Software Cybernetics

João W. Cangussu
Department of Computer Science
University of Texas at Dallas
cangussu@utdallas.edu

Scott D. Miller
Department of Computer Sciences
Purdue University
millersd@cs.purdue.edu

Kai-Yuan Cai
Department of Automatic Control
Beijing University of Aeronautics and Astronautics
kycai@buaa.edu.cn

Aditya P. Mathur
Department of Computer Sciences
Purdue University
apm@cs.purdue.edu

Abstract

Software Cybernetics explores the interplay between software and control. Though the concepts of software and cybernetics are well known when considered in isolation, the definition and scope of Software Cybernetics is sometimes blurred due to its novelty. A definition of Software Cybernetics and the delineation of its scope are the major goals of this article. Also presented here is a brief description of some applications of Software Cybernetics.

1 Introduction

Separately, the concepts of software and cybernetics are well known and found as in the definitions extracted from an on-line dictionary.

> **Software [1]**: The programs and procedures required to enable a computer to perform a specific task, as opposed to the physical components of the system (hardware).

> **Cybernetics [1]**: The study of communication and control, typically involving regulatory feedback, in living organisms, in machines and organizations and their combinations, for example, in sociotechnical systems, computer controlled machines such as automata and robots.

As we can see from the definition above, Cybernetics includes software that implements a control system but does not include the control of software itself, much less the control of the software development process. For example, the design of the software for a cruise control system of an automobile is considered part of Cybernetics but the design of a software/technique to regulate the behavior of another software system is not addressed in the scope of Cybernetics. A new area is therefore needed to develop control systems with this purpose. It should be noted that the definition of control systems (as seen below) used in this new area remains unchanged.

> **Control Systems** [1]: A control system is a device or set of devices that manage the behavior of other devices. Some devices or systems are not controllable. A control system is an interconnection of components connected or related in such a manner as to command, direct, or regulate itself or another system. A control loop is a collection of instruments and control algorithms arranged in such a fashion as to regulate a variable at a setpoint or reference value. The loop part of the name refers to the fact that most control loops make use of feedback in a continuous loop. These are referred to as closed loop control. There is also open-loop controller that does not directly make use of feedback. The most common control loop uses a feedback or PID controller.

Control theory has been successfully applied to solve problems in areas such as biology, management, and social sciences, among others. The successful application of the same concepts to control/regulate the behavior of software systems and/or of the software development process [54] in all its aspects is what we now refer to as Software Cybernetics. Software Cybernetics also includes principles and

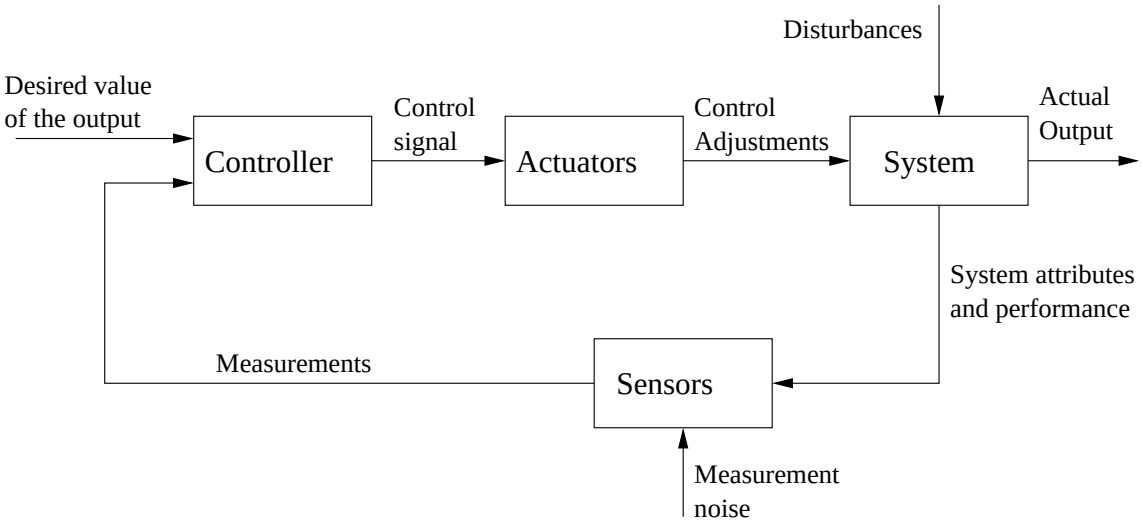

Figure 1. Typical feedback loop [33]

theories in software engineering that can be applied to control engineering. In this article we provide a definition of Software Cybernetics and delineate its scope. Research in Software Cybernetics has been applied to many distinct areas such as software development, adaptive software, network security, fault tolerance, etc.; a brief description of these applications appears in Section 3.

