The Journal of Systems and Software 124 (2017) 169–186

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Modern software cybernetics: New trends

Hongji Yang[a,*], Feng Chen[b], Suleiman Aliyu[b]

[a] Centre for Creative Computing, Bath Spa University, UK
[b] School of Computer Science and Informatics, De Montfort University, UK

ARTICLE INFO

Article history:
Received 9 June 2016
Revised 19 August 2016
Accepted 30 August 2016
Available online 8 September 2016

Keywords:
Software cybernetics
Control engineering
Software engineering
Computer science
Artificial intelligence

ABSTRACT

Software cybernetics research is to apply a variety of techniques from cybernetics research to software engineering research. For more than fifteen years since 2001, there has been a dramatic increase in work relating to software cybernetics. From cybernetics viewpoint, the work is mainly on the first-order level, namely, the software under observation and control. Beyond the first-order cybernetics, the software, developers/users, and running environments influence each other and thus create feedback to form more complicated systems. We classify software cybernetics as Software Cybernetics I based on the first-order cybernetics, and as Software Cybernetics II based on the higher order cybernetics. This paper provides a review of the literature on software cybernetics, particularly focusing on the transition from Software Cybernetics I to Software Cybernetics II. The results of the survey indicate that some new research areas such as Internet of Things, big data, cloud computing, cyber-physical systems, and even creative computing are related to Software Cybernetics II. The paper identifies the relationships between the techniques of Software Cybernetics II applied and the new research areas to which they have been applied, formulates research problems and challenges of software cybernetics with the application of principles of Phase II of software cybernetics; identifies and highlights new research trends of software cybernetic for further research.

1. Introduction

No one will doubt today that software is critical for modern society and is being used everywhere, which requires the software to be produced on time, within budget, and performed as expected. The fact that the software development industry is in a crisis was recognised in 1960 s. As one of the most important areas of computer science, software engineering had its origin as a solution to the "software crisis" (Dijkstra, 1972; Yang et al., 2008). According to IEEE, software engineering is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches, i.e., the application of engineering to software.

Problems associated with the software crisis have largely been caused by the character of the large and complicated software itself (Brooks, 1987). Complexity is an essential property of all large pieces of software. Software, as an artifact, is complex in nature. The difficulty of designing, implementing and launching software increases exponentially with the size of the system. It is difficult if not impossible to enumerate all the states and interactions of the software. Complexity is added by software's conformity, namely, software must conform to real-world constraints. Additional complexity arises from the fact that the software entity is constantly subject to pressures for change. However, this does not mean that software is easy to change. A final source of software complexity arises from software's inherent invisibility. Presently software systems, e.g. cyber-physical systems, Internet of Things, cloud computing, are becoming more and more complex and hence new models and methods in software engineering are required dramatically. Studies in cybernetics provide a means to control the complexity and adapt to change to make software more efficient and effective, namely, to apply techniques and principles of cybernetics to solve software development problems.

Software cybernetics is a subdivision of cybernetics in the domain of software engineering. The term software cybernetics was first used in Cai (2002a). The author mentioned that the idea of software cybernetics was proposed in 1994 with an attempt to apply cybernetic or control-theoretical approaches to solving problems in software engineering. There has been a consistent expansion since then, mainly through the International Workshop on Software Cybernetics (IWSC) (Cangussu et al., 2007). An overview of software cybernetics is available in other surveys (Cai, 2002a; Cai et al., 2003; Belli et al., 2006; Cangussu et al., 2007). Despite the excellent work in the surveys listed above, from cybernetics viewpoint, the work is mainly on the first-order level. There

* Corresponding author.
E-mail addresses: hongji.yang@gmail.com, h.yang@bathspa.ac.uk (H. Yang).

http://dx.doi.org/10.1016/j.jss.2016.08.095
0164-1212/

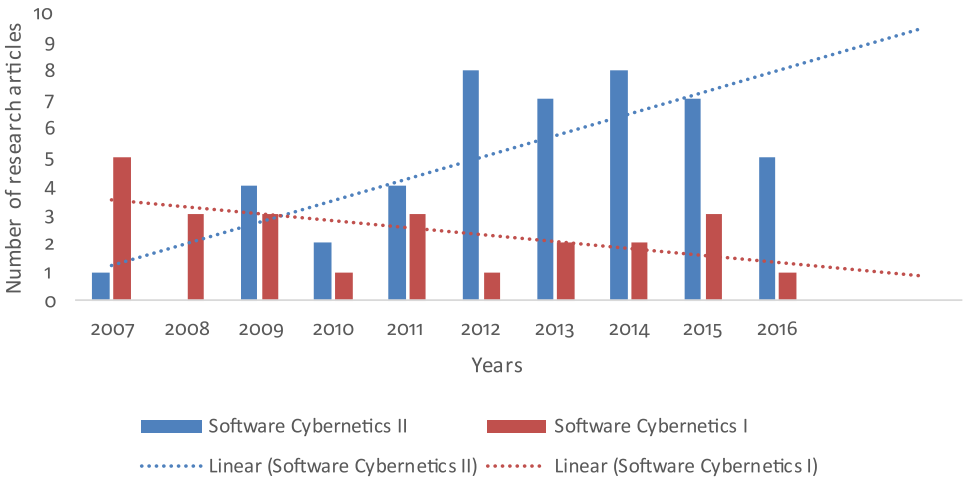

Fig. 1. Research trend of software cybernetics.

remains no comprehensive survey on all issues of software cybernetics. It is, therefore, timely to review the software cybernetics literature to shed new light on the new trends in software cybernetics based on the principles of the Phase II of software cybernetics. Hence, the aim of this review paper is to explore from a software engineering standpoint, the progression from first-order software cybernetics to higher-order software cybernetics.

This review attempts to group recent research on software cybernetics and suggests an upward linear trend (growth) in more recent research concerned with the Phase II software cybernetics as well as a gradual decline in research related to the Phase I of software cybernetics from a software engineering perspective (see Fig. 1). In preparing this article, a broad search for research articles was conducted by generating search words or phrases consisting of related keywords mainly to obtain a comprehensive list of relevant work. Essentially, the online search comprised simple keyword phrases like “Software cybernetics”, and more complex phrases such as “software engineering and feedback control” or “Software engineering and control engineering”. Quality scholarly research databases in the likes of IEEE Xplore Digital Library, ACM Digital Library, SpringerLink and ScienceDirect were used to narrow down the search results. In addition, Google was also used to check if we missed any important references. To improve the coverage, cross-referenced papers and around 10 key researchers' publications were also checked. At the end, around 120 research papers spread across several conference proceedings and journals were selected as references in this survey.

The rest of the paper is organised as follows: In Section 2, we introduce the background and some basic concepts of cybernetics and software cybernetics. Section 3 provides a series of related studies on software cybernetics by reviewing the state of the art. Section 4 is devoted to exploring the transition from the Phase I of software cybernetics to the Phase II of software cybernetics, discussing challenges of software cybernetics, and identifying the new trends of software cybernetics based on the new properties of software cybernetics. We have discussed several hot issues in some research areas, e.g., Internet of Things, big data, cloud computing, cyber-physical systems, and even creative computing. Lastly, Section 5 summarises the paper and draws conclusions.

2. Background and scope

2.1. Cybernetics movement

As defined by Wiener (1948), cybernetics is concerned with the scientific study of control and communication in animals and machines. Cybernetics is a transdisciplinary approach for exploring regulatory systems, focusing on how systems use information, models, and control actions to steer towards and maintain their goals. This approach, known as first-order cybernetics, is concerned with bringing the system to a stable state by negative feedback processes, which is designed to take place in isolation from its whole situation or environment as a closed system. Although this approach is useful at the level of engineering, it focuses on the local state of a system and overlooks the role of the observer. Here, the observer may be a designer, the user of the system - or even another system. The problem with first-order cybernetics is that it assumes that the system is isolated and closed. However, nothing is totally isolated and there are no closed systems in the real world. Meanwhile, in first-order cybernetics, the role of the observer is ignored. Normally, we treat first-order cybernetics as classical cybernetics.

Geyer and van der Zouwen (1978) discussed a number of characteristics of the emerging cybernetics, known as second-order cybernetics, cybernetics of cybernetics, or meta-cybernetics. One characteristic of cybernetics is observer-dependent and another characteristic is the links between individual and environment. They noted that a transition from classical cybernetics to the new cybernetics involves a transition from classical problems to emerging problems. Heylighen and Joslyn (2001) defined a system as an agent in its own right, interacting with another agent, the observer. Thus, the idea of interaction between observer and system was brought into play. Von Foerster (1979) also stated that first-order cybernetics was the cybernetics of observed systems and second-order cybernetics was the cybernetics of observing systems. The new cybernetics emphasises on communication between several systems which are trying to steer each other, which sometimes leads to the concepts of self-organisation and self-regulation.

While cybernetics of the first and second order might be insufficient to interact with the environment, cybernetics of the third order might handle this problem better. Systems now have become independent to the degree of regulating itself in relation to its surroundings. Third-order cybernetics regards a system more as an active interactive element that is dealing with the stability of the system with respect to both itself and its context. The application of third-order cybernetics includes virtual, proactive, anticipative technologies, and cyberspace, focusing on virtual systems with capable of evolving. In third-order cybernetics, the system can change goals without preprogramming. This means that the observer is considered as a proactive component that not only observes but also decides and acts. Fourth-order cybernetics takes

this one step further. Fourth-order cybernetics deals with, simultaneously, the system and its context. Fourth-order cybernetics may be one of the embodied, fully evolvable of the creative systems. This kind of cybernetics seems to be closely related to artificial life domain (Novikov, 2016).

It should be noted that any new type of cybernetics embeds the elements of the previous ones. Within new cybernetics, many of these contexts have merged to create more complicated systems. The scope of cybernetics is rapidly evolving to encompass hybrid cyber-physical systems with hierarchical distributed processes, data-driven decision-making, and observer-in-the-loop at various scales.

2.2. Principles of new cybernetics

Many of basic ideas have been expressed as a set of fundamental principles (laws) in the cybernetics domain, such as the principle of requisite variety, the principle of feedback, the principle of controllability, and the principle of homeostasis (Self-regulation). The principles of cybernetics have been applied in many fields. Cybernetics cuts across many traditional disciplinary boundaries. Applications of cybernetics are prevalent in computer science, particularly in the field of artificial intelligence, neural networks, and control engineering. However, some new features have been formulated in these cybernetics principles. New cybernetics emphasises autonomy, self-organisation, cognition, and the role of the observer in modelling a system.

2.2.1. The principle of requisite variety

This principle formulated by Ashby (1956) states that it is impossible to create the simple control system for the effective control of a complex system. Essentially, the control system ("regulator") must be as sophisticated as the complex system to be controlled. The complexity of cybernetic systems is usually imparted in a hierarchy. For the new cybernetics, this principle means the model needs to be built adequately and accordingly in terms of a complicated system.

2.2.2. Principle of feedback

Feedback is one of the basic notions and a useful fundamental principle of cybernetics, which can be applied to a great variety of systems and environments. Control theory has its roots in the use of feedback as a means to regulate physical processes and mediate the effect of modelling uncertainty and noise. A new type of feedback is the connection of the observed system to itself by means of the observer. Furthermore, the observed system, observer, and environments influence each other and thus create feedback to form a more complicated system, which emphasises the importance of many more relations among the parts, their interactions, and their relationships to the whole. Feedback may also be in the form of positive feedback.

2.2.3. The principle of homeostasis

Homeostasis has long been considered to be the ultimate goal of control. Homeostasis is the ability of the system to preserve the conservation of stability at the changing external conditions, which is the essence of first-order cybernetics. The cybernetics is often characterised as a science of optimal control of the complicated systems. The optimality is always connected with the chosen goal (criterion). Many real systems are extremely sensitive to weak external interaction. This weakness leads to the importance of even the smallest interaction of the observer with the observed system.

2.2.4. The principle of controllability

Controllability is a fundamental principle of modern control systems. It refers to the ability to move a system within its complete configuration space using only certain acceptable alterations. Controllability is the ability of a system to have control over its responses. For the new cybernetics, especially, the introduction of artificial intelligence makes the cybernetics systems become the complicated third or fourth order cybernetic system. The problems of large-scale systems are not possible to realise only on the base of feedback because of sheer difficulty in creating an ideal model of the controlled system. Many systems cannot be described in details by using purely logical, scientific methods, such as mathematical modelling. All systems can be classified as either deterministic or probabilistic. A deterministic system is a system that can be studied without any uncertainty, namely the system state can be predicted. If the prediction can only be made with some probability, such a system is called probabilistic one.

