment of the software under development as a controlled object, and the software being developed to be a controller, the synthesis of reactive software becomes a supervisory control problem. We will revisit this in Section 3.

(4). In the management of the software test process, the control theoretical approach can quantitatively forecast the test process trends and assist the manager in allocating testing resources [13]. In comparison with random testing, adaptive testing as a form of adaptive control uses less test cases to detect more software defects [14]. Hence, control theoretic approaches would appear to be necessary for optimizing the process.

The emerging area of software cybernetics can also be justified from the perspective of control science and engineering.

(1). The field of control science and engineering is applicationdriven and grew out of various application requirements [18]. The Watt governor drove people to stabilize systems of motion and investigate/develop general methods for testing and enforcing stabilization. This eventually led to the well known Routh-Hurwitz stability test among others. The popular PID (proportional-integralderivative) controller originated from the way in which a helmsman steered a ship. The power of the frequency response approach to the design of feedback systems was demonstrated in the Second World War to meet military requirements and objectives. For example, the anti-aircraft control problem led to the work of Nobert Wiener on filtering and prediction [19]. Wiener further published his celebrated book on cybernetics (or control and communication in animals and machines) in 1948 [20]. The problem of launching, maneuvering, guidance, and tracking of missiles and space vehicles was the major force that drove the establishment and proliferation of the state-space models and optimal control techniques, thereby establishing modem control theory. By analogy, it is reasonable to say that the control of software will become fertile ground for new advancements and techniques of control science and engineering.

(2). However software is dramatically different from traditional controlled objects. A characteristic feature of software is that software may contain various defects which can disrupt normal operations of software systems and lead to software failures in unexpected ways. Another distinct feature is that concurrency of various processes within a software system may dominate the behavior of the software system and make the behavior unpredictable

and non-repeatable even in a statistical sense. Then how does one represent or model software processes? Conventional transfer functions and state-space models may not be enough to represent software since they may be inappropriate to characterize the behavior of software defects and concurrency. Also, the ultimate goal of software engineering is to deliver quality software in a cost-effective and timely manner. Conventional control requirements such as stability, rise time and overshoot may no longer be appropriate. Software is a new application field with new representation models and new control requirements.

(3). Besides the classical control theory and modem control theory which uses transfer functions and state-space models to represent the controlled objects, the use of finite-state automata to represent discrete-event dynamic systems has lead to a new theory of control, the so-called supervisory-control theory [21]. The interconnections models and the T-S fuzzy models of controlled objects are other examples [22, 23]. We can expect that the new representation models of software will improve existing and/or lead to new control theories.

(4). New control requirements often lead to new research topics or theories. An example is the H2 and H-infinity performance criteria for robust control [24]. We also expect that the new control requirements of software may improve existing control theories and/or lead to new control theories.

(5). With the widespread application of control principles and theories to other fields such as biology and computer networking [25, 26], it is a logical step to apply control principles and theories to software processes. These new diverse applications will evolve the existing control theory in new and probably unexpected ways.

Last but not least, the emerging area of software cybernetics will undergo a similar evolution due to the challenges of controlling software processes. Modem aircraft adopt fly-by-wire flight control systems implemented with embedded software components and systems. In order to obtain satisfactory software control policies, system identification, adaptive and classical control theory and software reliability theory must be merged under the umbrella of software cybernetics.

3. Existing Research Topics

Many research areas have already benefited from the use of the concepts from software cybernetics although their relationship with software cybernetics has not yet been explicitly determined. Some of these contributions are described next. However the list below is not intended to be comprehensive and some works such as software process modeling with system dynamics [80] and proactive and autonomic computing [81] are not addressed here. In addition, fully workable examples using the tools of software cybernetics were reported in the cited references [13, 71] but are not reviewed here.