Software Cybernetics has been consistently expanding since its inception and the community is organizing itself mainly through the International Workshop on Software Cybernetics (IWSC). However, skepticism continues to exist as some reject the feasibility/utility of regulating software systems or its development process by mathematical laws. Also, there exists a belief that control is purely a continuous approach and overlooks the fact that there do exist discrete counterparts for all continuous techniques. Moreover, even when a continuous approach is used to model the behavior of the object under consideration, a control technique can be applied at discrete time intervals. The difficulties in applying feedback control to software processes, for example, have been delineated by Lehman, Perry, and Turski [43]. One of the difficulties they point out is the immaturity of software processes and they state "... we need research to establish appropriate theories from which to derive necessary control mechanisms and experimentations to establish their settings and effects". Although this has not yet been fully accomplished, results from research on Software Cybernetics have moved a few steps towards this goal. Another aspect contributing to the slow development and adoption of control-theoretic concepts for software is the paucity of control-system researchers involved in software engineering. Again, Software Cybernetics offers an environment where collaboration among both areas can flourish.

2 Definition and Scope

According to Norbert Wiener [72], cybernetics refers to control and communication in the man and the machine. Accordingly, one may define software cybernetics as control and communication in the software, its processes, and systems. However this definition is not satisfactory for two reasons. First, it implies that software cybernetics should cover various ad hoc control activities in software engineering and various communication mechanisms in concurrent computing. It would be difficult to distinguish software cybernetics from existing discipline of software engineering and the theories of concurrent computing. Second, the above definition rules out the possibility that the principles and theories of software engineering can be applied to control theories and engineering.

It is important to note that control is a well-established discipline, of which feedback and optimization are two central themes. Further, a solid theoretical foundation has yet to be established for the existing discipline of software engineering. Therefore, we define software cybernetics as an emerging discipline that explores the *theoretically justified* interplay between software and control. More specially, according to Cai et al. [11], software cybernetics addresses issues and questions that relate to: (i)-the formalization and quantification of feedback mechanisms in software processes and systems; (ii)-the adaptation of control theoretic 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. In response to these issues and questions, research sub-areas of software cybernetics can be divided into four

classes: fundamental principles, cybernetic software engineering, cybernetic autonomic computing, and software-enabled control.

Figure 1 presents the structure of a typical closed feedback control loop [33]. Though it is oriented towards physical systems, the same structure can be used to control software systems and software processes. The block labeled "System" in Figure 1 can be mapped to a software system or software process as the object to be controlled and the actuators could be exemplified as the process manager executing suggested changes (in case the object to be controlled is a software process), or the operating system allocating additional memory when the object under control is a software application. Clearly, the application of control requires quantification/qualification of the output variables to be controlled and the input values used to control them. This means that any measurable quantities/qualities with respect to software products and processes can be potentially controllable. For example, consider the control system proposed for the software system test phase by Cangussu et al. [21]; the controller uses the model parameters, and two inputs corresponding to the test manager's vectors for inducing changes in the process. The control technique explicity requires that the user be able to quantify the values of the control inputs, and implicitly requires–through the use of the the model parameters–the data from which the model parameters were calibrated. This example illustrates that the data requirements of a particular application of software cybernetics are primarily dictated by the underlying model, with a small additional requirement in the specification of the expected model inputs.

It should be clear that Software Cybernetics is not the only overlap of software systems/software engineering with control theory. The implementation of a controller for a boiler system represents this overlap (software being used to implement a control mechanism). However, Software Cybernetics is more comprehensive and can be seen more as the mapping of software systems/software engineering concepts to concepts in control systems. For example, the use of control theory to regulate the amount of memory reserved for a cache system represents this mapping.

2.1 Fundamental Principles

The research area of Fundamental Principles (FP) is concerned with the fundamental questions and theoretical foundation of software cybernetics. Such questions include: Can the software behavior be controlled? What role can feedback play in software processes and systems? How can software behavior be modeled in the framework of software cybernetics? Three specific topics should be addressed in this research area: modeling formalism, controllability and bisimulation, and feedback complexity and bisimulation.

2.1.1 Modeling Formalism

To address the question of whether software behavior can be controlled in a theoretically justified manner, it is important to examine how software behavior can be modeled. Three modeling formalisms have been proposed: formal models, dynamical system models, and controlled Markov chains. A typical formal model is the extended finite state machine (EFSM), which has been widely adopted to describe communication software behavior [3]. It is interesting to note that an EFSM can be reformulated as a closed-loop control system comprising a controlled object and a controller [57]. This implies that most software systems can be modeled as a control system.

Linear dynamic system models have been proposed to describe software testing process [21] and software service behavior [67]. In the continuous-time domain, the model is in the form of Eq. 1.

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \quad (1)$$

where $x(t)$, $u(t)$, and $y(t)$ are state vector, control input, and output, respectively, and A, B, C, and D are matrices of appropriate dimension. The discrete-time counterpart of Eq. 1 is given by Eq. 2

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases} \quad (2)$$

where k denotes the k^{th} sampling instant.

Finally, controlled Markov chains have been proposed to describe the software testing process [15]. The states in a controlled Markov chain are defined by software variables of interest such as the number of defects remaining in the software under test. As distinct testing actions are applied to the software under test, the software state transitions take place in accordance with a Markov law whose probability distributions depend on the applied testing actions.