2.3. Transition from phase I to phase II of software cybernetics

Software cybernetics, in reference to the description of 'cybernetics' by Wiener if software is regarded as part of the machine, can be defined simply as communication and control in software. However, most researchers in the area believe software cybernetics is more diverse in scope. Software cybernetics was described as the interplay between software or software behaviour and control (Cai et al., 2003). In its simplest form, the field of software cybernetics treated software problems and control problems in an integrated way (Cai et al., 2002b).

According to Cai et al. (2003), Software cybernetics addressed key issues and research questions in (i) formalising and quantifying feedback mechanisms in software processes and software, (ii) adopting principles or concepts in control theory to software processes and software, (iii) applying principles or concepts in software theory or engineering to control systems and processes, and (iv) integrating theories in software engineering and control engineering. Their survey also provided key perspectives as motivation and justification for research in software cybernetics.

In addition to this view, Cangussu et al. (2007) defined software cybernetics as "an emerging discipline that explores the theoretically justified interplay between software and control". Their perception for the scope of software cybernetics was a direct consequence of a perceived scarcity in the solid theoretical background for software engineering. Their survey went further to identify and describe four research sub-areas of software cybernetics as fundamental principles, cybernetic software engineering, cybernetic autonomic computing and software-enabled control.

With the advent of Social networks and widespread use of distributed computing and cloud computing in our daily life, the ubiquitous role of software systems suggests that for software cybernetics to add significant value to modern software systems, it will have to expand its scope to integrate inputs from a number of disciplines (Kenett, 2011). Zhu (2012) in his talk at the 9th IWSC annual workshop dedicated to software cybernetics in the era of cloud computing and what this meant for today's software cybernetics. He also suggested that software cybernetics may provide insights into software engineering problems of emergent behaviour in service oriented architecture, self-adaptive architectures, the role of software metrics in control and software evolution in the cloud.

We classify software cybernetics as software cybernetics I based on first-order cybernetics that is typified by feedback loop control e.g. modelling software systems using finite state machines. Software cybernetics II is based on the higher order cybernetics, which is characterised by developers, software under development and running environments influencing each other to form more complex systems. software cybernetics II is typically based on new software deployment and development models, e.g. Agent-based software engineering, cloud computing, and creative computing.

2.4. Theoretical foundation in software cybernetics

There are a number of techniques related to software cybernetics. Without covering all of them, the following discussion will provide a clear scope and taxonomy of the enabling techniques. There are currently two broad facets to research methods in the area software cybernetics: model-based with a mathematical framework and logic-centric approaches whose underlying principles are from the field of artificial intelligence.

2.4.1. Theoretical model in software cybernetics research

Various mathematical methods are used to design effective system models, which constitute the main methodological technique in software cybernetic research. Dynamic system models, formal models such as the extended finite state automata and controlled Markov chain exemplify model-based approaches. Supervisory-control theory is based on the finite state automata to represent discrete-event dynamic systems. Previous research work has developed linear dynamic system models to describe software service behaviours and the software test process. Cai (2002a) viewed software testing as a control problem and devised a Control Markov Chains (CMC) approach to determine an optimal test strategy. The CMC approach provides theoretic justification that for some circumstances a Markov model matches the software test profile. Hu et al. (2008) proposed a new adaptive software testing approach based on the improved CMC that aimed to replace several presumptions adopted by previous models with more realistic situations in software testing.

The finite state machine (FSM) is a good example of a formal model in software cybernetics. Gaudin and Bagnato (2011) described a set of safe behaviours as finite state machines (FSMs). In doing so, they relied on the Supervisory Control Theory to represent over-approximations of the behaviours of the system to be controlled. The extended finite state machine (EFSM) is widely used to model communication software behaviours (Joao et al., 2007). Yang and Gohari (2005) presented a framework to implement supervisory control map using extended finite state machine (EFSM) as an embedded part of the controlled system. Their work also showed that the constructed EFSM was able to exhibit the same behaviour as the supervised system. Wang and Cai (2006) developed algorithms that transformed EFSM for specification and description language (SDL) to the control model of discrete event systems (DES). Their research efforts indicated that EFSM could be expressed as a closed loop control system. The GK-tail algorithm was a technique that used interaction traces to automatically generate EFSM models of the behaviour of software systems (Lorenzoli et al., 2008).

In more recent work, Zhao et al. (2014) aimed at improving the GK-tail algorithm by proposing an improved method for modelling software behaviour based on EFSM. To verify the efficacy of their improved method, they designed and implemented a software behaviour modelling system. Wang and Cai (2012) investigated the supervisory control problem of the restricted EFSM model and proposed a necessary and sufficient condition and an optimal algorithm to the supervisor. The result was claimed promising to relate the software design problem to supervisory control theory and enriched the research content of software cybernetics.

Girard and Pappas (2007) developed a framework of system approximation for metric transition systems by developing a hierarchy of metrics for reachable set inclusion, language inclusion and simulation and bi-simulation relations. They proposed a compositional approximation framework for a synchronous composition operator and obtained approximations for the pseudo-metrics by considering Lyapunov-like functions called simulation and bi-simulation functions. Julius and Pappas (2009) developed a notion of approximation for a class of stochastic hybrid systems. The approximation framework was based on the so-called stochastic simulation functions. These Lyapunov-like functions could be used to rigorously quantify the distance or difference between a system and its approximate abstraction.

2.4.2. Application of artificial intelligence in software cybernetics research

Growth in the field of artificial intelligence has bolstered active research in the area of software cybernetics. Particularly, software engineering has become an important application area for machine learning techniques. Fuzzy logic, a knowledge-based formal model for machine learning, is a typical instance of the logic-centric or rule-based approach used in software cybernetics research. Yang et al. (2011) applied fuzzy based logic to control complex software systems with the aim of addressing challenges or uncertainty in complex software systems. The aim of their fuzzy-based approach was to develop a self-adaptive executable software framework to improve the performance of process control mission-critical systems. Ding et al. (2016) designed an adaptive control system based on fuzzy logic and update the controller itself with a set of fuzzy rules. The principles of software cybernetics were applied in service-based systems (SBS) to synthesise controllers for online adaptation and monitoring (Yau et al., 2007). This approach also included a logic-based technique (situation-aware) for planning resources offline taking as input timing and resource constraints. Park and Yeom (2013) used the concept of feedback in software cybernetics to propose an approach for validating Semantic Web Rule Language (SWRL) rules. Their method comprised preparation, structural analysis, contextual analysis and the SWRL rule adaptation. Their approach constituted a feedback loop in which the SWRL rule to be validated acted as the controlled object while the validation of SWRL rules represented the controller. The introduction of artificial intelligence has brought software cybernetics research to a new level, third-order cybernetics.

2.5. Scope of the survey

Based on the definition of “Interplay between software and control”, software cybernetic should include two perspectives: applying the theoretical principles of cybernetics to the domain of software engineering or applying software engineering methodologies to cybernetic/control system. The first perspective of software cybernetics research is meant to foster improvement in quality assurance of software as well as the software process. Cai et al. (2002b) viewed this as a new form of software engineering. Software-enabled control (SEC) (Bay and Heck, 2003; Heck et al., 2001) can be treated as the second perspective of software cybernetics research. The technical goal of SEC was to develop a sort of new software enabled control methods based on the principles and methods of software engineering (Cangussu et al., 2007). The work in SEC area exemplifies some of the software engineering methodologies in control software development, such as software patterns, reusable control software, and open control platforms. However, since SEC is an established research area, there is no new development in terms of software cybernetics. For this reason, the present survey will not review this topic further.

Cybernetic software engineering treats each phase of the software development process as a control problem. There are two natural lines of research. One would be to model each phase of the software development process as a feedback control problem. Another research view in this perspective treats problems in software engineering as search or optimisation problems. This field of research, in which computational search is applied to solve problems in software engineering is referred to as Search Based Software

Engineering (SBSE). There have been several important surveys in this widely studied general area (Harman & Jones, 2001; Harman, 2006; Harman, 2012a; Harman et al., 2012b; Harman et al., 2013; Sayyad & Ammar, 2013). While there is a body of work proposing SBSE to support software cybernetic, these are out-of-scope for the present survey. Instead, we would concentrate only on ideas of optimisation based control-theoretic techniques.

Software engineering areas to which software cybernetics has been applied will be reviewed. There have been two important surveys (Cai et al., 2003; Cangussu et al., 2007) in this widely studied general area. Hence, this survey focuses on the new publications after the review carried out by Cangussu et al. (2007).

3. Software cybernetics: software and software process viewpoint

3.1. Software requirements/specifications

Software requirement engineering process is an interactive process between the software developers and the end users. This research area is to seek practical synergies between the two disciplines of requirements and cybernetics, to explore the possibilities of formulating problems in requirements with concepts and frameworks from cybernetics, and to understand to what extent that known research results from cybernetics can be applied to address requirements problems to guide the corresponding process improvement (Liu et al., 2016b).

Xu et al. (2006) applied classical control theory to the requirement process improvement. They proposed a requirement process control (RPC) system, which was a framework for improving the requirement process. The practical application of their RPC system was limited to mature organisations because their approach overlooked the need for collection of historical data on the requirement elicitation process.

Liu et al. (2007) focused on how to improve the satisfaction of user's requirements using goal-oriented requirement models. Their article suggested the need to have a goal state that was quantifiable, streamline a set of non-trivial quantifiable parameters to create a feedback loop, and define action sets to enable control.

The system may fail in achieving any of its initial requirements. Souza (2012) considered feedback loops as first class citizens and provided a way of specifying goals as constraints on their success/failure. This research was based on the system being able to monitor its own requirements at runtime. The contributions were new types of requirements for a feedback control loop that implemented adaptability for a target system, and a systematic process and framework for conducting system identification and reconfiguration. Tools were designed to facilitate the design and implementation of adaptive systems using this approach.

The latest research effort in requirement elicitation focused on user behavioural data (Liu et al., 2016b). A data-driven requirements elicitation process was formulated as a feedback control system, where the classical requirements elicitation philosophy turned into a continuous optimisation to user behavioural models. Preliminary results from experiments showed that information on latent customer needs and application of current technology were necessary to guide improvements in the requirement process. As a result of the product-specific interpretation of user data, the practical applicability of their approach in different project settings was limited.

The application of software cybernetics to software requirements/specification builds the link between feedback loops and user requirement improvement that can be regarded as an optimisation problem.

3.2. Software design

Software design is increasingly concerned with how to develop better software with good and optimum solutions. There are widely accepted principles, methods, metrics and practises for architecture and program design. Software design today has become a challenging task due to the dynamic nature of the operational environments and conditions, such as changing user requirements, execution context variations, etc.

Autonomic software systems or what are also referred to as self-adaptive systems were suggested as a promising solution for managing the complex and uncertain nature of today's software-intensive systems (Brun et al., 2009; Ahuja & Dangey, 2014). The development of such systems showed to be significantly more challenging than traditional software systems. A promising starting point to meet these challenges was to apply cybernetic or control techniques when designing and reasoning about these systems. Feedback loops constituted an architectural solution for this, and were a first class citizen in the design of such systems (Huebscher & McCann, 2008).

The software engineering community has proposed numerous approaches for making software self-adaptive. These approaches take inspiration from machine learning and control theory, constructing software that monitors and modifies its own behaviour to meet goals. Control theory, in particular, has received considerable attention as it represents a general methodology for creating adaptive systems (Filieri et al., 2015). However, control-theoretical software implementations tend to be ad hoc and it is difficult to understand and reason about the desired properties and behaviour of the resulting adaptive software and its controller. Filieri et al. (2015) proposed a control design process for software systems that enabled automatic analysis and synthesis of a controller that was guaranteed to have the desired properties and behaviours. Self-adaptation ability is particularly desirable for mission critical software (MCS). Yang et al. (2011) proposed a fuzzy control-based approach to providing a systematic, engineering, and intuitive way for programmers to achieve software self-adaptation. The results of the experiments showed that the behaviours could be adjusted online to react to the interventions or changes from external runtime environments. Rammig et al. (2014) discussed general concepts of self-adaptive real-time systems, and how the necessity for adaptation could be identified using online model checking, and how self-adapting safety guards could be designed by means of artificial immune systems. An approach to integrating these techniques into an underlying platform architecture based on mixed-criticality virtualisation was proposed.

