Control-Theoretical Software Adaptation: A Systematic Literature Review

Stepan Shevtsov, Mihaly Berekmeri, Danny Weyns, Martina Maggio

Abstract—Modern software applications are subject to uncertain operating conditions, such as dynamics in the availability of services and variations of system goals. Consequently, runtime changes cannot be ignored, but often cannot be predicted at design time. Control theory has been identified as a principled way of addressing runtime changes and it has been applied successfully to modify the structure and behavior of software applications. Most of the times, however, the adaptation targeted the resources that the software has available for execution (CPU, storage, etc.) more than the *software* application itself. This paper investigates the research efforts that have been conducted to make software adaptable by modifying the software rather than the resource allocated to its execution. This paper aims to identify: the focus of research on control-theoretical software adaptation; how software is modeled and what control mechanisms are used to adapt software; what software qualities and controller guarantees are considered. To that end, we performed a systematic literature review in which we extracted data from 42 primary studies selected from 1512 papers that resulted from an automatic search. The results of our investigation show that even though the behavior of software is considered non-linear, research efforts use linear models to represent it, with some success. Also, the control strategies that are most often considered are classic control, mostly in the form of Proportional and Integral controllers, and Model Predictive Control. The paper also discusses sensing and actuating strategies that are prominent for software adaptation and the (often neglected) proof of formal properties. Finally, we distill open challenges for control-theoretical software adaptation.

Index Terms—Self-Adaptive Software, Control Theory, Software Adaptation.

1 Introduction

Software applications are, more than ever, forced to deal with change [1], [2]. The need for continuous availability of software is forcing developers to consider change as part of the development process. Software should be able to execute in conditions that differ from the ones it was initially designed for, for example because new hardware is available with respect to what was envisioned at design time [3]. Moreover, software should execute with incomplete knowledge of the execution environment and conditions and face changing requirements during operation [4]. Consequently, software engineers are developing new techniques to handle change at runtime without incurring into penalties and downtime, giving birth to what is commonly referred to as software self-adaptation [5], [6].

Different alternative approaches have been proposed for the design of self-adaptive software, a prominent one being architecture-based adaptation [7], [8], [9], [10]. In the architecture-based approach, the software generates and updates an explicit architectural model of itself and uses it to reasons about adaptation. Applying classic techniques like testing and model checking for providing assurances at runtime is challenging, especially because these techniques assume the availability of accurate models of the software behavior. The partial knowledge available at design time represents a challenge for architecture-based approaches, in particular regarding the formal guarantees that can be provided [11], [12].

Self-adaptive software must deal with change at runtime, when the knowledge of how to handle this change becomes available. The software engineer includes mechanisms to handle runtime variations in the software design and implementation [13]. Most of these mechanisms use feedback from the software and the environment to adapt some part of the execution and ensure that the requirements are met under changing execution conditions. Control theory was identified as a discipline that could offer insight on the design of adaptation mechanisms with formal guarantees [14], [15], [16], [17].

So far, most research on control-theoretical adaptation of computing systems focused on controlling lower-level elements/resources of the technology stack (CPU, storage, bandwidth, etc.) [18], [19], [20]. With respect to the adaptation of resource allocation, applying control theory to adapt the software behavior is a more complex problem [21], [22], [23], due to the difficulty of accurately modeling software, to the types of requirements and their tradeoffs [24] and to the need of instrumenting software to obtain sensor measurements and actuators [25], [26].

Research efforts applying control-theoretical adaptation to software exist [21], [27], [28], [29]. However, the results of these efforts are scattered and consequently, there is no clear view on state of the art. This calls for a consolidation of the knowledge on the application of control-theoretical principles to software adaptation. Such knowledge would provide understanding of the basic engineering principles, including the software models and the control mechanisms, as well as the types of achieved goals and provided guarantees.

To systematize the mentioned knowledge, we performed

- *S. Shevtsov is with the Linnaeus University Sweden E-mail: stepan.shevtsov@lnu.se*
- *M. Berekmeri is with the Grenoble Institute of Technology France, D. Weyns is with Katholieke Universiteit Leuven Belgium and Linnaeus University Sweden, and M. Maggio is with the Lund University Sweden.*

Revised manuscript received February 13, 2017

a systematic literature review, following a well-defined methodology that identifies, evaluates and interprets the relevant studies with respect to specific research questions and topics of interest [30]. In the review, we have analyzed research results from 41 main conferences and journals in software/systems engineering, adaptive systems and control theory, in the period 2000-2016. The focus of the study is on three different aspects: models, control strategies and formal guarantees.