2.1.2 Controllability and Bisimulation

Controllability is a fundamental concept in modern theories of control. Generally, a dynamic system is said to be controllable if there is a control input that transforms the system from an arbitrary state to the zero state in a finite length of time [24]. Further, a (formal) language of a discrete-event system is said to be controllable if the prefix closure is invariant under the occurrence of uncontrollable events [58]. On the other hand, bisimulation is a fundamental concept in process algebra and concurrent computing [52]. It determines if two processes in a computing system or two states in a state space are equivalent in some sense of actions and state transitions. Three classes of research have been delineated within the topic of controllability and bisimulation.

The first class reveals that there are inherent relationships between controllability and bisimulation [7, 59]. This is surprising and supports the suggestion that control theories and computing theories may be put into a unified theoretical framework.

The second class of research is devoted to the introduction of bisimulation relations for conventional dynamic systems [68]. It is shown that the abstract notion of bisimulation in the context of open maps may characterize the equivalence relations for discrete event systems, for conventional dynamic systems, as well as for hybrid systems [37, 40]. The third class of research is concerned with how to control a system so that the resultant closed-loop system bisimulates or approximately bisimulates another system in some sense [74].

2.1.3 Feedback Complexity and Limitation

It is well-known in the control community that feedback is not a panacea for achieving control goals. There are limitations on the role that feedback can play [62]. Normally, a closed-loop control system comprises a controller and a controlled object, which communicate with each other in a collaborative manner to achieve a given control objective. Feedback between the controller and the controlled object defines a kind of communication. On the other hand, the communication complexity theory is a well-established area in computer science, which is concerned with why communication is necessary for two collaborative agents to complete a given task [42]. A natural question arises: Can feedback limitation be formulated in the context of communication complexity theory ? This research topic remains largely unexplored.

2.2 Cybernetic Software Engineering

Cybernetic software engineering (CSE) treats software development as a control problem and applies control theoretic principles to guide software process improvements and quality assurance [12]. Since a software development process is often divided into several phases such as requirement analysis, design, and testing, control theoretic principles are applied to individual phases.

2.2.1 Software Requirement Acquisition

Software requirement acquisition is an interactive process between the software development personnel and the user, and feedback is an intrinsic feature of the process. It is argued that the requirement acquisition process be treated as feedback Requirements Process Control (RPC) system, where the requirements specification of the application serves as the object to be controlled [73]. The RPC perspective can then applied to assess the quality of the requirement acquisition process and to guide the corresponding process improvement. This research area is in its early stages of development.

2.2.2 Software Synthesis

Since most software systems can be treated as a control system, it is natural to raise the question whether control theories can help guarantee the correctness of software design solutions. A modest amount of research has been devoted to address this question where the control theories of discrete-event systems are applied. In the work of Marchand and Samaan, and Wang et al. [48, 70] the software under synthesis, modeled by a polynomial dynamical system (PDS) serves as the required controller, whereas the operating environment serves as the controlled object. Sridharan et al. [66, 65] applies the theory of supervisory control 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. On the other hand, it is shown that software fault-tolerance can be treated as a robust supervisory control problem and the traditional idea of diverse redundancy can be avoided [14].

2.2.3 Software Test Management

A control mechanism has been used to regulate the behavior of the system test process. A mathematical model capturing the dominant behavior of the process has been developed [21]. The parameters of the model are calibrated using data from the ongoing process by means of a least squares approach. A parametric control approach is then used to regulate the process and correct problems such as schedule slippage. The approach has been statically validated using a Kronecker product to conduct a sensitivity analysis [22] and has also been successfully applied to a series of projects from large corporations [20].

2.2.4 Adaptive Testing

Adaptive testing is the software testing counterpart to adaptive control and is the outgrowth of the controlled Markov chain (CMC) approach to software testing. In the CMC approach, the software under test serves as the controlled object and is modeled by a controlled Markov chain, whereas the test strategy serves as the corresponding controller. The software under test and the test strategy make up a closed-loop feedback control system. While the test strategy uses the test history to generate or select the next set of test cases to be applied to the software under test, adaptive testing implies that the underlying parameters in the test strategy be also updated on-line during testing by using the test history. It is shown that adaptive testing can not only be applied for software reliability improvements by removing the detected

defects [15, 17], but also for software reliability assessment by freezing the code of the software under test [13].

2.3 Cybernetic Autonomic Computing

Autonomic computing (referred later in Section 3 as *CAC*) is an initiative launched by IBM [2]. It is aimed at making computing systems self-managing, which implies that computing should be self-configuring, self-optimizing, self-healing, and self-protecting. By cybernetic automatic computing we refer to autonomic computing achieved by applying control-theoretic or cybernetic principles and methods. Moreover, computing systems are treated as a feedback closed-loop control system and thus should be self-stabilizing. Several research topics have been addressed in the literature as follows.