Patikirikorala et al. (2012) conducted a systematic survey on the design of self-adaptive software systems using control engineering approaches. A classification model was built to capture and represent the information about literature at a high-level of abstraction. The analysis results showed that the introduction of the feedback loop and controller into the management system potentially enabled the software systems to achieve the runtime performance objectives and maintain the integrity of the system when they were operating in unpredictable and dynamic environments. Liu et al. (2012) proposed a problem-oriented approach to modelling the system composed of the self-adaptive software and its context as an adaptive control system which was equipped with two kinds of feedback loops: context-aware feedback loops and requirements-aware feedback loops. Five classes of software problems were identified to address the different concerns of the adaptive requirements behind the feedback loops. Souza (2012) advocated that adaptive systems would be designed this way from as early as the requirements engineering stage and that reasoning over requirements was fundamental for run-time adaptation. The proposal was goal-oriented and targets software intensive

socio-technical systems in an attempt to integrate control-loop approaches with decentralised agents inspired approaches.

Dobson et al. (2007) presented a model derived from approaches to modelling dynamical systems in which the adaptive behaviour of an autonomic system might be described and analysed as a whole. Insaurralde and Vassev (2014) presented autonomic control architecture for avionics software of unmanned space vehicles. Wang et al. (2012) proposed a general supporting framework for self-adaptive software systems. Three key issues were covered in the framework: (1) the overall control architecture, which adopted the double closed-loop style and respectively included the self-adaptation loop and the self-learning loop; (2) a general descriptive language, which was an application-independent and unified language to represent self-adaptation knowledge about target systems; (3) three implementation mechanisms, including forward reasoning, planning and reinforcement learning using feedback, which were supported by the above descriptive language and executed at runtime in different modules. Finally, one scenario of on-demand services of massive data mining tasks was selected and the case study demonstrated how the framework was customised as required and how the approach worked. Abeywickrama et al. (2013) proposed an approach to developing self-adaptive systems based on feedback loops. SimSOTA was developed as an Eclipse plug-in to support the modelling, simulating and validating of self-adaptive systems based on the proposed feedback loop-based approach. A case study in cooperative electric vehicles was used to evaluate the proposed approach.

Autonomic software systems or self-adaptive systems were complex systems would have to be self-managed: self-configuring themselves for operation, self-protecting from attacks, self-healing from errors and self-tuning for optimal performance (Huebscher & McCann, 2008). Autonomic computing (Lin et al., 2005) was an intelligent computing approach to self-managing computing systems with minimum human interference in a way to provide a stable computing environment. Such an environment could be defined in terms of self-sustaining features of an autonomic computing: they were able to change structure or behaviour at run-time to deal with continuously changing environments and emerging requirements that might be unknown at design-time. Autonomic computing embedded automation in management software such that it could adapt to changes in the configuration, provisioning, protection, and resource utilisation variations at runtime. Also, autonomic computing, as a control system, aimed to resolve constraints related to the optimal usage of resources based on external requests made by users or processes in a reactive way (Solomon et al., 2007).

Alvares et al. (2015) proposed the design of Autonomic Managers (AMs) based on logical discrete control approaches. AMs were largely used to autonomously control reconfigurations within software components. This management was performed based on past monitoring events, configurations as well as behavioural programs defining the adaptation logics and invariant properties. The challenge here was to provide assurances on navigation through the configuration space, which required taking decisions that involved predictions on possible futures of the system. A domain specific language was defined to provide high-level constructs to describe behavioural programs in the context of software components, which could also be translated to Finite State Automata for verification or discrete controller synthesis. The authors believed that the approach could be applied to other domains such as robotics and cloud computing.

Resource management in a large, heterogeneous, and distributed environment becomes a challenging task. Existing resource management techniques, frameworks, and mechanisms can be insufficient to handle these environments, applications, and resource behaviours. Autonomic cloud computing systems check, monitor, control the working of cloud-based systems and applications according to the running situation, such as self-healing, self-protecting, self-configuring, and self-optimising, without the involvement of humans. The current research on autonomic cloud computing is more focused on self-optimising and self-healing aspects. Research on self-configuring and self-protecting policies can provide protection and incorporate dynamic scalability in autonomic cloud computing (Buyya et al., 2012; Mayer et al., 2013).

Self-healing (Kumar & Mukherjee, 2014) is an emerging research discipline, regarded as one of the key autonomic computing attributes. The complexities in computer systems are increasing hence the results in systems that are prone to errors will cause major problems for a user. Ravindran (2014) proposed a self-healing mechanism that monitored, diagnosed and repaired the corrupted files in the application to its original state. An analysis section of the application was done by maintaining the hash values of corresponding files at runtime and recovering the corrupted file from the original application.

Autonomic systems based on QoS (Quality of Service) parameters are inspired by biological systems that can easily handle problems like uncertainty, heterogeneity, dynamism, faults, and so forth. The goal of autonomic systems is to execute an application within a deadline by fulfilling QoS requirements as described by users with minimum complexity. Singh and Chana (2015) depicted QoS-aware autonomic resource management in the Cloud, which would help researchers to find the important characteristics of autonomic resource management and would also help to select the most suitable technique for autonomic resource management in a specific application.

Quality requirements of a software system cannot be optimally met, especially when it is running in an uncertain and changing environment. In principle, a controller at runtime can monitor the change impact on quality requirements of the system, update the expectations and priorities from the environment, and take reasonable actions to improve the overall satisfaction. In practice, however, existing controllers are mostly designed for tuning low-level performance indicators rather than high-level requirements. Peng et al. (2010) combined goal models with feedback loop controllers to make dynamic trade-offs among conflicting soft goals (i.e., the goals with no binary satisfaction criteria). Reflecting the business value of customers, the controller adjusted the preference ranks of soft goals on the basis of runtime feedback. The experimental study on a Web-based system validated that combining PID control theory with preference-based goal reasoning was effective in runtime self-tuning for a real-life software system. Ding et al. (2016) presented a software cybernetics approach to self-tuning the performance of DBMSs. An adaptive control based on fuzzy logic was designed to control the performance parameters, and update the controller itself with a set of fuzzy rules. Experimental results showed that the proposed method was feasible and effective.

Software cybernetics for software design focuses on the adaptive/autonomic feature of modern software. It is natural to apply control-theoretic or cybernetic principles and methods when designing and reasoning about these systems to develop better software in the dynamic operational environments and conditions.

3.3. Implementation/Programming

Software cybernetics aims at improving the reliability of software by introducing the control theory into software implementation systematically. 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. Most software systems can be treated as control systems, and control theories can help guarantee the correctness of software design solutions. This can be aided by supervisory control techniques (Phoha et al., 2005), which commonly augment existing systems to impose constraints. For example, software fault-tolerance can be treated as a robust supervisory control problem (Cai & Wang, 2004; Wang & Cai, 2012). A modest amount of research has applied the control theories of discrete event systems for program synthesis (Cangussu et al., 2007). Supervisory controller synthesis becomes viable as engineers nowadays are familiar with building models for simulation and validation purposes. The synthesised models provide an opportunity for verification, performance, and reliability analysis, increasing the confidence in the control design and validating it before expensive prototypes are built. Ding et al. (2016) applied the principles and concepts in software cybernetics to guide the synthesis of software controllers for monitoring and adapting system behaviours.

Yau et al. (2007) presented a software cybernetics approach to deploying and scheduling workflows with timing and resource constraints in Service-based Systems (SBS). A logic-based technique for modelling and solving timing and resource constraints for workflows in SBS was developed to generate the initial resource assignments, schedules and deployment plans of agents for workflows.

Chen et al. (2009) applied negative feedback from control theory to the software system verification. Software testing, model checking and their two combinations with the negative feedback mechanism were explored.

The principles and concepts in software cybernetics are applied to guide the synthesis of software controllers for monitoring and adapting system behaviours. Baeten and Markovski (2015) proposed a model-driven system engineering approach, referred to as supervisory controller synthesis, which targeted discrete-event control software for high-tech and complex systems. The proposed framework supported extensions with quantitative features for development of quality control software with a process-theoretic foundation. Several industrial case studies highlighted the advantages of the proposed approach. Liao et al. (2013) used a special class of Petri nets, called Gadara nets, to systematically model multithreaded programs with lock allocation and release operations. They proposed an efficient optimal control synthesis methodology for ordinary Gadara nets that exploited the structural properties of Gadara nets via siphon analysis.

Software cybernetics was applied in the process of verification to establish a nested control system by Liu et al. (2016a). The proposed method verified functional requirements in a dynamic environment with constantly changing user requirements, in which the program served as a controlled object, and the verification strategy determined by software behavioural model served as a controller. The main contribution included: 1) software behavioural model was established in software design phase, and a concern-based construction approach was proposed, which started from obtaining the software expected functionality extracted from a requirement text; 2) Program abstract-relationship model was constructed; and 3) Feedback in a form of intermediate code was generated in the process of verification.

Adams et al. (2013) introduced the concept of using cybernetics as a foundational approach for developing cyber security principles. They explored potential applications of an interdisciplinary approach to control theory, systems theory, information theory and game theory to cyber security from a defensive perspective, and introduced the fundamental principles for building non-stationary systems. Vinnakota (2013) presented a generic cybernetics paradigms framework for cyberspace to study the cyberspace holistically from different perspectives like economics, engineering, software and society. This framework was used to study various aspects of cyber-security in any context of a nation, an enterprise or an organisation.

Co et al. (2009) proposed an approach to improving the resilience of software systems that might be subject to attacks from malicious adversaries. The approach was to impose a lightweight and process-level software control system that continuously monitored an application for signs of attack or compromise and responded accordingly. The system used software dynamic translation (SDT) to seamlessly insert arbitrary sensors into an executing application's binary code. Using the information gathered by the sensors, the control system continuously monitored the health of the system and whether the system was under certain attacks. If the control system determined the system was compromised, the appropriate actuators (also inserted by the software dynamic translator) were activated to generate an appropriate response.

There are many ways to improve the quality of code. When people discuss quality control in terms of software cybernetics, it means to apply control theory to improve the quality of system implementation.

3.4. Software testing

In the process of software testing, test cases are selected in accordance with a given testing strategy and applied to the software under test. If we treat the corresponding testing strategy as a control policy or controller, we can treat the software under test as an uncertain controlled object. Further, if a testing goal is given explicitly, the test data selection becomes as an optimal control problem. 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.

Traditionally, the control theoretical approach can quantitatively forecast the test process trends and assist the manager in allocating testing resources with Controlled Markov Chains (CMC) approach. The CMC approach designs an optimal testing strategy to achieve an explicit optimisation goal. Several theorists have proposed the idea of the Markov chain statistical test (MCST), a method of conjoining Markov chains to form a 'Markov blanket', arranging these chains in several recursive layers and producing more efficient test sets samples as a replacement for exhaustive testing. Feedback control was applied to adjust software test process parameters, e.g. through measurements of software reliability, to satisfy desired objectives by Cangussu et al., (2001), Cangussu et al., (2002). Miller et al. (2006) comprised model predictive control and the use of parameter correction to improve the performance of the software test process.

Software testing techniques have to be developed in parallel with the new paradigms, complexity, and scale of software systems. A range of advanced approaches has been proposed to reflect this trend in the context of software cybernetics.