More precisely, in software engineering, models typically rely on architectural concepts, like components and connectors. In control theory, on the contrary, models are typically behavioral based – in the case of discrete event control – and equation-based – for discrete and continuous time control. One of the crucial topics of this survey is the role of models in control-theoretical software adaptation. The second topic this survey focuses on is control structures. In control theory, a controller structure is chosen based on the characteristics of the specific problem, like the presence or absence of model uncertainties or the required speed of convergence towards goals. Finally, in software engineering, development time techniques such as code reviews and model checking are usually coupled with runtime techniques like quantitative verification [31] to provide guarantees on the adaptation process. In control theory goals are usually expressed as setpoints and guarantees are expressed and obtained at design time, in terms of the ability to reach the desired objective whenever feasible. Guarantees are typically given on the model, and their validity is evaluated against model inaccuracies and parametric uncertainty.

The remainder of this paper is organized as follows: Section 2 provides information about the specific focus of the review, Section 4 provides some background on control theory, Section 5 contains information about related surveys and efforts, Section 6 discusses the research methodology used for this survey, Section 7 describes the findings of this survey. From the analysis, we have derived some insights that helped us to outline relevant challenges for future research, that are described in Section 8. Finally, Section 9 discusses threads to validity, Section 10 concludes the paper.

2 Focus of the Literature Study

This section describes the focus of the conducted literature review in detail. We distinguish between the software system being adapted, discussed in Section 2.1, and the control technique being applied, described in Section 2.2.

2.1 Software Adaptation

Control-theoretical adaptation was used in a variety of computing systems [15], [32] with different objectives. This systematic literature review focuses on *software adaptation*[1]. Software adaptation here refers to the actual adaptation of a running software application and to the adaptation throughout the software development live cycle, from requirements to design, construction, testing, deployment, software maintenance, and evolution. Figure 1 shows the typical layered structure of modern computing systems. Each layer is illustrated with the example elements from three domains: webservices, warehouse logistics, and cloud applications.

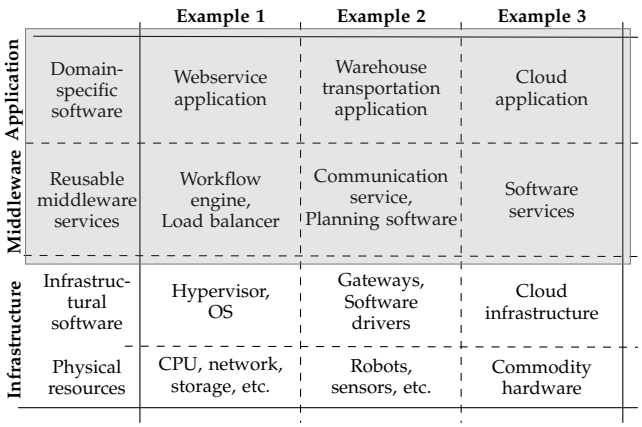


Fig. 1: Typical layering of modern computing systems. The focus of the review is on the layers marked in grey.

The lower part of the infrastructure layer includes physical resources such as CPU, storage, sensors and cloud hardware resources. The upper part of this layer incorporates infrastructural software; examples are hypervisors, software drivers, and cloud infrastructure. Adaptation at the infrastructure layer has been reviewed in the past [32], with a particular focus on resource provisioning techniques based on control theory. These approaches typically treat resources as flows and the control problem is often mapped to flow regulation [18], [33].

Adaptation of application software and middleware services is fundamentally different from adaptation performed at the infrastructure layer. The differences include:

- Software exhibits three possible adaptation dimensions: requirements, structure, and behavior [24];
- Context, goals and requirements are domain-dependent and can change during runtime [4], [13];
- There is a necessity to use complex and potentially multiple system models simultaneously [21], [25];
- The choice of proper sensors and actuators for adapting software can be challenging [16];
- The design space for the adaptation of software applications is often multi-dimensional [22], [23];
- Usually, there is a complex interplay between the qualities that are subject to adaptation on the one hand and the space of available adaptation options on the other hand [34].