2.3.1 Software-Aging Control

Aging is a phenomena widely studied in hardware reliability community. Hardware systems tend to age due to physical deterioration such that the corresponding failure rate function behaves as a non-decreasing function of the operating time. In 1990s it was observed that software systems also suffered from the aging phenomena [39, 49]. Software aging emerges as a result of computing resource contention, memory leakage, file-space fragmentation, and so on. It causes the computing systems to demonstrate performance degradation and then to hang, panic, and crash. This is particularly true for Web servers in the Internet environment. Three questions can be raised for software aging. First, what are the underlying aging mechanisms [36, 63] ? Second, how can software aging be modeled [32, 45] ? Finally, how can software aging be forecasted and controlled [28, 6] ? Software-aging control is an interesting research area deserving of additional investigation.

2.3.2 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 and often open environment such as the Internet. Adaptive software operating in the environment of this kind identifies the changes in the environment, adjust its internal architecture and/or parameters, and assesses its behavior to provide satisfactory quality of service (QoS). Feedback is an imperative kind of activity for adaptive software and a crucial problem is how to formulate, quantify and optimize the underlying feedback mechanism. Three questions have been partially addressed in the literature. First, what architectures should be adopted for adaptive software [25, 30] ? Second, what control algorithms should be developed for adaptive software [56, 26, 19] ? Finally, what quality attributes should be employed to assess adaptive software [46] ? Adaptive software is a research topic in which control-theoretic principles and methods can play a fundamental role.

2.3.3 Self-Stabilizing Software

The notion of self-stabilization was introduced by Dijkstra in 1974 [27] and a many self-stabilizing computing algorithms has been reported in the literature [61, 29]. A system is said to be self-stabilizing if, regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps. This implies that self-stabilizing software can resume normal operation in the presence of transient software faults. Although it is observed that self-stabilization shares some concepts with self-management in autonomic computing [38], research on self-stabilizing software is still in early stages. This topic is important for two reasons. First, existing mainstream mechanisms for software fault-tolerance lack solid theoretical foundation [14] and self-stabilizing software may be considered as a new kind of fault-tolerant software that is theoretically justified. Second, the notion of self-stabilization is different from the traditional notion of Lyapunov's stability, and self-stabilizing software may lead to a new theory of stability.

2.3.4 Autonomic Computing Prototyping

While autonomic computing systems are claimed to mimic the autonomic neurons in biological systems [31, 41], the field of autonomic computing does not currently provide researchers with a clear idea of what is required to develop an autonomic computing system [46]. What are the fundamental concepts and principles of autonomic computing? What are the qualitative and quantitative goals of autonomic computing? What are the architectures that autonomic computing systems should adopt? What are the computing models and algorithms that autonomic computing should be based on? All these questions should be addressed in the research works on autonomic computing. By autonomic computing prototyping we mean that these questions are addressed and examined by developing various prototyping systems. These can be observed in the various works presented at the international conferences on autonomic computing [4, 5].

2.4 Software-Enabled Control

Software-enabled control (referred later in Section 3 as *SEC*) was initially a research program launched by the U.S. Defense Advanced Research Projects Agency (DARPA) [60, 8]. The motivation for SEC is two-fold.

First, conventional control systems including adaptive control systems and robust control systems are often over-designed based on simplified models of system dynamics and well-defined operational environments. This leads to underperformance in normal environments and control vulnerabilities that arise in extreme environments such as damaged control surfaces for modern aircrafts. Second, developments in software technology have enabled new apparatus for control systems, including device networks, smart sensors, programmable actuators, and systems-on-a-chip. A challenge is how to design control/software, or how to exploit software and computation to achieve new control capabilities. This requires a new perspective of system dynamics. Besides conventional accounts of parameter uncertainty, noise, and disturbance, the new system dynamics should also take into account dynamic tasking, sensor and actuator reconfiguration, fault detection and isolation, and structural changes in plant model and dimensionality. It should also treat software as a dynamic system which has an internal state, time scales, transients and saturation points, responds to inputs, and produces outputs. SEC is now an established research area that shares the general idea of software cybernetics [60, 8]. In the following we identify three research topics that relate to SEC.

2.4.1 Control Software Architecture

Research in control software architecture is concerned with particular types of software architectures that fit well the implementation of complex control algorithms. There are two primary concerns. First, the control software of concern is mostly real-time embedded software. This is particularly true for airborne software of modern aircrafts. Second, monolithic structures should be avoided to reflect the closed-loop feedback feature of control systems [60, 8]. Overall, the control software architecture should follow the new perspective of system dynamics. An outgrowth of this research topic is the so-called open control platform (OCP) for Unmanned Aerial Vehicles (UAVs), which is an object-oriented software infrastructure that allows seamless integration of cross-platform software and hardware components in any control system architecture.

2.4.2 Software-Enabled Control Synthesis

The endeavor for software-enabled control was carried out mainly for modern aircrafts in general, and UAVs in particular. How to put flight control, flight management, task management, and software constraints into a unified framework is a grand challenge for computer science, control theory and software engineering. For example, the distributed controller enabled by emerging real-time middleware support can consist of hierarchical systems, integrated subsystems, or independent confederated systems, such as multi-vehicle systems. It is unclear how to synthesize the required coordinated control algorithms.