3.1. Feedback Mechanisms in Software Processes

Suppose that multiple versions of a software product are released to the market consecutively. Changes are made from one version to another to improve the underlying software functionality and quality. This constitutes a software evolution process. Then, two fundamental questions can be raised. Are there any invariant patterns in the software evolution processes? What role does feedback play in the various software evolution processes?

Lehman is one of the early researchers who paid attention to the role of feedback in software evolution processes. He classifies software programs as S-programs, P-programs, and Eprograms , and proposes the so-called laws of software evolution [27-29]. On the other hand, Basili proposed the concept of software experience factory [30]. Obviously, software experience is a form of feedback information and the software experience factory approach is actually feedback based.

The importance of the work of Lehman and Basili is that they initiated research on feedback mechanisms in software processes. Feedback is ubiquitous in software processes. On the one hand, feedback drives software processes to demonstrate some invariant laws, and on the other, appropriate feedback leads to better reuse of software experience and improves software processes. However we note that the work of Lehman and Basili does not formalize, quantify or optimize the underlying feedback mechanisms. The corresponding theoretical foundation is needed to interpret and improve the roles of feedback mechanisms in software processes.

3.2. Bisimulation and Controllability

Controllability is a fundamental notion in modern control theory [31]. Roughly speaking, a dynamic system is controllable if an arbitrary state of it can be moved into another arbitrary state in a finite length of time. Wonham extends the notion of state controllability to that of language controllability of finitsstate automata for discreteevent systems with uncontrollable events [21]. A language of a discrete-event system is controllable if the prefix closure is invariant under the occurrence of uncontrollable events. The controllability condition tests if a controller can be synthesized and connected to a given controlled object so that undesirable behavior is forbidden and desirable behavior is delivered. The controlled object and the entire system comprising the controlled object and the controller are, in some sense, equivalent with respect to controllability.

Bisimulation is a fundamental notion in concurrency theory and theoretical computer science [10]. Concurrent behavior can be observed extensively in distributed software and communication software. Bisimulation are partitions of the state space of software or computing system that preserves observability and reachability properties. Roughly speaking, bisimulation is a notion of property preserving abstraction for reducing the complexity of finite state systems. Bisimulation identifies the equivalence between a complex system and a simplifying one. Note that, as mentioned in the previous paragraph, the controllability problem can be treated as an equivalence problem in some sense. It is natural to ask if there are some inherent relationships between bisimulation and controllability.

Barrett & Lafortune show that a language of a finite-state automaton is controllable if and only if the finite-state automaton (without supervisor) and an automaton corresponding to the language can form a bisimulation relation [32]. Rutten introduces the notion of controllability relation and shows that the conventional notion of language controllability can be defined in terms of bisimulation or its variants in the setting of coalgebra [33]. In a series of papers [34-39], Pappas, et. al., systematically introduce the notion of bisimulation and the like to conventional dynamic systems.

The importance of the research topic of bisimulation and controllability is due to the fact that fundamental notions in two seemingly unrelated fields, computer/software science and control theory, can be formulated under a unified framework, and the theoretical tools in software science and engineering can be a powerful impetus for the development of control theories and techniques. However this research topic is still in its infancy and many questions remain open. For example, how can nonlinear systems and stochastic systems be formulated in the setting of bisimulation? How can the bisimulation relation of dynamic systems be identified and tested on-line?

3.3. Adaptive Software

The environments where software products are executing today have considerably increased in complexity. The number of simultaneous users on distinct platforms with different resource constraints and the dynamic interaction among all of these elements constitute the basis for this complex environment. A question arises from this scenario: "How can we design software to cope with such dynamic environments?" Adaptive softwareisystems address the solution for this problem by designing software that adapts itself according to changes in the environment. One approach to the design of adaptive systems has focused on resource allocation, such as an operating system scheduling processes, QoS (Quality of Service) guarantee, and fault tolerance [74-77].