Adaptive testing is a new form of software testing that is based on the feedback and adaptive control principle and can be treated as the software testing counterpart of adaptive control. It means that a software testing strategy should be adjusted on-line by using the testing data collected during software testing (Cai et al., 2007; Cai et al., 2008; Ye et al., 2009). Cai et al. (2007) proposed a new strategy of adaptive software testing to employ fixed-memory feedback for on-line parameter estimations. An experimental study of adaptive testing for software reliability assessment, where the adaptive testing strategy, the random testing strategy and the operational profile based testing strategy, were applied to the Space program in four experiments (Cai et al., 2008). The experimental

results demonstrated that the adaptive testing strategy could really work in practice and might noticeably outperform the other two. Therefore, the adaptive testing strategy could serve as a preferable alternative to the random testing strategy and the operational profile based testing strategy if high confidence in the reliability estimates was required or the real-world operational profile of the software under test could not be accurately identified. In addition, this strategy might contribute to testing large-scale software systems more than to testing small scale software systems. Ye et al. (2009) investigated the computational complexity of the parameter estimation process in two adaptive testing strategies which adopted different parameter estimation methods, namely the genetic algorithm (GA) method, and the recursive least square estimation (RLSE) method. A controlled experiment on the Space program was conducted to measure the relationship between computational complexity and the failure detection efficiency for the two strategies. Abuseta and Swesi (2015) attempted to address self-adaptive software testing issues and propose a testing framework around the feedback control loop proposed by IBM blueprint.

Web Services (WS) and Service-Oriented Architecture (SOA) present a set of unique testing challenges. As services are distributed, it is necessary to test them using a distributed architecture. Furthermore, as these services may keep on changing, testing needs to be adaptive. Bai et al. (2007) proposed an adaptive testing framework which could continuously learn and improve the built-in test strategies. The framework allowed different test cases to be selected based on the recent test results.

Cai et al. (2008) examined the dynamic behaviour of software testing. A set of simplifying assumptions was adopted to formulate and quantify the software testing processes. It was demonstrated that under the simplifying assumptions, the software testing processes could be treated as a linear dynamic system and the software testing processes could be classified as linear or non-linear, and there was an intrinsic link between software testing and system dynamics.

In an attempt to address the bottlenecks of the dynamic random testing, Zhang et al. (2014) proposed history based dynamic random software testing to improve the traditional random testing and random-partition testing strategies by following the idea of software cybernetics. The approach took advantage of historical test data to approximate each sub domain's defect detection rate in real time. The testing profile was dynamically updated during software testing according to testing data collected online so as to improve the subsequent software testing process.

As the overall literature reveals, many software cybernetics research projects are concerned with software testing. There are many ways to treat a software testing problem as a control problem.

3.5. Rejuvenation and software evolution

There are various software rejuvenation approaches existing in literature that can be broadly categorised into two categories namely model based approaches and measurement based approaches (Cotroneo et al., 2011; Sudhakar et al., 2014). Most work focused on predicting the time-to-aging-failure and on the optimal scheduling of software rejuvenation strategies (Cotroneo et al., 2011).

Agepati et al. (2013) adopted the concept of feedback loop control to present a generalised condition-based software rejuvenation model that is applicable to a wide range of applications. The rejuvenation model includes a stochastic deterioration process, a set of rejuvenation actions and their effects, and a schedule inspection policy that identifies the system deterioration. The optimal rejuvenation policy that minimises the overall cost associated with the system is obtained using Markov decision processes.

Li et al. (2009) proposed a self-evolving control method for software in Complex Avionics System to guarantee the behaviours' reliability of software systems at runtime. This method adopted software sensors to monitor the behaviours of the runtime system and achieved self-evolving based on feedback control and scheduling. The method was used in an avionics system to increase the self-adaptability, the reliability and the efficiency of system practically. The result proved that this software cybernetics method made software system easier to maintain.

Liu et al. (2015) proposed a holistic software rejuvenation based fault tolerance scheme for cloud applications, which contained three indispensable parts: adaptive failure detection, ageing degree evaluation, and checkpoint with trace replay based component rejuvenation. Through a preliminary and qualitative evaluation, it showed that the new fault tolerance scheme brought promising improvement on the availability of cloud applications.

Donaires (2010) designated the development and maintenance of complex software systems in situations where the software process and the software architecture needed to change dynamically in order to cope with the impact of unpredicted and frequent environmental changes. A systemic-cybernetic process model, which was a composition of Stafford Beer's viable system model (VSM) and Barry Boehm's spiral model, was proposed to provide adaptability to the software architecture and self-organising capability to the software process.

Machida et al. (2010) presented analytic models using stochastic reward nets for three time-based rejuvenation techniques of virtual machine monitor. Three techniques in terms of steady-state availability and the number of transactions lost were compared. The optimal combination of rejuvenation trigger intervals for each rejuvenation technique was found by a gradient search method. Okamura and Dohi (2011) developed a dynamic rejuvenation policy for a multistage degradation software system and formulated the underlying optimisation problem by a semi-Markov decision process. They also developed an online adaptive algorithm based on the Q-learning and investigated its asymptotic properties.

In order to improve the efficiency and quality of software evolution, Gao et al. (2011) built the model and proposed two different types of feedbacks in software evolution requirement process. The process model of software evolution requirement based on feedback was formalised by coloured dual-transitions Petri net to manage the changing process of software evolution requirement, and thus software could be evolved efficiently with high quality.

Gaudin and Bagnato (2011) presented an approach for system maintenance after the system was deployed. The proposed approach based on control theory allowed for automatic generation of maintenance fixes. The system was instrumented so that it could later be monitored and interacted with a supervisor at runtime to avoid future executions of faulty or vulnerable system functionalities.

Li et al. (2016) proposed a closed-loop feedback mechanism for business process execution. In their feedback control system, process mining played an important role in generating feedback of process execution for the purpose of the redesign. A discovery method based on a kind of augmented event log would bring new research directions for process discovery. Their work presented a case study for application of the data mined model in business process evolution.

Software rejuvenation or software evolution can be treated as a control problem by monitoring the age of the software and manage the change of software.

3.6. Software project management

Software project risk management can help in reducing the incidence of failure and a variety of problems including cost and schedule overruns, unmet user requirements, and the production of systems that are not used or do not deliver business value. Cao and Chen (2009) proposed an approach for optimising software project process based on project returns, which were used as a criterion to assess the quality of a software project process. A model of optimising software risk control and an optimisation algorithm were proposed in this paper. It provided managers with an effective tool to make the risk control decisions and implement the process optimisation at the project planning stage to greatly promote the possibility of success of software projects.

Kandjani et al. (2012) followed an enterprise architecture cybernetics method to reduce the complexity of global software development by using extended axiomatic design theory, thereby increasing the probability of success. Ponisio and van Eck (2012) proposed a framework that provided a set of measurements (selected from the research literature) for control of software development in cooperative settings, and a set of principles and guidelines for the design of an information infrastructure that provided managers with control information. The metrics that support feedback between operational and strategic levels helped organisations to succeed in dealing with this new context of inter-organisational development. Shankar (2012) addressed a framework to build competence of building software solutions in IT Industry. Since the system consisted of the problem, people capabilities, and the solution, there were many visible and invisible relationships between the parameters that constituted different parts. Cybernetics concepts and principles were used to understand the various interrelationships between the component parameters.

It is critical to deploy human resources at the right time in software maintenance projects to deliver varying workloads under the committed Service Level Agreements (SLA). Kundu and Mukherjee (2014) developed a theoretical framework based on cybernetic principles that recommended ramp-up/ramp-down of resources, considering the practical constraints, ensuring the fulfilment of the SLA with the customer with minimal resource cost. The software was developed based on the framework to aid project managers responsible for software maintenance projects. Park (2015) developed an activity-state mapping algorithm and a goal-activity cover algorithm based on the OMG Essence standard, which could help automate the health monitoring of project states and the adaptive planning of project activities in a software engineering project.

The real world software project management problem is determined by the various interrelationships between the components within a project. Goal-activity implies the control of a system to meet the target.

4. Applications of software cybernetics II

Software development has witnessed the transformation from stand-alone, monolithic systems to today's complex, distributed, interconnected, interoperable, adaptive and autonomous systems. These technology trends lead to challenges that need to be addressed with software engineering principles, methods, and practices. These challenges change the role of software and people in the systems, in which technology, software, and people play an equally important part in the systems (Bellavista et al., 2014). The likely nature of modern software systems forms a context of software cybernetics research. Current research and research trend on cloud computing, cyber-physical systems, big data and creative computing will be reviewed and discussed in the context of software cybernetics.

4.1. Software cybernetics in cloud computing

Clouds will become the dominant computing environment of the current and the next decade by delivering all kinds of services, focusing on large-scale resource sharing, innovative applications, and high-performance orientation. Cloud computing is defined by NIST (http://csrc.nist.gov/groups/SNS/cloud-computing/) as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Software engineering issues need to be addressed to make the most effective use of the clouds. Zhu (2012) discussed many issues of cloud computing and software cybernetics, such as service architecture, agent-based computing, metrics and software evolution.

As Clouds are complex, large-scale, and heterogeneous distributed systems, management of their resources is a challenging task. They need automated and integrated intelligent strategies for provisioning of resources to offer services that are secure, reliable, and cost-efficient. Hence, effective management of services becomes fundamental in software platforms that constitute the fabric of computing Clouds. Autonomic computing provides a path towards controlling cloud computing services (Ionescu, 2011). Buyya et al. (2012) identified open issues in autonomic resource provisioning and presented innovative management techniques for supporting SaaS applications hosted on Clouds. A conceptual architecture and early results evidencing the benefits of autonomic management of Clouds were presented. Mayer et al. (2013) discussed one of the case studies of the ASCENS project, which was a vision of an autonomic cloud: A cloud which was based on voluntary computing and using peer-to-peer technology to provide a platform-as-a-service. It used self-awareness and self-adaptation as the main ingredients for managing the execution of arbitrary applications. However, many aspects of this technology required further research, such as self-adaptation performance in the cloud, large-scale tests, alternative implementation models, etc.

Elasticity is an important feature of cloud computing and can be understood as how a computational cloud fits variations in their workload by provisioning and de-provisioning resources. Autonomic computing brings many concepts quite useful in the construction of elastic cloud computing solutions, such as control loops and thresholds-based rules. Coutinho et al. (2015) proposed an elastic architecture for cloud computing based on concepts of autonomic computing. Konstanteli et al. (2014) proposed a mechanism using probabilistic optimisation model, for admission control of a set of horizontally scalable services. Their model reduced the resources required to assure a given quality of service by employing statistical knowledge of the elastic workload requirements of services.

4.1.1. Service oriented computing

Along with the popularity of the Internet, a great amount of attention has centred on service-oriented computing (SOC). SOC is the computing paradigm that utilises services as fundamental resources for developing applications. Because services provide a uniform and standard information paradigm for a wide range of computing devices, they will be vital in the next phase of distributed computing development. The developers can compose existing web service components to create new applications for complex service requirements.

Applications in service-based systems can often be viewed as the composition of various computing services following specific workflows. Techniques based on Service Oriented Architecture (SOA) to enable utility computing have emerged and become a cost-effective way for organisations to outsource their computing

tasks to infrastructure providers and receive computing services on-demand. Yau et al. (2007) proposed a software cybernetics approach, by modelling and solving timing and resource constraints for deploying and scheduling workflows.

Liu et al. (2009) proposed a control-based approach to the security adaptation problem in adaptive service-based systems. A performance index that incorporated security requirements and delayed deadlines was proposed to transform the problem into an optimisation problem. An example application using the proposed security technique was implemented to demonstrate the feasibility of the approach. The experimental data showed that the system provided a desirable balance between security and delay requirements.

Cloud applications are typically composed of multiple cloud service components communicating with each other through web service interfaces, where each component fulfils specified functionalities. Lack of effective fault tolerance scheme is one of the major obstacles for enhancing availability and efficiency of complex and ageing cloud application systems. Liu et al. (2015) proposed an adaptive failure detection and ageing degree evaluation approach to predict which cloud service components deserved foremost to be rejuvenated and a component rejuvenation approach based on checkpoints with trace replay was proposed to guarantee the continuous running of cloud application systems.

4.1.2. Agent-based systems

There is comparatively little work found in the area of agent-based systems for software cybernetics, despite the nature of multi-agent systems seems very closely aligned to software cybernetics II. Some work has been discussed in search based software engineering area (Harman et al., 2012b; Harman et al., 2013). From the cybernetic viewpoint, an agent-based system consists of a population of individuals that interact and share information, seeking to solve a common goal, in which the potential for cybernetics in the area of software agents is enormous.