2.4.3 Control Software Validation

Conventional software validation assumes that the underlying algorithms implemented by the software under validation are given a priori and are not adjusted online. For example, conventional software testing assumes that a test oracle is given a priori. However, this assumption is not true for software-enabled control that follows the new perspective of system dynamics. Control algorithms may be adjusted or even re-configured in response to sensor/actuator failures or unexpected conditions. This may trigger the reconfiguration of control software to adopt an alternative software architecture. On the other hand, software reconfiguration in response to software component failures may require that control algorithms are updated to guarantee flight safety. There is a dynamic process of interactions between control algorithms and software systems. This imposes a challenge to software validation [60, 8].

3 Applications of Software Cybernetics

In this section we highlight applications developed within the research areas presented in Section 2. Since the majority of the research work falls within more than one of the areas from Section 2, we have organized the works by application area. The research topics are then individually categorized based on the four main Software Cybernetics research areas defined in Section 2 and are also listed below.

- Fundamental Principles - *FP*
- Cybernetic Software Engineering - *CSE*
- Cybernetic Autonomic Computing - *CAC*
- Software-Enable Control - *SEC*

3.1 Process Management

By *process management* we refer to the work directed toward the task of bringing control-theoretic approaches to bear on the perennial problems of software process improvement and control. Xu et al. [73] have mapped the 66 key practice areas from the Requirements Engineering Good Practice Guide [64] to the corresponding parts of a typical control system (i.e. to actuators, sensors, etc.) and sketch an overview of the process of building a Requirements Process Controller. Their work falls in the *CSE* area.

Management of the construction phase of incremental software development is addressed by Miller [51], where a

state-model of the construction phase is proposed, and an outline given of a control strategy based on Model Predictive Control (MPC) [18]. The control attempts to minimize the deviation between the actual progress and the schedule while balancing the cost of the control resources with the cost of schedule deviation. These projects can be characterized as part of *FP* and *CSE* areas.

Modeling and control of the System Test Phase (STP) of software development within a control-theoretic formalism has been addressed Cangussu et al. [21, 22] where a state-model is constructed for the STP and a partial eigenvalue-assignment control technique is proposed. The control technique presents a test manager with a set of options which will likely achieve the quality objectives by the schedule deadline. Miller et al. propose [50] a controller for the STP model based on MPC. This work can also be characterized as part of *FP* and *CSE* areas.

Buy et al. build upon their work on time-extended Petrinets [9, 10] to construct a supervisory controller capable of enforcing constraints on a class of workflow processes. Workflows can be used to describe many processes, including those in software development. This work is best characterized by the area of *CSE*.

Padberg has studied the link between software process modeling and Markov Decision Theory [55], where a model of software development is proposed as a Markov Decision Process, and an optimal schedule is derived using a dynamic programming approach. *CSE* again is the best characterization of this work.

3.2 Software Development

While there is a body of work proposing architecture and design elements to support software cybernetic implementations, these are out-of-scope for the present survey. Instead we focus on the *actual usage* of control-theoretic techniques and ideas.

The task of designing software can be aided by supervisory control techniques which commonly augment existing systems to impose constraints. Examples of design synthesis via supervisory control are given in Sections 3.1 and 3.4.

Software testing has also received considerable attention from the software cybernetics community [15]. Cai et al. [17] view the software under test as a controlled object which is modeled by a controlled Markov chain. The testing strategy is synthesized as an optimal controller of the software under test. This body of work falls within the *CSE* area.

Research has also focused on the application of adaptation techniques to improve random testing. Chan et al. [23] propose an adaptive *center of gravity* constraint to pure random testing to improve the input domain coverage with fewer tests. Cai [16] proposes a dynamic partitioning of the input domain for random testing to improve the test selection process. As above, *CSE* characterizes this work.

3.3 Adaptive Software

Control-theoretic foundations for the construction of adaptive software have been studied [19, 26, 56]. For example, a system identification technique is used to capture the behavior of a software application with respect to a specified resource usage [19]. The derived model is then used to predict constraint violations and the software is adapted accordingly to avoid such violations. An increase in software robustness is achieved with this adaptation. The work on adaptive systems is better characterized by the *CAC* research area.

3.4 Safety

Software cybernetics has been used to address the enforcement of safety policies in collaborative environments. Sridharan et al. [65, 66] propose a safety enforcement environment called "ConnectedSpaces" with a formalism for describing and exchanging the safety policy of a device within a ConnectedSpace. A form of supervisory control is then applied to achieve online generation of a safety controller for the ConnectedSpace that adapts as devices enter and exit the ConnectedSpace. This research project involves both areas of *FP* and *CSE*.

Adaptive software introduces its own safety concerns: Is it possible for the adaptation to fail and leave the system in a dangerous state ? This issue is addressed by Liu et al. [47] where a stability monitor is constructed based on Lyapunov Stability Theory [75]. The stability monitor determines whether the current data will prevent the adaptation process from converging (i.e. due to abnormal or incorrect data) and, if so, it prevents the data from reaching the adaptation routine. *FP* and *CAC* characterizes these projects.

3.5 Networking

Techniques based on control theory have been applied to common problems in networking as well. Moerdyk et al. [53] propose a hybrid optimal controller based on Model Predictive Control which achieves load-balancing in a cluster of computer nodes.