Two of the adaptive software approaches are of interest here due to the use of control theory concepts: the SMART Framework [77] and the QoS framework [76]. Both approaches are based on control theory and on automatic tuning **of** the controller. SMART differs from the QoS framework by adjusting the system not based on resource allocation but on swapping to a component that provides the same functionality but better copes with the environment status. Also, SMART is a more generic approach by allowing the specification of the system constraints and their relation to the alternative choices.

3.4. Software Synthesis

Statistics indicate that over half of software defects are related to the software design phase [40]. Different from conventional informal methods or formal methods of software synthesis [40, 41], he control theoretical approach to software synthesis treats the operating environment as a given controlled object and the software under synthesis as the required controller [43]. The applicability of this approach is extended to more forms of non-functional requirements of state reachability and invariance in references [44]. On the other hand, Sridharan, Mathur & Cai show that the theory of supervisory control can well be applied to synthesize the safety controllers for ConnectedSpaces which is a collection of one or more devices, each described by its Digital Device Manual and reachable over a network [45, 46].

3.5. Software Test Process Control

Under or overestimation of time and cost is common in the software test process. In many companies the estimation depends only on the manager's experience. Furthermore, the evaluation of the progress of the test process is as inaccurate as the initial estimation. Measurements such as reliability and coverage can help access the progress of a test process. But in practice, just a small number of companies collect such information. Even when progress can be accessed and a deviation from expected values is detected, no techniques are used to determine the required changes in the process to correct the observed deviation. Techniques such as System Dynamics [SO] and Software Process Simulation provide an open loop solution for this problem by answering "What if?" questions.

A new technique based on control theory has been developed to provide a closed loop solution for the above problem. A set of assumptions and corresponding equations are combined leading to a state space model. The availability of the model allows the application of

feedback control and once a deviation from the expected behavior is detected, a set of alternative solutions is computed to correct for such deviation. The model is also used to predict the behavior of the process. The prediction is based on a calibration algorithm using data from the ongoing process. In any case, as stated above, the model is used to compute the required changes in the process to fulfill the manager's expectations [13].

The model mentioned above has been statically and dynamically validated. A tensor product based sensitivity analysis and an extremal case analysis were used for the static validation of the model. The first was also helpful in pointing out flaws in an early version of the model guiding to the current version of the model [77]. With regard to the dynamic validation, the model has been successfully applied to a variety of software projects. The predictions of future behavior as well as the estimate of the initial number of defects were, on average, around 80% accurate.

3.6. The CMC Approach and Adaptive Testing

Conventional software testing techniques are mainly defined a priori and do not address how to formalize, quantify or optimize the feedback mechanisms in software testing. To address this question, Cai treats software testing as a control problem and proposes the CMC (Controlled Markov Chain) approach to software testing [14]. The software under test serves as the controlled object and is modeled as a controlled Markov chain, where the software testing strategy serves as the corresponding controller. The software state is defined as the number of remaining software defects. Suppose the related software parameters such as the initial number of software defects and software defect detection rates are known, then an optimal testing strategy can be derived on the top of existing control theory of Markov chains to achieve a given testing goal. An example goal is to remo ves all the remaining defects in the least number of tests.

In practice the required software parameters are unknown and mu st be estimated by using test data. This leads to adaptive (software) testing. Adaptive testing adjusts the testing strategy or improves the testing technique on-line by learning from the history of software testing or other sources and is a form of adaptive control. Studies demonstrate that adaptive testing can significantly outperform the random testing [14]. It uses fewer test cases to detect more defects. Further, the variance of the number of test cases for detecting a given number of defects is reduced in comparison with the random testing. The CMC approach and adaptive testing also apply to the optimal stopping problem [52], software testing with testing resource constraints [53] and optimal software reliability assessment [54]. A case study with the Space program shows that adaptive testing can significantly outperform random testing [71].

Here we note that Chen et. al. propose the so-called "adaptive random testing" to take account of the patterns of failure-causing inputs for guiding the test case selections [55, 56]. Following the definitions described in Section 2, we see that " adaptive software testing" is a form of more feedback control than adaptive control.