Sim (2012) introduced an agent-based paradigm for the design and construction of software tools and testbeds for resource management in cloud computing. The contributions of his work included: an agent-based search engine for cloud service discovery, a developing agent-based negotiation mechanism that could be used for helping service negotiation and commerce in the Clouds and finally, a distributed problem-solving technique for automating cloud service selection and integration. Gutierrez-Garcia and Sim (2013) proposed a self-organising framework for negotiating agents to perform cost-effective cloud service selection and integration. Their research promoted self-organisation for agents in the service composition environment by proposing a semi-recursive contract net protocol, based on FIPA's contract net protocol, enhanced with service capability tables to keep track of feasible service contractors. A game theoretic approach was adopted by Sim (2015) to facilitate the formation of InterCloud coalitions and his work devised a four stage cloud to cloud interaction protocol and strategies for InterCloud agents. Mathematical proofs showed the InterCloud coalition formation strategies converge to a subgame perfect equilibrium and every cloud agent in an InterCloud coalition received a payoff equal to its shapely value.

4.2. Software cybernetics for cyber-physical system

Cyber–physical systems (CPSs) are the next generation of engineered systems in which computing, communication and control technologies are tightly integrated (Kim & Kumar, 2012). Cyber-physical Systems are integrations of computation and physical processes. Embedded computers and networks monitor and control the physical processes usually with feedback loops where physical processes affect computations and vice versa (Lee, 2008). Kim and Kumar (2012) overviewed CPS research in many relevant research domains such as networked control, hybrid systems, real-time computing, real-time networking, wireless sensor networks, security, and model-driven development. Being able to deal with the increasing complexity of software systems as triggered by cyber-physical systems or large scale distributed systems requires fundamentally new models and approaches in software engineering. The economic and societal potential of such systems is vastly greater than what has been realised, and major investments are being made worldwide to develop the technology (Bellavista et al., 2014).

4.2.1. Network systems

The constant growth in IT is making communication networking more and more complex to manage. One very promising solution is Software Defined Networking (SDN) that decouples the data and control planes, having one centralised controller for the network. This gives chances to control and manage the network as desired, thus opening many new possibilities (Adami et al., 2015). Ravindran (2014) applied software cybernetics to manage adaptation behaviour of complex network systems, in which model-based software techniques were employed to assess the quality of adaptation in a network system in the presence of uncontrollable external environment conditions. A cyber-physical system (CPS) based software structure was provided to evaluate the non-functional attributes of the output behaviour, and the network and algorithm parameters were justified automatically. The advantages of the CPS-style structure of an adaptive network system were that it reduced the development cost of distributed control software via software reuse and modular programming. The CPS-style structure also enabled easier system evolutions: i.e., adding or modifying the controller functionality, without weakening the system correctness. Adami et al. (2015) built a system to enable QoS control and routing in Software Defined Networks. When the OpenFlow controller installed a rule for a flow, it also took care of placing it in the right queue. The experimental results showed the system behaving as expected, managing in a more efficient way the network resources and giving guarantees about traffic handling.

Nakano (2011) reviewed various biological materials and mechanisms that could be exploited to create network systems. Common characteristics of such network systems were summarised as: (1) small scale and functionally complex, (2) biocompatibility, (3) energy efficiency, and (4) self-assembly. Biological systems presented fascinating features, such as autonomy, scalability, adaptability, and robustness, and the concepts and mechanisms were successfully applied to network systems design. Key design principles were summarised as: (1) massive numbers of redundant components, (2) local interactions and collective behaviour, (3) stochastic or probabilistic nature, and (4) feedback-based control.

4.2.2. Internet of Things

The Internet of Things (IoT) implies a wide set of intertwined and interconnected devices and things to provide value to stakeholders. The Internet of Things has become a reality with the emergence of Smart Cities, populated with large amounts of smart objects that are used to deliver a range of citizen services. The IoT paradigm relies on the pervasive presence of smart objects or "things", which raises a number of new challenges in the software engineering domain: Orchestrating smart objects at a large scale, service discovery, data gathering, data processing, etc. Perera and Vasilakos (2016) suggested how IoT resources could be described using semantics so as to enable resource discovery. To achieve this, a knowledge driven approach was proposed in their research referred to as Context-Aware Sensor COnfiguration Model (CASCOM) to simplify the configuration of IoT middleware platforms.

4.3. Software cybernetics in big data technology

Big data is an emerging technology that has attracted the attention of many researchers and practitioners in industrial systems engineering and cybernetics (Choi et al., 2016). Large and variegated data from business transactions, social media, and the Internet of Things is estimated to grow at 30 to 60% per year. In order to make good decisions by enabling automatic control, big data must be captured, processed, integrated, analysed, and archived, which leads to valuable knowledge for users. An approach to integrating architecture analysis and design language (AADL), Modelicaml, and Hybrid Relation Calculus for the development of big data driven cyber-physical systems was proposed in Zhang (2014). The proposed method was further used to specify and model the Vehicular Ad-hoc NETwork (VANET). Choi et al., (2016) analysed the challenges and opportunities of big data analytics and examined the reliability, security, and their operational risk management. Chang (2015) presented system design, development, and analysis on Social Cloud to ensure a smooth delivery of big data processing. The cybernetics functions ensured that 100% job completion rate for big data processing on Social Cloud with no costs involved. This offered a unique contribution for cybernetics to meet big data research challenges. However, as much as previous work has discussed the relationship of big data and cybernetics, but little has been proposed about how to model and better analyse the big data with principles and techniques in cybernetics.

4.4. Software cybernetics and creative computing

Software Cybernetics emphasises controlling software while Creative Computing emphasises being creative, in the whole software life cycle. It seems that they conflict each other. Nevertheless, controlling and being creative are almost the two most important aspects of software development and evolution. Yang and Zhang (2014) discussed how best to combine these two aspects in improving software by proposing models for controlling software behaviours and the process of software development. Accordingly, their research proposed the application of cybernetic and creative rules to the software development process. An application of formal rules in both domains presented a pragmatic approach to designing a quality user interface for a computing device. Furthermore, their work also described the benefits of applying principles from software cybernetics and creative computing to controlling the behaviour of software. By classifying software behaviour into functional and non-functional properties, ideas from the two fields could be jointly applied to control software behaviour in a number of possible ways. For instance, while cybernetics is applied to guide the implementation of phases in the software process, exploratory creativity can be used to change software functionalities, e.g., communication, data manipulation, and scientific computation.

4.5. Analysis to techniques, applications and trends

Our survey classifies 70 quality research articles, this represents a summary of research papers published after the review undertaken by Cangussu et al. (2007), addressing research problems related to software cybernetics (see Table 1). The survey as presented in Table 1 includes research work published since then and its classification scheme mainly covers the year of publication, research activity, applied model or technique, cybernetic order/theme, and case study. As seen earlier in the paper, Fig. 1 shows the trend of growth in more recent publications related to software cybernetics II. In this section, a further and deeper analysis of the overall area is provided to show the relationships and trends of software cybernetics.

Our broad classification of current status in software cybernetics is software cybernetics I and II. The former is based on first-order cybernetics by applying negative feedback mechanism to the construction of software systems. From Fig. 1, we can see the research trend of transformation from the former to the later. As we know, with higher order cybernetics, lower order cybernetics contexts have merged to create more complicated systems. By introducing new elements of higher order cybernetics, the scope of software cybernetics can be evolved accordingly. This means that, theoretically, all the software systems can be evolved to four-order cybernetics level. The cybernetics order in Table 1 is the level of the designed systems. Table 2 includes some features that we used to judge the orders of software cybernetics. It should be noted that the features in the lower level are a subset of the features at a higher level. In this paper, only new features appear in the table.

It should be also noted that the observed systems in software cybernetics include the software process and the software itself. It is observed that the subject of software process research can be much mature. Normally, the application of software cybernetics in software process focuses mainly on feedback or optimisation. The term “control”, e.g. quality control, implies the potential application of feedback control in software cybernetics. Optimisation has also been well investigated in the area of search based software engineering. From the cybernetics viewpoint, a feedback mechanism is first-order and optimisation can be first-order or second-order level. Higher order cybernetics applications can only exist in more complicated software systems. The main features of these systems are hybrid cyber-physical, adaptive and autonomic, hierarchical distributed, data-driven, and smarter systems.

5. Conclusions

It is becoming clear that we are in an era of software pervasiveness. Modern products and services increasingly embed software or are customised, optimised or managed using software (Bellavista et al., 2014). Dynamic environments, rapidly changing requirements, unpredictable and uncertain operating conditions require a new mode of application development and deployment. Software and services need to become smarter, self-organised, sustainable, resource efficient, robust and safe in order to meet stakeholder demands. The software cybernetic research area presents new opportunities and challenges to the software engineering research community.

Over recent years, the increasing richness and sophistication of modern software systems have challenged conventional design time software modelling analysis and has led to many studies exploring non-conventional approaches. This paper concentrates on the transition from the first generation of software cybernetics to the second generation of software cybernetics, which is evolutionary, not revolutionary. Many of the discussed issues have been studied at least to some extent in the past but were typically not in the central spotlight of the new cybernetics. It is our hope that researchers from artificial intelligence, game theory, cloud computing, creative computing, big data, IoT, cyber-physical system and other pertinent research areas to come together and work towards the establishment of a solid foundation that can be used in practice for effective and efficient development of modern complicated software, which may also give rise to new software engineering methods and tools.

To build autonomic and self-adaptive large-scale software systems, both software systems, and external environments need to be modelled so that they can understand or even learn from each other to produce better responses to any changes. Artificial intelligence and software cybernetics might reunion in some way to achieve such kind of software systems at the third-order/fourth-order software cybernetics level.

Table 1
Comparative research on software cybernetics in the period of 2007–2016.

Author(s)	Year	Research activity	Aim/ objectives	Software engineering perspective(s)	Model specification/ approach	Case study	Cybernetic dimension	Software cybernetics theme
Liu, H. et al	2016	verification of program relationships relying on software behaviour (VPRB)	Improving software reliability	Software implementation/ Verification	SBM/ PARM	Light Control System (LCS)	First order	Phase I
Li et al	2016	Business process management	Modelling software behaviour based on augmented event logs	Software evolution/ Implementation	Petri nets / Process discovery method	Patient registration system	Second order	Phase II
Ding et al	2016	Online transaction processing system (OTPS)	Synthesis of software controllers for monitoring and adapting performance parameters	Software implementation	Rule-based (Fuzzy logic)	Oracle 11 g database	Third order	Phase II
Choi et al	2016	Business operations and risk management	A survey	Software project management	Big data analytics	-	Second order	Phase II
Liu, L. et al	2016	Requirement software engineering	Control framework for user data driven requirement elicitation	Requirement elicitation	Data analytics	Netease Youdao Dictionaries (Online)	Second order	Phase II
Perera & Vasilakos	2016	Internet of Things (IoT)	Knowledge-based Resource discovery	Software design/ implementation	CASCOM	Global Sensor Networks (GSN)	Second order	Phase II
Chang	2015	Social networking (Big data)	Big Data analytics for Social Networks	Software implementation	Big Data cybernetics	Facebook	Second order	Phase II
Liu et al	2015	Service-oriented computing (SoC)	Improving the availability of Cloud applications	Software rejuvenation	fault tolerant scheme, Web Services	-	Second order	Phase II
Filieri et al	2015	Control design process for self-adaptive systems	Steps for Self-adaptive controller synthesis	Software design	Control architectural approach	Real-time video encoding	First order	Phase I
Coutinho et al	2015	Autonomic computing	Elastic control mechanism for Cloud computing	Software design	Algorithmic		Second order	Phase II
Singh & Chana	2015	Resource management in cloud computing	A review	Software design	QoS-aware	-	Second order	Phase II
Park	2015	Software engineering project	Goal-driven adaptive software engineering	Software process	Algorithmic (Essence-based)	Learning Management system (LMS)	Second order	Phase II
Baeten & Markovski	2015	Autonomic control software	Supervisory control theory roles	Software design/Implementation	Model-driven approaches	MRI scanner + other examples	First order	Phase I
Alvares et al	2015	Autonomic software	Behavioural model-based control	Software design/ Implementation	DSL called Ctrl-F (FSA)	Znn.com (platform for self-adaptive systems)	First order	Phase I
Adami et al	2015	Communication network management (Cyber-physical systems)	Quality of service control	Software design/ Implementation	Software-defined networking (SDN) + Floodlight	Mininet (simulator)	Second order	Phase II
Abuseta & Swesi	2015	Self-adaptive systems	Testing framework	System design/ Architecture	Knowledge-based approach + UML	-	Second order	Phase II
Zhao et al	2014	Software systems	Behavioural modelling	System design	EFSA	CVS client	First order	Phase I
Zhang et al	2014	Random testing	Software testing framework	Software testing	DRT-h	Software under test (SUT)	Second order	Phase II
Zhang	2014	Cyber-physical control systems	Modeling framework	Software design/ Implementation	AADL, Modelicaml + Hybrid Relation Calculus	VANET	Second order	Phase II
Yang & Zhang	2014	Creative computing	Control and being creating	Software development lifecycle	Rule-based	Syzygy surfer + other examples	Second order	Phase II
Ravindran	2014	Network systems (Cyber-physical systems)	Adaptive behaviour in complex network systems	Software design	CPS-based approach	QoE-aware Video transport	Second order	Phase II
Rammig et al	2014	Real-time software (Cyber-physical systems)	Self-adaptive control	Software design/ Architecture	VMM-approach	Power PC 405 architecture	Second order	Phase II