Tan et al. [71] propose a technique for handling high-bandwidth traffic aggregates (e.g. DOS attacks) by installing rate throttles in upstream routers and building control-theoretic algorithms to adaptively, robustly, and fairly set the throttle rates at the routers. *CAC* can be used to characterize the projects in this section.

3.6 Fault Tolerance

Software rejuvenation refers to the idea that software can repair its internal state to prevent a more severe future failure. A framework for adaptive software rejuvenation is proposed by Bao et al. [6] with examples given for monitoring and adapting the rejuvenation schedule in response to resource loss (e.g. memory leaks.)

A self-stabilizing program is one which guarantees the arrival at a legitimate state after a finite number of steps, regardless of the initial state [27]. Gouda and Herman [34] relate program adaptation to self-stabilization in the context of fault tolerance. The research areas of *CAC* and *SEC* characterize these projects.

3.7 Information security

Venkatesan and Bhattacharya [69] propose a threat-adaptive security policy in which a trust model is developed as a finite state machine from a set of rules specifying how trust is to be adjusted. Depending on the level of trust, the system requires varying levels of authentication; the intent is to improve performance while retaining control over the access of untrusted users.

An approach to security quantification is proposed by Griffin at al. [35]. A state-space representation of security is proposed, followed by a stochastic attack model. Analysis of the pair yields estimates of mean time-to-failure, the probability of reaching a particular fail-state, and a method of optimizing the security policy. *FP* and *CSE* are a good characterization for these work.

4 Future Directions

Many areas have already benefited from the use of control theoretical aspects but much more needs to be explored. There is always the alternative of increasing the number of areas where Software Cybernetics can be applied. This is naturally occurring as more and more researchers embrace the benefits of applying control techniques and theories within their research areas. The diverse areas surveyed in Section 3 provide a clear indication of this phenomena. However, most of the research on Software Cybernetics can be considered to be in their preliminary stages and much more detailed solutions with a full body of results/concepts must be in place before Software Cybernetics can achieve a reasonable level of maturity.

For example, the work on software project management has been almost restricted to the final phases of the development process or more specifically to the testing phases. This is due to the fact that later phases are easier to quantify and less subjective than early phases of the development process. The lack of better and/or more precise quantification mechanisms to represent early phases are not only a matter of their subjectivity but also due to the immaturity of software engineering [43]. Clearly, Software Cybernetics research in this area has to move towards all the phases of the development process until a full body of models and control mechanisms are in place to regulate the entire development process.

Another aspect to be considered is how Software Cybernetics will require the development of new techniques (or the adjustment of existing ones) used in control. For example, the quantification of noise is well understood and used when controlling physical systems. However, noise in software is difficult to quantify. Though most researchers make assumptions about noise (for example, assuming a White Gaussian (Normal) random noise [44], $w[n] \sim \mathcal{N}[0, \sigma^2]$, of zero mean and variance σ^2 at an instant n to represent noise), we believe that there has not been a detailed study about the quantification of noise for software systems. The availability of such study may lead to the development of new techniques to handle the specifics of software systems.

5 Concluding Remarks

Control theory is a tool that can be used to regulate systems and processes. Software systems do not operate in a static environment and many aspects of software (either at operating system, networking, or application level) need to be regulated to improve performance, increase reliability, and even regulate resource usage. Similarly, the same concepts can be applied to regulate the development process of a software product. Software Cybernetics is therefore an area that brings together researchers from both the software and the control systems communities to develop solutions for these problems. The initial results of research projects on Software Cybernetics are an indication of the benefits of molding this new exciting area. Developments targeting network security, safety, testing process, and fault tolerance, among others, demonstrate this potential.

References

[1] Wikipedia. http://en.wikipedia.org/wiki/Main_Page.

[2] Autonomic computing: IBM perspective on the state of information technology. IBM http://www.ibm.com/research/autonomic, 2001.

[3] Introduction to SDL 88. http://www.sdl-forum.org/sdl88tutorial/index.html, 2002.

[4] Proceedings of the international conference on autonomic computing. IEEE Computer Society, 2004.

[5] Proceedings of the international conference on autonomic computing. IEEE Computer Society, 2005.

[6] Yujuan Bao, Xiaobai Sun, and K. S. Trivedi. Adaptive software rejuvenation: degradation model and rejuvenation scheme. In *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on Dependable Systems and Networks*, pages 241–248, June 2003.

[7] G. Barrett and S. Lafortune. Bisimulation, the supervisory control problem and strong model matching for finite state machines. *Discrete Event Dynamic Systems: Theory and Applications*, 8:377–429, 1998.

[8] J. S. Bay and B. S. Heck. Software-enabled control: An introduction to the special section. *IEEE Control Systems Magazine*, pages 19–20, 2003.

[9] Ugo Buy and Houshang Darabi. Deadline-enforcing supervisory control for time Petri nets. In *CESA'2003 – IMACS Multiconference on Computational Engineering in Systems Applications*, Lille, France, July 2003. Electronic proceedings on CD-ROM.