4. Potential Research Topics

Besides those identified by Cai, Chen, and Tse including the relationships among controllability, observability and testability [16], there are other potential research topics.

4.1. Adaptive Software Rejuvenation and Software Performance Control

Traditionally software systems are treated as different from hardware systems due to their non-aging nature. However recent research reveals that software may age or degrade and that the phenomenon of software aging should not be ignored [57-59]. This is particularly true for the software in a networking environment, where software aging may lead to hang or crash of servers. The underlying causes of software aging include memory leaks, unreleased file locks, accumulation of un-terminated threads, data corruptionhound-off accrual, file-space fragmentation, shared memory pool latching, and others. An effective approach to dealing with software aging is to rejuvenate the software system of concern periodically or at desirable instants of time before software aging leads to hang or crash of software service. To this end, the current status of software operation is monitored and necessary information is collected to decide when and what parts of the software should be rejuvenated. This is essentially a control problem of software and provides avenues for control theories and techniques to play a role. However most of existing researches on software rejuvenation lacks a solid control foundation [60, 61]. Can software rejuvenation be treated as a control problem? How can software be rejuvenated adaptively?

On the other hand, the performance and quality of service (QoS) is important for network software. The network should adjust its operation and service scenarios in accordance with the changes in network traffic flow. This is again a control problem of software and a largely unexplored research topic, although there have been some researches in this area [62].

4.2. Control Theoretical Approach to Software Fault Tolerance

Several approaches have been proposed for software fault tolerance, including N-version programming, recovery block programming, Self-checking Nversion programming, checkpointing programming, and others [63]. Unfortunately existing software fault tolerance schemes are rarely related to control theories and techniques and lack a solid theoretical foundation. Treating software fault tolerance as a control problem is a promising idea and may lead to new vitas and approaches for software fault tolerance with theoretically sound foundation.

4.3. Logical Foundation for Control Systems

In a recent report on "future directions in control, dynamics, and systems" written by a group of control scientists [64], control was defined to be "the use of algorithms and feedback in engineered systems". A natural question is that if a logical foundation is necessary or feasible for control systems. The work of Vassilyev addresses the potential of applying temporal or higher-order logic to control system synthesis [65]. Partially related to the logical foundation for control systems are logics for hybrid systems [66].

4.4. Community Complexity in Control Systems

Community complexity theory is an active research direction in software community and theoretical computer science [70]. It is concerned with how much communication is necessary for collaborative agents to complete a given task. A simple example is that two processors collaborate with each other to compute a given value and there must be a minimum number of communications for the computation process to obtain a right answer.

We can treat the controlled object and the controller in a control system as two collaborative agents to perform a control task. Then how much communication or feedback is necessary or sufficient for performing the control task? Is there any Emitation or impossibility for feedback control to perform a given control task? Questions of this kind have been studied in control community from a purely control perspective [67-69]. Can they be related to community complexity theory [70]?

5. On-going Research Projects

Among many institutions where the concepts of software cybernetics are being applied, below is a summary of the projects related to the author's organizations.

5.1. Purdue On-going Research Projects

An initial model was developed to predict schedule slippage during the system test process. This model, and the associated feedback controller, is used to determine the space of possible changes to be made to the process parameters in order to ensure that the schedule will be met. The model has been applied in two commercial environments with encouraging results [13]. A more elaborate model of the test process, that captures the flow of various events in the process starting from test generation during the requirements phase, has been developed. The model is currently undergoing validation studies.

5.2. UTD On-going Research Projects

Two major projects under the perspective of software cybernetics are been developed at the Computer Science Department at the University of Texas at Dallas.