(*continued on next page*)

Table 1 (*continued*)

Author(s)	Year	Research activity	Aim/ objectives	Software engineering perspective(s)	Model specification/ approach	Case study	Cybernetic dimension	Software cybernetics theme
Kundu & Mukherjee	2014	Software maintenance	Efficient resource usage based on SLA	Software development project	Algorithmic (based on bipartite graph matching)	(CLRMS – Closed loop resource management system)	Second order	Phase II
Kumar & Naik	2014	Software systems	Autonomic control	Software design/ Implementation	Algorithmic	Dummy applications	First order	Phase I
Insaurralde & Vassev	2014	Unmanned space vehicles	Autonomic control	Software design	ASSL approach	BepiColombo Mission	Second order	Phase II
Ahuja & Dangey	2014	Autonomic computing	Survey	Software development	-	-	Second order	Phase II
Vinnakota	2013	Cyberspace	Framework for cyber-security	Software design/ architecture	Cybernetic framework approach	-	Second order	Phase II
Ravindran & Rabby	2013	Network systems (Cyber Physical Systems)	Adaptive intelligence in network systems	Software design/ Implementation	CPS-based approach	Multi-source video transfer	Second order	Phase II
Park & Yeom	2013	Situation-aware software	Validation of SWRL rules	Software verification/Validation	Logic- based approach	“Fire situation”	Second order	Phase II
Mayer et al	2013	Cloud computing	Enabling autonomic cloud	Software design/ Architecture	SCEL-software component ensemble language	ASCENS	Second order	Phase II
Liao et al	2013	Concurrency programming	Eliminating bugs via control synthesis	Software design	Gadara nets (a special class of Petri nets)	Algorithm	first order	Phase I
Gutierrez-Garcia & Sim	2013	Agent-based Cloud service composition	Cloud services composition	System design	AI, semi-recursive contract net protocol	Three Simulation experiments	Third order	Phase II
Agepati et al	2013	Software ageing & rejuvenation	Software rejuvenation	System design/ Implementation/Evolution	Algorithmic (MDP-approach)	Web server	First order	Phase I
Adams et al	2013	Cybersecurity	New paradigms in cyber security based on cybernetics	Software architecture/ Implementation	-	Web Browsing	Third order	Phase II
Abeywickrama et al	2013	Self-adaptive systems	Design and Implementation of SIMSOTA	Software design/Implementation	SOTA model	E-Mobility	Second order	Phase II
Wang & Cai	2012	Extended finite state machines	Supervisory control synthesis	Software verification/ Implementation	Algorithmic (EFSM)		First order	Phase I
Wang et al	2012	Self-adaptive software systems	Control design	Software verification/Implementation	rule-based (Descriptive Language)	Oozie workflow engine	Second order	Phase II
Souza	2012	Self-adaptive systems	Requirement-based design for Adaptive system	Requirement engineering	Awareness requirements (AwReqs and EvoReqs)	CADs	Second order	Phase II
Shankar	2012	Software process	Competence building in people in Software industries	Software project management	Model-based	Pilot studies	Second order	Phase II
Ponisio & van Eck	2012	Software process	Metric based control	Software project management	Framework approach	-	Second order	Phase II
Patikirikorala et al	2012	Self-adaptive software systems	A survey on control engineering approaches	Software design	Taxonomy	-		
Liu et al	2012	Self-adaptive software systems	Modelling feedback loops	Software design	Problem-oriented approach	Cruise control system	Second order	Phase II
Kandjani et al	2012	Software development projects	Reducing the complexity of Global software developments	Software design/ architecture	Extended axiomatic design theory	-	Second order	Phase II
Harman	2012	Software engineering	Artificial intelligence role	Software engineering/ Process	Survey of AI techniques	-	Second order	Phase II
Buyya et al	2012	Cloud computing	Autonomic management	Software design	Autonomic iterative optimisation	Dengue fever prediction application	Second order	Phase II
Yang et al	2011	Mission critical software	Self-adaptive framework	Software design	Fuzzy logic	MCS	Second order	Phase II
Okamura & Dohi	2011	Software systems	Reinforcement learning for software rejuvenation	Software design	Algorithmic- (semi MDP)	Garbage collection for application	First order	Phase I

(*continued on next page*)

Table 1 (*continued*)

Author(s)	Year	Research activity	Aim/ objectives	Software engineering perspective(s)	Model specification/ approach	Case study	Cybernetic dimension	Software cybernetics theme
Nakano	2011	Network systems (Cyber-physical systems)	Biologically Inspired Systems	Software design	A review	-	Second order	Phase II
Ionescu	2011	Cloud computing	Self-management	Software design	A position paper	-	Second order	Phase II
Gaudin & Bagnato	2011	Software systems	Supervisory control	Software maintenance	FSM	Basic calculator	First order	Phase I
Gao et al	2011	Software evolution	Requirement modelling	Requirement elicitation	CPDN and SERPM		First order	Phase I
Forsyth et al	2011	Self-managing software systems	Environment modelling	Software design/ Maintenance	Learner classifier system and Genetic algorithms	3D social worlds	Second order	Phase II
Machida et al	2010	Cloud computing	Software rejuvenation in VMM	Software design	Gradient search method	Cold VM and Warm VM rejuvenation	Second order	Phase II
Peng et al	2010	Software systems	Value-based feedback control	Software design	Algorithmic (PID Control theory)	Web-based system	Second order	Phase II
Donaires	2010	Complex software systems	Software process control	Software design	Process model approach (based on Viable system model)		First order	Phase I
Ye et al	2009	Adaptive testing	Complexity in parameter estimation	Software testing	Parameter estimation methods (GA and RLSE)	Space program	Second order	Phase II
Co et al	2009	Cyber awareness and security	Improving the resilience of software	Software implementation	Process-level software approach	Tamper detection system and memory error detection system (MEDS)	First order	Phase I
Liu et al	2009	Service based systems	Balance trade-off between service performance and security	Software design	3-tier intelligent control approach (with optimisation)	Application based on ASBS prototype‘	Second order	Phase II
Julius & Pappas	2009	Stochastic hybrid systems	Approximations	Software verification	Stochastic bisimulation function approach	Brownian motion alongside other examples	First order	Phase I
Li et al	2009	Avionics system	Adaptive control software	Software design/Architecture	Self-evolving scheduling algorithmic approach(using formalised model)	Flight control system	Second order	Phase II
Cao & Chen	2009	Software project	Risk control	Software project management	Particle swarm optimisation	A computable example	Second order	Phase II
Brun et al	2009	Self-adaptive systems	Engineering feedback loops	Software design/ Verification/ Maintenance	A review of current and future approaches	-	-	-
Chen et al	2009	Software development	Control in software verification	Software verification/ Implementation	Model-checking techniques (of FSA)	-	First order	Phase I
Lorenzoli et al	2008	Software systems	Generation of software behaviour models	Software verification/ Implementation	GK-Tail Algorithm (EFSM)	Shopping cart	First order	Phase I
Hu et al	2008	Software reliability	Adaptive software testing	Software testing	Improved CMC approach	Space program	First order	Phase I
Cai et al	2008	Software reliability	Experimental study for adaptive testing	Software testing	Comparative analysis with random testing and operational profile based testing	Space program	First order	Phase I
Yau et al	2007	Service-based systems	Deploying and scheduling workflows	Software design/ Implementation	Logic-based technique	AS^3 logic	Second order	Phase II
Solomon et al	2007	Autonomic systems	Real-time reference architecture design	Software design/ Architecture	Modular architectural approach	-	First order	Phase I
Liu et al	2007	Software process	Goal-oriented requirement process	Requirement elicitation	A position paper	-	-	-
Girard & Pappas	2007	Discrete and continuous systems	Metrics approximation	Software verification/ Validation	Algorithmic (Transition systems)	Deterministic and non-deterministic continuous systems	First order	Phase I
Dobson et al	2007	Autonomic systems	Closed-form specification	Software design/ Architecture	Model –derived approach	Braking system	First order	Phase I
Cai et al	2007	Software reliability	Adaptive testing	Software testing	Fixed-memory feedback approach	Space Program	First order	Phase I
Bai et al	2007	Reliability in Web services	Adaptive testing	Software testing	Test broker(Control architecture) approach	-	First order	Phase I

Table 2
Orders of software cybernetics.

Level	Defining characteristics/System features
First order	Negative feedback; 'Self-steering' is isolated from the act of observation;
Second order	Positive and negative feedback; Interaction between observer and observed; Supervisory control; 'Self-steering' is affected by observer; Cybernetics of Cybernetics; Agent; Autonomous; self-adaptive; optimisation; creative
Third order	Active-interactive; context-aware; Co-evolution;
Four order	Self-awareness of the observer; System is contextualised, embedded and integrated into the context; Meta-system; Self-regeneration; Self-healing; Co-defining context; Redefine itself;

References

Abeywickrama, D.B., Hoch, N., Zambonelli, F., 2013. SimSOTA: engineering and simulating feedback loops for self-adaptive systems. In: ACM International Conference on Computer Science and Software Engineering, pp. 67–76. doi:10.1145/2494444.2494446.

Abuseta, Y., Swesi, K., 2015. Towards a framework for testing and simulating self-adaptive systems. In: 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp. 70–76. doi:10.1109/ICSESS.2015.7339008.

Adami, D., Donatini, L., Giordano, S., Pagano, M., 2015. A network control application enabling software-defined quality of service. In: IEEE International Conference on Communications (ICC 2015), pp. 6074–6079. doi:10.1109/ICC.2015.7249290.

Adams, M.D., Hitefield, S.D., Hoy, B., Fowler, M.C., Clancy, T.C., 2013. Application of cybernetics and control theory for a new paradigm in cyber security. CoRR arXiv preprint arXiv:1311.0257.

Agepati, R., Gundala, N., Amari, S.V., 2013. Optimal software rejuvenation policies. In: IEEE conference on Reliability and Maintainability Symposium (RAMS 2013), pp. 1–7. doi:10.1109/RAMS.2013.6517695.

Ahuja, K., Dangey, H., 2014. Autonomic Computing: An emerging perspective and issues. In: IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT 2014), pp. 471–475. doi:10.1109/ICICICT.2014.6781328.

Alvares, F., Rutten, E., Seinturier, L., 2015. Behavioural model-based control for autonomic software components. In: IEEE International Conference on Autonomic Computing (ICAC 2015), pp. 187–196.

Ashby, W.R., 1956. An Introduction to Cybernetics. Chapman & Hall, London.

Baeten, J., Markovski, J., 2015. The role of supervisory controller synthesis in automatic control software development. Sci. Comput. Programm. 97, 17–22. doi:10.1016/j.scico.2013.11.016.

Bai, X., Chen, Y., Shao, Z., 2007. Adaptive web services testing. In: 31st Annual IEEE International Computer Software and Applications Conference (COMPSAC 2007), pp. 233–236. doi:10.1109/COMPSAC.2007.53.

Bay, J.S., Heck, B.S., 2003. Software-enabled control: an introduction to the special section. IEEE Control Syst. Mag. 23 (1), 19–20. doi:10.1109/MCS.2003.1172826.