[10] Ugo Buy and Houshang Darabi. Sidestepping verification complexity with supervisory control. In *Proceedings 2003 Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation – The Monterey Workshop Series*, Chicago, Illinois, September 2003. Available at www.cs.uic.edu/ shatz/SEES.

[11] K. Y. Cai, J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur. An overview of software cybernetics. In *Proceedings of the 11th International Workshop on Software Technology and Engineering Practice*, pages 77–86. IEEE Computer Society Press, 2003.

[12] K. Y. Cai, T. Y. Chen, and T. H. Tse. Towards research on software cybernetics. In *Proc. 7th IEEE International Symposium on High Assurance Systems Engineering*, pages 240–241, 2002.

[13] K. Y. Cai, Y. C. Li, and K. Liu. Optimal and adaptive testing for software reliability assessment. *Information and Software Technology*, 46:989–1000, 2004.

[14] K. Y. Cai and X. Y. Wang. Towards a control-theoretical approach to software fault-tolerance. In *Proc. the 4th International Conference on Quality Software*, pages 198–205. IEEE Computer Society Press, 2004.

[15] Kai-Yuan Cai. Optimal software testing and adaptive software testing in the context of software cybernetics. *Information and Software Technology*, 44:841–855, 2002.

[16] Kai-Yuan Cai, Tao Jing, and Cheng-Gang Bai. Partition testing with dynamic partitioning. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, volume 2, pages 113–116, July 2005.

[17] Kai-Yuan Cai, Y. C. Li, and K. Liu. Optimal software testing in the setting of controlled markov chains. *European Journal of Operational Research*, 162(2):552–579, 2005.

[18] E. F. Camacho and C. Bordons. *Model Predictive Control*. Springer Publication, January 2004.

[19] J. W. Cangussu, K. Cooper, and C. Li. A control theory based framework for dynamic adaptable systems. In *Proceedings of the 19th Annual ACM Symposium on Applied Computing*, March 2004.

[20] João W. Cangussu, Richard M. Karcich, Raymond A. DeCarlo, and Aditya P. Mathur. Software release control using defect based quality estimation. In *Prooceding of* 15^{th} *International Symposium on Software Reliability Engineering*, Saint-Malo, Bretagne, France, November 2-5 2004. IEEE.

[21] Joao W. Cangussu, Raymond A. DeCarlo, and Aditya P. Mathur. A formal model of the software test process. *IEEE Transactions on Software Engineering*, 28(8):782 – 796, August 2002.

[22] Joao W. Cangussu, Raymond A. DeCarlo, and Aditya P. Mathur. Using sensitivity analysis to validate a state variable model of the software test process. *IEEE Transactions on Software Engineering*, 29(5):430 – 443, May 2003.

[23] F. T. Chan, K. P. Chan, T. Y. Chen, and S. M. Yiu. Adaptive random testing with cg constraint. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 96–99, 2004.

[24] C. T. Chen. *Linear System Theory and Design*. CBS College Publishing, 1984.

[25] C. Dellarocas, M. Klein, and H. Shrobe. An architecture for constructing self-evolving software systems. In *Proc. the Third International Software Architecture Workshop*, pages 29–32, 1998.

[26] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. E. Kaiser, and D. Phung. A control theory foundation for self-managing computing systems. *IEEE Journal on Selected Areas in Communications*, 23(12):2213–2222, 2005.

[27] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[28] T. Dohi, K. Goseva-Popstojanova, K. Vaidyanathan, K. S. Trivedi, and S.Osaki. *Preventive Software Rejuvenation - Theory and Applications*. Springer Handbook of Reliability, Springer-Verlag, 2002.

[29] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[30] P. Oreizy et al. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems,*, pages 54–62, May/June 1999.

[31] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[32] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of preventive maintenance in transactions based software systems. *IEEE Transactions on Computers*, 47(1):96–107, 1998.

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

[34] M. G. Gouda and T. Herman. Adaptive programming. *Software Engineering, IEEE Transactions on*, 17(9):911 – 921, September 1991.

[35] C. Griffin, B. Madan, and T. Trivedi. State space approach to security quantification. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, volume 2, pages 83–88, July 2005.

[36] K. C. Gross, V. Bhardwaj, and R. Bickford. Proactive detection of software aging mechanisms in performance critical computers. In *Proc. the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, 2003.

[37] E. Haghverdi, P. Tabuada, and G. J. Pappas. Bisimulation relations for dynamical, control, and hybrid systems. *Theoretical Computer Science*, 2005.

[38] K. Herrmann, G. Muhl, and K. Geihs. Self management: The solution to complexity or just another problem. *IEEE Distributed Systems Online*, 6(1), 2005.

[39] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. Software rejuvenation: Analysis, module, and applications. In *Proc. The 25th International Symposium on Fault-Tolerant Computing*, pages 381–390, 1995.

[40] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[41] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, pages 41–50, January 2003.

[42] E. Kushilevitz and N. Nisan. *Communication Complexity,*. Cambridge University Press, 1997.

[43] M. M. Lehman, D. E. Perry, and W. M. Turski. Why is it so hard to find feedback control in software process? In *Proceedings of 19th International Australasian Computer Science Conference*, pages 107–115, Melbourne, Australia, January 1996.