(1). Stochastic software process control: Under many circumstances, a stochastic rather than a deterministic model appears to be a better solution to represent the process. The characterization of the disturbances in the stochastic model is done according to the CMM level of the organization. Two noise sequences are inserted into the model representing unforeseen perturbations and noise in the data collection process. The effect of the later can be minimized by the use of a Kalman filter. Finally a feedback software control tool to hide the complexity of the models from software managers will be made available through a web-server.

(2). Adaptive software: The SMART Framework for adaptive software dynamically tune the controller in adaptive software using system identification techniques such as the least square approach and Markov parameters. In addition to the advantages of any adaptive approach, the SMART framework presents two features that distinguish it from a multitude of other techniques: flexibility and predictability.

5.3. Beijing On-going Research Projects

Several on-going research projects are carried out in the research group in Beijing.

(1). Adaptive testing with fixed-memory feedback. A new adaptive testing scheme is proposed [71, 72]. In this new scheme only a fixed amount of latest test data is fed back to on-line parameter estimation for adaptive testing.

(2). Adaptive mutation testing. Research has demonstrated that the mutation testing can be formulated in the setting of the CMC approach and adaptive testing

[73]. This leads to reduction of computational burden of mutation testing.

(3). Control theoretical approach to software architecture synthesis. Section **3.4** explains the rationale of applying control theories to the software synthesis. An on-going research project is devoted to taking into account of architectures for software synthesis.

(4). Control theoretical approach to software fault tolerance. Existing fault-tolerant software is largely based on the idea of code redundancy. An on-going project is devoted to develop a control-theoretical approach to software fault tolerance without code redundancy [79].

6. Concluding Remarks

The interplay between control and software has been the focus of this paper. Pointing to the early work of Lehman and Basili that is related to software cybernetics in some sense, we have argued the need for comprehensive research to formalize and quantify this interplay. A large number of problems that arise during the development of software and its operation have been pointed out and the possibility of using control theoretic approaches to solve these problems has been raised. It is our hope that the arguments and discussion presented here will encourage researchers from both the software and the control communities to come together and work towards the establishment of a solid foundation that can be used in practice for effective and efficient development of high quality software. The development of software cybernetics should also give rise to new control theories.

References

[1]. J.E.Hopcroft, R.Motwani, J.D.Ullman, ***Introduction to Automata Theory, Languages, and Computation (Second Edition),*** Addison-Wesley, 2001.

[2]. R.S.Pressman, ***Software Engineering: A Practitioner's Approach (Fifth Edition),*** McGraw-Hill, 2001.

[3]. A.M.Langer, ***Applied Ecommerce: Analysis and Engineering for Ecommerce Systems,*** John Wiley & Sons, 2001.

[4]. E.G.Guimares, A.T.Maffeis, R.P.Pinto, C.A.Miglinski, E.Cardozo, M.Bergerman, M.F.Magalhaes, " REAL – A Virtual Laboratory Built from Software Components", ***Proceedings of the IEEE,*** Vol.91, No.3, March 2003, pp440-448.

[5]. H.B.Newman, M.H.Ellisman, J.A.Orcutt, " Data-Intensive E-Science Frontier Research" , ***Communications of the ACM,*** Vol.46, No.11, November 2003, pp68-77.

[6]. E.Rieffel, " An Introduction to Quantum Computing for Non-Physicists",***ACM Computing Surveys,*** Vol.32, No.3, pp300-335.

[7]. V.Firoiu, J.Y.L.Boudec, D.Towsley, Z.L.Zhang, " Theories and Models for Internet Quality of Service", ***Proc. IEEE,*** Vol.90, No.9, 2002, pp1565-1591.

[8]. K.Y.Cai, L.Chen, " Analyzing Software Science Data with Partial Repeatability", ***Journal of Systems and Software,*** Vol.63, 2002, pp173-186.

[9]. C.A.R.Hoare, ***Communicating Sequential Processes,*** Rentice-Hall, 1985.

[10]. R.Milner, ***Communication and Concurrency,*** Prentice-Hall, 1989.