Bellavista, P., et al., 2014. Software engineering key enabler for innovation. NESSI Working Group: Networked European Software and Services Initiative White Paper.

Belli, F., Cai, K.Y., DeCarlo, R., Mathur, A., 2006. Introduction to the special section on software cybernetics. J. Syst. Softw. 79 (11), 1483–1485. doi:10.1016/j.jss.2006.03.037.

Brooks, F.P., 1987. No silver bullet: essence and accidents of software engineering. IEEE Comput. 20, 10–19. doi:10.1109/MC.1987.1663532.

Brun, Y., et al., 2009. Engineering self-adaptive systems through feedback loops. In: Software Engineering for Self-Adaptive Systems, Springer Berlin Heidelberg, pp. 48–70. doi:10.1007/978-3-642-02161-9_3.

Buyya, R., Calheiros, R.N., Li, X., 2012. Autonomic cloud computing: open challenges and architectural elements. In: Third International Conference on Emerging Applications of Information Technology (EAIT2012), pp. 3–10.

Cai, K.Y., 2002a. Optimal software testing and adaptive software testing in the context of software cybernetics. Inf. Softw. Technol. 44 (14), 841–855. doi:10.1016/S0950-5849(02)00108-8.

Cai, K.Y., Chen, T.Y., Tse, T.H., 2002b. Towards research on software cybernetics. In: 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02), p. 240. doi:10.1109/HASE.2002.1173129.

Cai, K.Y., Cangussu, J.W., DeCarlo, R.A., Mathur, A.P., 2003. An overview of software cybernetics. In: Eleventh Annual IEEE International Workshop on Software Technology and Engineering Practice, pp. 77–86. doi:10.1109/STEP.2003.4.

Cai, K.Y., Wang, X.Y., 2004. Towards a control-theoretical approach to software fault-tolerance. In: Fourth International Conference on Quality Software (QSIC 2004), pp. 198–205. doi:10.1109/QSIC.2004.1357961.

Cai, K.Y., Gu, B., Hu, H., Li, Y.C., 2007. Adaptive software testing with fixed memory feedback. J. Syst. Softw. 80 (8), 1328–1348. doi:10.1016/j.jss.2006.11.008.

Cai, K.Y., Jiang, C.H., Hu, H., Bai, C.G., 2008. An experimental study of adaptive testing for software reliability assessment. J. Syst. Softw. 81 (8), 1406–1429. doi:10.1016/j.jss.2007.11.721.

Cangussu, J.W., Mathur, A.P., DeCarlo, R.A., 2001. Feedback control of the software test process through measurements of software reliability. In: 12th International Symposium on Software Reliability Engineering (ISSRE 2001), pp. 232–241. doi:10.1109/ISSRE.2001.989477.

Cangussu, J.W., DeCarlo, R.A., Mathur, A.P., 2002. A formal model of the software test process. IEEE Trans. Softw. Eng. 28 (5), 782–796. doi:10.1109/TSE.2002.1027800.

Cangussu, J.W., Cai, K.-Y., Miller, S.D., Mathur, A.P., 2007. Software cybernetics. Wiley Encyclopedia Comput. Sci. Eng. doi:10.1002/9780470050118.ecse707.

Cao, P., Chen, F., 2009. A risk control optimization model for software project. In: International Conference on Computational Intelligence and Software Engineering (CiSE 2009), pp. 1–4. doi:10.1109/CISE.2009.5362886.

Chang, V., 2015. A Cybernetics social cloud. J. Syst. Softw. 1–17. doi:10.1016/j.jss.2015.12.031.

Chen, J., Zhang, Q., Bruda, S.D., 2009. Cybernetics in software system verification. In: International Conference on Intelligent Human-Machine Systems and Cybernetics, 2009 (IHMSC'09), pp. 274–277. doi:10.1109/IHMSC.2009.192.

Choi, T., Chan, H., Yue, X., 2016. Recent development in big data analytics for business operations and risk management. IEEE Trans. Cybern. (99) 1–12. doi:10.1109/TCYB.2015.2507599, PP.

Co, M., Coleman, C.L., Davidson, J.W., Ghosh, S., Hiser, J.D., Knight, J.C., Nguyen-Tuong, A., 2009. A lightweight software control system for cyber awareness and security. In: 2nd International Symposium on Resilient Control Systems (ISRCS'09), pp. 19–24. doi:10.1109/ISRCS.2009.5251353.

Cotroneo, D., Natella, R., Pietrantuono, R., Russo, S., 2011. Software aging and rejuvenation: where we are and where we are going. In: IEEE Third International Workshop on Software Aging and Rejuvenation (WoSAR 2011), pp. 1–6. doi:10.1109/WoSAR.2011.15.

Coutinho, E.F., Gomes, D.G., de Souza, J.N., 2015. An autonomic computing-based architecture for cloud computing elasticity. In: IEEE Latin American Network Operations and Management Symposium (LANOMS 2015), pp. 111–112. doi:10.1109/LANOMS.2015.7332681.

Dijkstra, E.W., 1972. The humble programmer - ACM turing lecture 1972. Commun. ACM 15 (10), 859–866. doi:10.1145/355604.361591.

Ding, Z., Wei, Z., Chen, H., 2016. A software cybernetics approach to self-tuning performance of on-line transaction processing systems. J. Syst. Softw. doi:10.1016/j.jss.2016.03.012.

Dobson, S., Bailey, E., Knox, S., Shannon, R., Quigley, A., 2007. A first approach to the closed-form specification and analysis of an autonomic control system. In: 12th IEEE International Conference on Engineering Complex Computer Systems, pp. 229–237. doi:10.1109/ICECCS.2007.6.

Donaires, O.S., 2010. Programming in the complex: Cybernetic insights into software process and architecture. Syst. Res. Behav. Sci. 27 (6), 667–679. doi:10.1002/sres.1014.

Filieri, A., et al., 2015. Software engineering meets control theory. In: 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press, pp. 71–82.

Forsyth, H., Laws, A., Bendiab, A.T., 2011. Evolutionary environmental modelling in self-managing software systems. In: Developments in E-systems Engineering (DeSE), pp. 370–375. doi:10.1109/DeSE.2011.109.

Gao, T., Li, T., Xie, Z., Xu, J., Qian, Y., 2011. A process model of software evolution requirement based on feedback. In: International Conference on Information Technology, Computer Engineering and Management Sciences (ICM 2011), pp. 171–174. doi:10.1109/ICM.2011.183.

Gaudin, B., Bagnato, A., 2011. Software maintenance through supervisory control. In: IEEE Software Engineering Workshop (SEW), pp. 97–105. doi:10.1109/SEW.2011.20.

Geyer, R.F., von der Zouwen, J., 1978. Socio-cybernetics. Martinus Nijhoff, Leiden/Boston/London.

Girard, A., Pappas, G.J., 2007. Approximation metrics for discrete and continuous systems. IEEE Trans. Autom. Control 52 (5), 782–798.

Gutierrez-Garcia, J.O., Sim, K.M., 2013. Agent-based cloud service composition. Appl. Intell. 38 (3), 436–464. doi:10.1007/s10489-012-0380-x.

Harman, M., Jones, B.F., 2001. Search-based software engineering. Inf. Softw. Technol. 43 (14), 833–839.

Harman, M., 2006. Search-based software engineering for maintenance and reengineering. In: 10th European Conference on Software Maintenance and Reengineering (CSMR 2006), p. 1.

Harman, M., 2012a. The role of artificial intelligence in software engineering. In: First International Workshop on Realizing AI Synergies in Software Engineering, pp. 1–6. (IEEE Press).

Harman, M., Mansouri, S.A., Zhang, Y., 2012b. Search-based software engineering: trends, techniques, and applications. J. ACM Comput. Surv. (CSUR) 45 (1). doi:10.1145/2379776.2379787, SurveysArticle No. 11.

Harman, M., Lakhotia, K., Singer, J., White, D.R., Yoo, S., 2013. Cloud engineering is search based software engineering too. J. Syst. Softw. 86 (9), 2225–2241. doi:10.1016/j.jss.2012.10.027.

Heck, B.S., Wills, L., Vachtsevanos, G.J., 2001. Software enabled control: background and motivation (I). In: American Control Conference, pp. 3433–3438. doi:10.1109/ACC.2001.946161.

Heylighen, F., Joslyn, C., 2001. Cybernetics and second-order cybernetics. In: Meyers, R.A. (Ed.), Encyclopaedia of Physical Science & Technology, Third ed. Academic Press, New York.

Hu, H., Jiang, C.H., Cai, K.Y., 2008. Adaptive software testing in the context of an improved controlled Markov chain model. In: 32nd Annual IEEE International Computer Software and Applications (COMPSAC'08), pp. 853–858. doi:10.1109/COMPSAC.2008.186.

Huebscher, M., McCann, J., 2008. A survey of autonomic computing - degrees, models, and applications. J. ACM Comput. Surv. (CSUR) 40 (3). doi:10.1145/1380584.1380585.

Insaurralde, C.C., Vassev, E., 2014. Autonomic control architecture for avionics software of unmanned space vehicles. 33rd IEEE/AIAA Digital Avionics Systems Conference 8B3-8B1 doi:10.1109/DASC.2014.6979537.

Ionescu, D., 2011. Autonomic computing: the path towards controlling cloud computing services. 3rd IEEE International Symposium on Logistics and Industrial Informatics 11-11 doi:10.1109/LINDI.2011.6031133.

Joao, M.F., Simon, T., Jens, B.J., Oscar, R., 2007. Designing tool support for translating Use Cases and UML 2.0 sequence diagrams into a coloured petri net. In: International Workshop on Scenarios and State Machines, Scenarios and State Machines (SCESM, 2007), p. 2. doi:10.1109/SCESM.2007.1.

Julius, A.A., Pappas, G.J., 2009. Approximations of stochastic hybrid systems. IEEE Trans. Autom. Control 54 (6), 1193–1203. doi:10.1109/TAC.2009.2019791.

Kandjani, H., Bernus, P., Wen, L., 2012. Enterprise architecture cybernetics for complex global software development: reducing the complexity of global software development using extended axiomatic design theory. In: Seventh IEEE International Conference on Global Software Engineering (ICGSE '12), pp. 169–173. doi:10.1109/ICGSE.2012.19.

Kenett, R.S., 2011. Future directions of software cybernetics: A position paper. In: 35th IEEE Annual Computer Software and Applications Conference Workshops, pp. 43–44. doi:10.1109/COMPSACW.2011.18.

Kim, K.D., Kumar, P.R., 2012. Cyber–physical systems: a perspective at the centennial. IEEE Special Centennial Issue 1287–1308. doi:10.1109/JPROC.2012.2189792.

Konstanteli, K., Cucinotta, T., Psychas, K., Varvarigou, T.A., 2014. Elastic admission control for federated cloud services. IEEE Trans. Cloud Comput. 2 (3), 348–361. doi:10.1109/TCC.2014.2325034.

Kumar, K.P., Naik, N.S., 2014. Self-Healing model for software application. In: Recent Advances and Innovations in Engineering (ICRAIE), pp. 1–6. doi:10.1109/ICRAIE.2014.6909207.

Kundu, J., Mukherjee, A., 2014. Implementation of software cybernetics for efficient resource usage in software maintenance project. In: The Fourth Conference on Emerging Applications of Information Technology (EAIT), pp. 127–132. doi:10.1109/EAIT.2014.29.

Lee, E.A., 2008. Cyber-physical systems: design challenges. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369. doi:10.1109/ISORC.2008.25.

Li, C., Ge, J., Huang, L., Hu, H., Wu, B., Hu, H., Luo, B., 2016. Software cybernetics in BPM: modeling software behaviour as feedback for evolution by a novel discovery method based on augmented event logs. J. Syst. Softw. doi:10.1016/j.jss.2016.03.013.

Li, G., Du, C., Song, C., Cai, X., 2009. A self-evolving control method for software in complex avionics system. In: IEEE International Conference on Computational Intelligence and Software Engineering (CiSE 2009), pp. 1–4. doi:10.1109/CISE.2009.5365340.

Liao, H., Wang, Y., Stanley, J., Lafortune, S., Reveliotis, S., Kelly, T., Mahlke, S., 2013. Eliminating concurrency bugs in multithreaded software: a new approach based on discrete-event control. IEEE Trans. Control Syst. Technol. 21 (6), 2067–2082. doi:10.1109/TCST.2012.2226034.