This review includes studies that apply control to software elements, rather than to hardware resources or software infrastructure. The grey area in Figure 1 highlights the focus of this study. The focus avoids cluttered results that mix models, controllers, goals and guarantees for different layers of computing systems.

2.2 Control Techniques

The focus of this study is also restricted by the control techniques used for the adaptation system design. Figure 2

1. Adaptation refers to actions that lead to change of the software application, from architecture reconfiguration and component replacement to parameter changes.

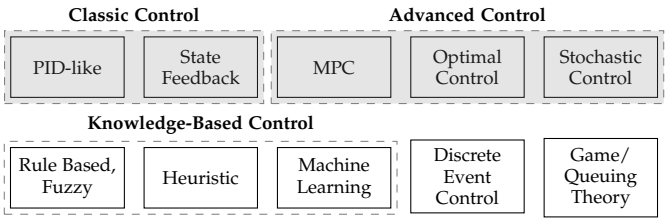


Fig. 2: Control-theoretical techniques. In grey the techniques that are considered in this literature review.

shows an extension of the taxonomy of potential control techniques proposed in [16], [35].

In this literature review, we analyze studies that use either classic or advanced control theory to adapt software systems. This includes the use of Proportional Integral Derivative (PID) controllers and controllers synthesized with pole placement or loop shaping techniques, Model Predictive Control (MPC) regulators, optimal controllers like Linear Quadratic Regulators (LQG), $\mathcal{H}_\infty$ controllers. We also delve into adaptive and stochastic control.

Despite the fact that knowledge-based control approaches are usually considered closely related to control-theoretical ones, the foundations of the two techniques are quite different. Knowledge-based strategies rely on building an ontology that is then used to decide what is the best strategy to achieve a specific goal [36], [37], while purely control-theoretical approaches rely on models. In the case of knowledge-based strategies – including fuzzy controllers, rule-based control, case-based reasoning, heuristics, and machine learning – the controller cannot rely on cases that it has not seen during its training phase. Equation-based models are in nature approximations of reality and the use of such models comes with the underlying assumption that the behavior of the system in a point in between observations can be interpolated based on the data available.

Logical control based on discrete event systems (DES) is a substantial part of control theory. However, in contrast to the classic control-theoretic approaches studied in this survey, the models applied in DES and the type of properties that can be proven based on these models are substantially different. In particular, DES relies on transition system models, such as Petri nets and timed automata [38], [39], [40]. Assessing the properties of such systems in software engineering community is performed with different formal methods and tools (e.g., reachability properties expressed in a particular logic using model checking). Hence, we excluded studies that apply DES and related approaches to realize adaptation. Finally, queuing theory and game theory have offered, in recent years, a basis for the development of different adaptation mechanisms [41], [42]. However, these mechanisms are quite different compared to the basic control-theoretical approaches and the type of guarantees that can be provided are different, making it difficult to compare approaches. Notably, studies that combine the use of queuing models with classic or advanced control strategies are still in the focus of our review.

Due to their different nature with respect to control-theoretic adaptation, we exclude knowledge-based, discrete event, game-theoretic and queuing-based approaches from this literature study.

3 Self-adaptive Systems Background

This section provides a brief background on self-adaptive systems based on [43]. We start with explaining the principles and motivation of self-adaption. Then we introduce basic concepts and illustrate the realization of the adaptation-specific elements with typical examples from a software engineering perspective. For additional reading, we refer the interested reader to [13], [44], [45], [46], [47], [48].

3.1 Principles of Self-adaptation

The term self-adaptation is not precisely defined in the literature. [5] refers to a self-adaptive system as a system that "is able to adjust its behavior in response to their perception of the environment and the system itself", [14] adds that "the self prefix indicates that the system decides autonomously (i.e., without or with minimal interference) how to adapt or organize to accommodate changes in its context and environment." These researchers take the stance of an external observer and look at a self-adaptive system as a one that handles changing external conditions and events.

[8] contrasts traditional adaptation mechanisms, such as exceptions in programming languages and fault-tolerant protocols, with mechanisms that are realized by means of a feedback loop to achieve various goals by monitoring and adapting system behavior at runtime. These authors state that a design with external feedback loops provide a more effective engineering solution for self-adaptation compared to a design with internal mechanisms. Initial evidence for this statement was provided in [49].