[44] Alberto Leon-Garcia. *Probability and Random Processes for Electrical Engineering*. Addison-Wesley Publishing Company Inc, 2nd edition, May 1994.

[45] L. Li, K. Vaidyanathan, and K. S. Trivedi. An approach to estimation of software aging in a web server. In *Proc. International Symposium on Empirical Software Engineering*, October 2002.

[46] P. Lin, A. MacArthur, and J. Leaney. Defining autonomic computing: A software engineering perspective. In *Proc. the 2005 Australian Software Engineering Conference*. IEEE Computer Society, 2005.

[47] Y. Liu, S. Yerramalla, E. Fuller, B. Cukic, and S. Gururajan. Adaptive control software: can we guarantee safety? In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 100–103, 2004.

[48] H. Marchand and M. Samaan. Incremented design of a power transformer station controller using a controller synthesis methodology. *IEEE Transactions on Software Engineering*, 26(8):729–741, 2000.

[49] E. Marshall. Fatal error: How patriot overlooked a scud. *Science*, page 1347, March 1992.

[50] S. D. Miller, R. A. DeCarlo, A. P. Mathur, and J. W. Cangussu. A control-theoretic approach to the management of the software system test phase. *Journal of Systems and Software; Special section on Software Cybernetics*, 11(79):1486–1503, November 2006.

[51] Scott D. Miller. A control-theoretic aid to managing the construction phase in incremental software development. In *30th Annual International Conference on Computer Software and Applications (COMPSAC).*, 2006.

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

[53] Brian Moerdyk, Raymond A. Decarlo, Douglas Birdwell, Milos Zefran, and John Chiasson. Hybrid optimal control for load balancing in a cluster of computer nodes. In *Proceedings of the IEEE International Conference on Control Applications*, October 2006.

[54] L. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, pages 2–13, 1987.

[55] Frank Padberg. Linking software process modeling with markov decision theory. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. 28th Annual International*, volume 2, pages 152–155, 2004.

[56] J. Palsberg, C. Xiao, and K. Lieberherr. Efficient implementation of adaptive software. *ACM Transactions on Programming Languages and Systems*, 17(2):264–292, 1995.

[57] P.Wang and K.Y.Cai. Representing extended finite state machines for SDL by a novel control model of discrete event systems. In *Proceedings of the Sixth International Conference on Quality Software*. IEEE Computer Society Press, 2006.

[58] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. IEEE*, 77:81–98, 1989.

[59] J. J. M. M. Rutten. Coalgebra, concurrency, and control. CWI, SEN-R9921, 1999.

[60] T. Samad and G. Balas, editors. *Software-Enabled Control: Information Technology for Dynamical Systems*. IEEE Press, 2003.

[61] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[62] M. M. Seron, J. H. Braslavsky, and G. C. Goodwin. *Fundamental Limitations in Filtering and Control*. Springer, 1997.

[63] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu. Software aging and multifractality of memory resources. In *Proc. the 2003 International Conference on Dependable Systems and Networks*, 2003.

[64] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley and Sons, Inc., 1997.

[65] Baskar Sridharan, Aditya P. Mathur, and Kai-Yuan Cai. Synthesizing distributed controller for safe operation of connected spaces. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communication*, pages 452–459, 2003.

[66] Baskar Sridharan, Aditya P. Mathur, and Kai-Yuan Cai. Using supervisory control to synthesize safety controllers for connnectedspaces. In *Proceedings of the 3rd International Conference on Quality Software*, pages 186–193. IEEE Computer Society Press, 2003.

[67] C.Lu R.Zhang Y.Lu T.F.Abdelzaher, J.A.Stankovic. Feedback performance control in software services. *IEEE Control Systems Magazine*, pages 74–90, June 2003.

[68] A. J. van der Schaft. Equivalence of dynamical systems by bisimulation. *IEEE Transactions on Automatic Control*, 49(12):2160–2172, 2004.

[69] R. M. Venkatesan and S. Bhattacharya. Threat-adaptive security policy. In *Performance, Computing, and Communications Conference, 1997. IPCCC 1997., IEEE International*, pages 525–531, February 1997.

[70] X. Y. Wang, Y. C. Li, and K. Y. Cai. On the polynomial dynamical system approach to software development. *Science in China (Series F)*, 47(4):437–457, 2004.

[71] Chee wei Tan, Dah ming Chiu, J. C. S. Lui, and David K. Yau. Handling high-bandwidth traffic aggregates by receiver-driven feedback control. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, volume 2, pages 143–145, July 2005.

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

[73] Hong Xu, Pete Sawyer, and Ian Sommerville. Requirement process establishment and improvement: from the viewpoint of cybernetics. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, volume 2, pages 89–92, July 2005.

[74] C. Zhou, R. Kumar, and S. Jiang. Control of nondeterministic discrete-event systems for bisimulation equivalence. *IEEE Transactions on Automatic Control*, 51(5):754–765, 2006.

[75] V. I. Zubov. *Methods of A. M. Lyapunov and Their Application*. U. S. Atomic Energy Commission, 1957.