Lin, P., MacArthur, A., Leaney, J., 2005. Defining autonomic computing: a software engineering perspective. In: 2005 Australian Software Engineering Conference, pp. 88–97. doi:10.1109/ASWEC.2005.19.

Liu, C., Jiang, C., Hu, H., Cai, K.Y., Huang, D., Yau, S.S., 2009. A control-based approach to balance services performance and security for adaptive service-based systems (ASBS). In: 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09), pp. 473–478. doi:10.1109/COMPSAC.2009.178.

Liu, C., Zhang, W., Zhao, H., Jin, Z., 2012. A problem-oriented approach to modelling feedback loops for self-adaptive software systems. In: 19th Asia-Pacific Software Engineering Conference (APSEC), pp. 440–445. doi:10.1109/APSEC.2012.77.

Liu, H., Liu, Y., Liu, L., 2016a. The verification of program relationships in the context of software cybernetics. J. Syst. Softw. doi:10.1016/j.jss.2016.01.031.

Liu, J., Zhou, J., Buyya, R., 2015. Software Rejuvenation based fault tolerance scheme for cloud applications. In: 8th IEEE International Conference on Cloud Computing (CLOUD), pp. 1115–1118. http://dx.doi.org/10.1109/CLOUD.2015.164.

Liu, L., Jin, Z., Lu, R., 2007. Towards controllable requirements engineering processes based on cybernetics. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), pp. 229–232. doi:10.1109/COMPSAC.2007.221.

Liu, L., Zhou, Q., Liu, J., Cao, Z., 2016b. Requirements cybernetics: elicitation based on user behavioural data. J. Syst. Softw. doi:10.1016/j.jss.2015.12.030.

Lorenzoli, D., Mariani, L., Pezzè, M., 2008. Automatic generation of software behavioural models. In: 30th International ACM Conference on Software Engineering, pp. 501–510. doi:10.1145/1368088.1368157.

Machida, F., Kim, D.S., Trivedi, K.S., 2010. Modeling and analysis of software rejuvenation in a server virtualized system. In: Second IEEE International Workshop on Software Aging and Rejuvenation (WoSAR), pp. 1–6. doi:10.1109/WOSAR.2010.5722098.

Mayer, P., et al., 2013. The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing. In: 7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW), pp. 89–94. doi:10.1109/SASOW.2013.16.

Miller, S.D., DeCarlo, R.A., Mathur, A.P., Cangussu, J.W., 2006. A control-theoretic approach to the management of the software system test phase. J. Syst. Softw. 79 (11), 1486–1503. doi:10.1016/j.jss.2006.03.033.

Nakano, T., 2011. Biologically inspired network systems: a review and future prospects. IEEE Trans. Syst. Man Cybern. Part C 41 (5), 630–643.

Novikov, D.A., 2016. Cybernetics: From Past to Future. Springer.

Okamura, H., Dohi, T., 2011. Application of reinforcement learning to software rejuvenation. In: 10th International Symposium on Autonomous Decentralized Systems (ISADS), pp. 647–652. doi:10.1109/ISADS.2011.92.

Park, J., Yeom, K., 2013. A feedback-based approach to validate SWRL rules for developing situation-aware software. In: 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 41–46. doi:10.1109/COMPSACW.2013.21.

Park, J.S., 2015. Essence-based, goal-driven adaptive software engineering. In: IEEE/ACM 4th SEMAT Workshop on General Theory of Software Engineering (GTSE), pp. 33–38. doi:10.1109/GTSE.2015.12.

Patikirikorala, T., Colman, A., Han, J., Wang, L., 2012. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In: 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (IEEE Press), pp. 33–42. doi:10.1109/SEAMS.2012.6224389.

Peng, X., Chen, B., Yu, Y., Zhao, W., 2010. Self-tuning of software systems through goal-based feedback loop control. In: 18th IEEE International Requirements Engineering Conference (RE), pp. 104–107. doi:10.1109/RE.2010.22.

Perera, C., Vasilakos, A.V., 2016. A knowledge-based resource discovery for Internet of Things. Knowl. Based Syst. doi:10.1016/j.knosys.2016.06.030, (In Press).

Phoha, V., Nadgar, A., Ray, A., Phoha, S., 2005. Supervisory control of software systems. In: Quantitative Measure for Discrete Event Supervisory Control. Springer, New York, pp. 207–238. doi:10.1007/0-387-23903-0_8.

Ponisio, L., van Eck, P., 2012. Metrics-based control in outsourced software development projects. Softw. IET 6 (5), 438–450. doi:10.1049/iet-sen.2011.0199.

Rammig, F.J., Grosbrink, S., Stahl, K., Zhao, Y., 2014. Designing self-adaptive embedded real-time software–towards system engineering of self-adaptation. In: Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 37–42. doi:10.1109/SBESC.2014.15.

Ravindran, K., Rabby, M., 2013. Software cybernetics to infuse adaptation intelligence in networked systems. In: Fourth IEEE International Conference on the Network of the Future, pp. 1–6. doi:10.1109/NOF.2013.6724499.

Ravindran, K., 2014. Software cybernetics to manage adaptation behaviour of complex network systems. In: 23rd IEEE International Conference on Computer Communication and Networks (ICCCN), pp. 1–8. doi:10.1109/ICCCN.2014.6911727.

Sayyad, A.S., Ammar, H., 2013. Pareto-optimal search-based software engineering (POSBSE): a literature survey. In: 2nd IEEE International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), pp. 21–27. doi:10.1109/RAISE.2013.6615200.

Shankar, P.R., 2012. The cybernetics of enabling competence in people competence building to ensure quality and productivity in people in software industries. In: IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom), pp. 83–87. doi:10.1109/CyberneticsCom.2012.6381622.

Sim, K.M., 2012. Agent-based cloud computing. IEEE Trans. Serv. Comput. 5 (4), 564–577. doi:10.1109/TSC.2011.52.

Sim, K.M., 2015. Agent-based interactions and economic encounters in an intelligent InterCloud. IEEE Trans. Cloud Comput. 3 (3), 358–371. doi:10.1109/TCC.2015.2389839.

Singh, S., Chana, I., 2015. QoS-aware autonomic resource management in cloud computing: a systematic review. ACM Comput. Surv. (CSUR) 48 (3). doi:10.1145/2843889, Article No. 42.

Solomon, B., Ionescu, D., Litoiu, M., Mihaescu, M., 2007. Towards a real-time reference architecture for autonomic systems. In: IEEE International Workshop on Software Engineering for Adaptive and Self-Managing Systems, p. 10. doi:10.1109/SEAMS.2007.20.

Souza, V.E.S., 2012. A requirements-based approach for the design of adaptive systems. In: 34th International Conference on Software Engineering (IEEE), pp. 1635–1637. doi:10.1109/ICSE.2012.6227218.

Vinnakota, T., 2013. A cybernetics paradigms framework for cyberspace: key lens to cyber security. In: IEEE International Conference on Computational Intelligence and Cybernetics (CYBERNETICSCOM), pp. 85–91. doi:10.1109/CyberneticsCom.2013.6865787.

Von Foerster, H., 1979. Cybernetics of cybernetics. In: Klaus, Krippendorff (Ed.), Communication and Control in Society. Gordon and Breach, New York, pp. 5–8.

Wang, L., Gao, Y., Cao, C., Wang, L., 2012. Towards a general supporting framework for self-adaptive software systems. In: 36th Annual IEEE Computer Software and Applications Conference Workshops (COMPSACW), pp. 158–163. doi:10.1109/COMPSACW.2012.38.

Wang, P., Cai, K.Y., 2006. Representing extended finite state machines for SDL by a novel control model of discrete event systems. In: Sixth IEEE International Conference on Quality Software (QSIC 2006), pp. 159–166. doi:10.1109/QSIC.2006.53.

Wang, X.Y., Cai, K.Y., 2012. Supervisory control of a kind of extended finite state machines. In: 24th IEEE Chinese Control and Decision Conference (CCDC), pp. 775–780. doi:10.1109/CCDC.2012.6244119.

Wiener, N., 1948. Cybernetics: or Control and Communication in the Animal and the Machine. Technology Press, Boston, MA.

Xu, H., Sawyer, P., Sommerville, I., 2006. Requirement process establishment and improvement from the viewpoint of cybernetics. J. Syst. Softw. 79 (11), 1504–1513. doi:10.1016/j.jss.2006.03.050.

Yang, H., Chen, F., Zhou, Y., Zhao, M., Wang, Y., Guo, H., 2008. Software evolution for evolving China. In: Ordonez de Pablos, P., Lytras, M.D. (Eds.), The China Information Technology Handbook. Springer, pp. 1–33. doi:10.1007/978-0-387-77743-6_21, ISBN 9780387777429.

Yang, H, Zhang, L., 2014. Controlling and being creative: software cybernetics and creative computing. In: IEEE 38th International Computer Software and Applications Conference Workshops (COMPSACW), pp. 19–24. doi:10.1109/COMPSACW.2014.7.

Yang, Q., Lü, J., Xing, J., Tao, X., Hu, H., Zou, Y., 2011. Fuzzy control-based software self-adaptation: a case study in mission critical systems. In: IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 13–18. doi:10.1109/COMPSACW.2011.13.

Yang, Y., Gohari, P., 2005. Embedded supervisory control of discrete-event systems. In: IEEE International Conference on Automation Science and Engineering, pp. 410–415. doi:10.1109/COASE.2005.1506804.

Yau, S.S., Huang, D., Zhu, L., Cai, K.Y., 2007. A software cybernetics approach to deploying and scheduling. In: 11th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2007 (FTDCS'07), pp. 149–156. doi:10.1109/FTDCS.2007.7.

Ye, F., Liu, C., Hu, H., Jiang, C.H., Cai, K.Y., 2009. On the computational complexity of parameter estimation in adaptive testing strategies. In: 15th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'09), pp. 87–92. doi:10.1109/PRDC.2009.22.

Zhang, L., 2014. A framework to specify big data driven complex cyber physical control systems. In: IEEE international Conference on Information and Automation (ICIA), pp. 548–553. doi:10.1109/ICInfA.2014.6932715.

Zhang, L., Yin, B.B., Lv, J., Cai, K.Y., Yau, S.S., Yu, J., 2014. A history-based dynamic random software testing. In: IEEE 38th International Computer Software and Applications Conference Workshops (COMPSACW), pp. 31–36. doi:10.1109/COMPSACW.2014.9.

Zhao, X., Xue, J., Hu, C., Ma, R., Zhang, S., 2014. Research on software behavior modeling based on extended finite state automata. In: IET Communications Security Conference (CSC 2014), pp. 1–5. doi:10.1049/cp.2014.0744.

Zhu, H., 2012. Software cybernetics in the age of cloud computing - challenges and opportunities. Speech at the 9th IEEE International Conference on Software Cybernetics (IWSC '09).

Hongji Yang is a professor and deputy of creative computing, Centre for Creative Computing, Bath Spa University His current research interests include Software Engineering and Creative Computing. He received his BSc and MPhil from Jilin University, China in 1982 and 1985 respectively, and hid PhD from Durham University, UK in 1994. He served as a Programme Co-Chair at IEEE International Conference on Software Maintenance 1999 (ICSM '99) and is serving as the Programme Chair at IEEE Computer Software and Application Conference 2002 (COMPSAC'02). He is chief editing the International Journal of Creative Computing. He has published five books and well over 300 papers in Software Engineering, Computer Networking and Creative Computing. Prof Yang is an IEEE Computer Society Golden Core member and a recipient of the Meritorious Award with IEEE Computer Society.

Feng Chen received BSc, Mphil and PhD at Nankai University, Dalian University of Technology and De Montfort University in 1991, 1994 and 2007. He is a senior lecturer at De Montfort University. His research interests include software engineering, distributed computing, knowledge engineering and image processing.

Suleiman Aliyu received the bachelor of technology and master of science degrees in computer science from Abubakar Tafawa Balewa University, Bauchi, Nigeria in 2003 and 2010. He is currently working towards a doctorate degree in computer science at De Montfort University, Leicester, UK. His research interests include Inter-cloud computing, computational intelligence, and smart systems.