[11]. B.Littlewood, L.Strigini, " Validation of Ultrahigh Dependability for Software-Based Systems", ***Communications of the ACM,*** Vol.36, No.11, 1993.

[12]. J,Sztipanovits, G.Karsai, " Self-Adaptive Software for Signal Processing", ***Communications of the ACM,*** Vol.41, No.5, 1998.

[13]. 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.

[14]. 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.

[15]. 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.

[16]. 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.

[17]. K.Y.Cai, J.W.Cangussu, R.A.DeCarlo, A.P.Mathur, S.Miller, " Feedback and Adaptive Control for Software Testing" , working paper, 2003.

[18]. S.Bennett, " A Brief History of Automatic Control" , ***IEEE Control Systems Magazine,*** June 1996, pp17-25.

[19]. N.Wiener, ***Extrapolation. Interpolation and Smoothing of Stationary Time Series with Engineering Applications,*** MIT Press, 1949.

[20]. N.Wiener, ***Cybernetics: or Control and Communication in the Animal and the Machine,*** John Wiley, 1948.

[21]. P.J.Ramadge, W.M.Wonham, " The Control of Discrete Event Systems",***Proc. IEEE,*** Vol.77, 1989, pp81-98.

[22]. J.C.Willems, " On Interconnections, Control, and Feedback" , ***IEEE Transactions on Automatic Control,*** V01.42, No.3, 1997, pp326-339.

[23]. L.X.Wang, ***A Course in Fuzzy Systems and Control,*** Prentice-Hall, 1997.

[24]. K.Zhou, J.C.Doyle, K.Glover, ***Robust and Optimal Control,*** Prentice-Hall, 1996.

[25]. B.K.Ghosh, J.He, " Dynamics and Control Problems in Biology: Some New Challenges", ***IEEE Control Systems Magazine,*** August 2001, p27.

[26]. P.R.Kumar, "New Technological Vistas for Systems and Control: The Example of Wireless Networks", ***IEEE Control Systems Magazine,*** February 2001, pp24-37.

[27]. M.M.Lehman, " Software Engineering, the Software Process and their Support",***Software Engineering Journal,*** VOL6, No.5, 1991, pp243-258.

[28]. M.M.Lehman, " Laws of Software Evolution Revisited . In: ***Software Process Technology,*** (C.Montangero ed.), Springer-Verlag, 1996, pp108-124.

[29]. M.M.Lehman, " Process Modeling – Where next?", ***Proceedings of International Conference on Software***

0-7695-2218-1/04 $20.00 © 2004 **IEEE**

Engineering, 1997, pp 549-552.
[30]. Basili, V.R., C. Caldiera and H.D. Rombach, "The Experience Factory," in J.J. Marciniak (ed), ***Encylopedia of Software Engineering,*** Vol.1, pp 469-476. John Wiley & Sons, 1994.
[31]. C.T.Chen, ***Linear System Theory and Design.*** CBS College Publishing, 1984.
[32]. G.Barrett, S.Lafortune, " Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines", ***Discrete Event Dynamic Systems: Theory and Applications,*** Vol.8, 1998, pp377-429.
[33]. J.J.M.M.Rutten, "Coalgebra, Concurrency, and Control", CWI, SEN-R9921, 1999.
[34] R.Alur, T.Henzinger, G.Lafferriere, G.J.Pappas, " Discrete Abstraction of Hybrid Systems", ***Proc. IEEE,*** Vol.88, No.7, pp971-984.
[35]. G.J.Pappas, G.Lafferriere, S.Sastry, " Hierarchically Consistent Control Systems", ***IEEE Transactions on Automatic Control,*** Vol.45, No.6, 2000, pp1144-1160.
[36]. G.J.Pappas, G.Lafferriere, " Hierarchies of Stabilizability Preserving Linear Systems", ***Proc. the 40th IEEE Conference on Decision and Control,*** 2001, pp2081-2086.
[37]. G.J.Pappas, S.Simic, " Consistent Abstractions of Affine Control Systems", ***IEEE Transactions on Automatic Control,*** Vol.47, No.5, 2002, pp745-756.
[38]. Haghverdi, P.Tabuada, G.J.Pappas, " Bisimulation Relations for Dynamical and Control Systems", ***Electronic Notes in Theoretical Computer Science,*** Vol.69, Elsevier, 2003.
[39]. G.J.Pappas, " Bisimilar Linear Systems", ***Automatica,*** Vol.39, 2003, pp2035-2047.
[40]. A.M.Neufelder, ***Ensuring Software Reliability,*** Marcel Dekker, 1993.
[41]. E.M.Clarke, J.M.Wing, "Formal Methods: State of the Art and Future Directions", ***ACM Computing Surveys,*** Vol.28, 1996, pp626-643.
[42]. E.Clarke, O.Grumberg, D.Peled, ***Model Checking,*** MIT Press, 2001.
[43]. 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.
[44]. 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.
[45]. B.Sridharan, A.P.Mathur, K.Y.Cai, " Using Supervisory Control to Synthesize Safety Controllers for ConnnectedSpaces", ***Proc. the 3rd International Conference on Quality Software,*** IEEE Computer Society Press, 2003, pp186-193.
[46]. 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.
[47]. B.Beizer, *Software Testing Techniques (2nd Edition)*, Van Nostrand Reinhold, 1990.
[48]. H.Zhu, P.A.Hall, J.H.R.May, " Test Coverage and Adequacy", ***ACM Computing Survey,*** Vol.29, No.4, 1997, pp366-427.
[49]. T.Y.Chen, Y.T.Yu, " On the Expected Number of Failures Detected by Subdomain Testing and Random Testing", ***IEEE Transactions on Software Engineering,*** Vol.22, No.2, 1996, pp109-119.
[50]. P.G.Frankl, E.J.Weyuker, " A Formal Analysis of the Fault-Detecting Ability of Testing Methods", ***IEEE Transactions on Software Engineering,*** **Vol.**19, No.2, 1993, pp202-213.
[51]. W.J.Gutjahr, " Partition Testing vs. Random Testing: The Influence of Uncertainty", ***IEEE Transactions on Software Engineering,*** Vol.25, No.5, 1999, pp661-674.
[52]. 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.
[53]. 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.
[54]. 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.
[55]. 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.
[56]. T.Y.Chen, F.C.Kuo, R.G.Merkel, S.P.Ng, " Mirror Adaptive Random Testing", ***Proc. the 3rd International Conference on Quality Software,*** IEEE Computer Society Press, 2003, PP4-9.
[57]. Y.Huang, C.Kintala, N.Kolettis, N.D.Fulton, " Software Rejuvenation: Analysis, Module, and Applications", ***Proc. The 25th International Symposium on Fault-Tolerant Computing,*** 1995, pp381-390.
[58]. A.Avritzer, E.J.Weyuker, " Monitoring Smoothly Degrading Systems for Increased Dependability", ***Empirical Software Engineering,*** Vol.2, No.1, 1997, pp59-77.
[59]. V.Castelli, R.E.Harper, P.Heidelberger, S.W.Hunter, K.S.Trivedi, K.Vaidyanathan, W.P.Zeggert, " Proactive Management of Software Aging", ***IBM Journal of Research and Development,*** Vol.45, No.2, 2001.
[60]. Y.Bao, X.Sun, K.S.Trivedi, " Adaptive Software Rejuvenation: Degradation Model and Rejuvenation Scheme", ***Proc. the 2003 International Conference on Dependable Systems and Networks,*** 2003.
[61]. A.Pfening, S.Garg, A.Puliafito, M.Telek, K.S.Trivedi, " Optimal **Software** Rejuvenation for Tolerating Soft Failures", ***Performance Evaluation,*** Vol.27&28, 1996, pp491-506.
[62]. T.F.Abdelzaher, J.A.Stankovic, C.Lu, R.Zhang, Y.Lu, "Feedback Performance Control in Software Services", ***IEEE Control Systems Magazine,*** June 2003, pp74-90.
[63]. M.R.Lyu (ed), ***Software Fault-Tolerance,*** John Wiley & Sons, 1995.
[64]. R.M.Murray, et al, " Control in an Information Rich World", http://www.cds.caltech.edu/~murray/cdspanel/, 2002.
[65]. S.N.Vassilyev, " Logical Approach to Control Theory and Applications", ***Nonlinear Analysis, Methods and Applications,*** Vol.30, No.4, 1997, pp1927-1937.
[66]. J.M.Davoren, A.Nerode, " Logics for Hybrid Systems",

froc. IEEE, Vol.88, No.7, 2000, pp985-1010.
[67]. M.M.Seron, J.H.Braslavsky, G.C.Goodwin, ***Fundamental Limitations in Filtering and Control,*** Springer, 1997.
[68]. L.L.Xie, L.Guo, " How much Uncertainty can be Dealt with by Feedback?', ***IEEE Transactions on Automatic Control,*** Vol.45, No.12, 2000, pp2203-2217.
[69]. J.Chen, R.H.Middleton (eds), " Special Issue on New Developments and Applications in Performance Limitation of Feedback Control", ***IEEE Transactions on Automatic Control,*** Vol.48, No.8, 2003.
[70]. E.Kushilevitz, N.Nisan, ***Communication Complexity***, Cambridge University Press, 1997.
[71]. K.Y.Cai, B.Gu, H,Hu, Y.C.Li, " Adaptive Software Testing with Fixed-Memory Feedback- Extended Abstract", ***Proc. COMPSAC the Firs! International Workshop on Software Cybernetics,*** Hong Kong, September 28-30, 2004, IEEE Computer Society Press.
[72]. K.Y.Cai, B.Gu, H.Hu, Y.C.Li, " Behavior of an Adaptive Software Testing Strategy with Respect to the Amount of Feedback Information", working paper, 2003.
[73]. K.Y.Cai, B.Gu, " Adaptive Mutation Testing" , in preparation, 2004.
[74]. T. F. Abdelzaher, " An Automated Profile Subsystem for QoS-aware Services", IEEE Real-Time Technology and Applications Symposium, Washington, DC, June 2000.
[75]. T.F.Abdelzaher, N.Bhatti, " Web Sewer QoS Management by Adaptive Content Delivery", ***froc. International Workshop on Quality of Service,*** London, ***UK,*** June 1999.
[76]. Y.Lu, T.F.Abdelzaher, C.Lu, G.Tao, " ***An*** Adaptive Control Framework for QoS Guarantees and its Application to Differential Caching Services", ***froc. 10th International Workshop on Quality of Service,*** pages 23-32,2002.
[77]. J.W.Cangussu, K.Cooper, C.Li, -- A Control Theory Based Framework for Dynamic Adaptable Systems", ***Proc. 19th ACM Symposium on Applied Computing***, Nicosia, Cyprus, March 14-17, 2004.
[78]. 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. 28, no. ***5,*** pages 430-443, May 2003.
[79]. K.Y.Cai, X.Y.Wang, " Towards a Control-Theoretical Approach to Software Fault-Tolerance", ***froc. 4th International Conference on Quality Software,*** IEEE Computer Society Press, 2004.
[80]. B.W.Boehm, R.J.Madachy, ***Software Process Modeling with System Dynamics,*** John Wiley & Sons, 2003.
[81]. R.Want, T.Pering, D.Tennenhouse, -- Comparing Autonomic and Proactive Computing", ***IBM Systems Journal,*** Vol.42, No.1, 2003, pp129-135.