[22] refer in this context to "disciplined split" as a basic principle of a self-adaptive system, referring to an explicit separation between a part of the system that deals with the domain concerns and a part that deals with the adaptation concerns. Domain concerns relate to the goals for which the system is built; adaptation concerns relate to the system itself, i.e., the way the system realizes its goals under changing conditions. These researchers take the stance of a system engineer and look at self-adaptation from the perspective of how the system is conceived.

From these two perspectives, [43] identifies two basic principles that complement one another and determine what is a self-adaptive system: (1) the external principle: a self-adaptive system is a system that can handle changes in its context, the system itself and its goals autonomously (i.e., without or with minimal human interference), and (2) the internal principle: a self-adaptive system comprises two distinct parts: the first part interacts with the environment and is responsible for the domain concerns (concerns for which the system is built); the second part interacts with the first part and is responsible for the adaptation concerns (concerns about the domain concerns). The first principle takes the perspective of an external observer who considers a self-adaptive system as a black box and looks at its interaction with the environment, while the second principle takes the perspective of the engineer who considers the internals of a self-adaptive system and looks at its primary structure.

3.2 Conceptual Model of Self-adaptive Software

Figure 3 shows a conceptual model of a self-adaptive software system. It consists of four basic elements: environment,

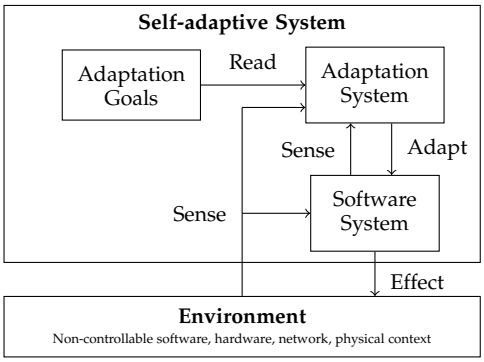


Fig. 3: Model of a self-adaptive software system.

software system, adaptation goals, and adaptation system. These basic elements are abstract and very general, i.e. they do not depend on a type of deployment, coordination between system components and the decision-making entity. A wide variety of approaches have been studied and applied that realize the basic elements in different ways. We illustrate the realization of the adaptation-specific elements (adaptation goals and adaptation system) with typical examples from a software engineering perspective.

Environment refers to the part of the external world with which the self-adaptive system interacts and in which the effects of the system will be observed and evaluated. The environment can include both physical and virtual entities. As the environment is not under control of the software engineer, there may be uncertainty in terms of what is being sensed or what will be the result of effecting actions. An example of the environment of a robotic system is the physical environment in which the robots can move, but also the drivers of the cameras that the robots use to sense its surrounding.

Software System comprises the application code that realizes the system goals for the domain at hand. To that end, the software system senses the environment and can effect the environment. For example, a robot can plan a path to perform a transportation task. During its mission, it can use a camera to detect obstacles, compute an alternative path if necessary, and steer the vehicle around obstacles to avoid collisions.

Adaptation Goals are goals of the adaptation system over the software system; they usually relate to qualities of the software system. [2] distinguishes between four types of high-level adaptation goals: self-configuration (i.e., systems that configure themselves automatically), self-optimization (systems that continually seek ways to improve their performance or cost), self-healing (systems that detect, diagnose, and repair problems resulting from bugs or failures), and self-protection (systems that defend themselves from malicious attacks or cascading failures). For example, a self-optimization goal of a robot may be to ensure that a particular number of tasks are achieved within a certain time window under changing operation conditions, e.g., dynamic task loads or reduced bandwidth for communication.

Adaptation goals are often expressed in terms of the uncertainty they have to deal with. Example approaches are the specification of quality of service goals using probabilistic temporal logics [31], and fuzzy goals whose satisfaction is represented through fuzzy constraints [50]. Adaptation goals are typically a first-class entities at runtime, enabling the adaptation system (see below) to reason about the adaptation goals during operation.

Self-adaption can support dynamic changes of the adaptation goals themselves. Such changes usually require the involvement of stakeholders. For example, in addition to the self-optimization goal, a new adaptation goal is dynamically added to a robot system that can handle the sudden loss of a power source. Dynamic changes of adaptation goals are not shown in Figure 3 as they require an evolution of the self-adaptive system, typically including an update of the adaptation system and probably also of the software system itself. An example approach that support dynamic changes of the adaptation goals is described in [51].

Adaptation System manages the software system. To that end, the adaptation system comprises adaptation logic that deals with the adaption goals. To realize the adaptation goals, the adaptation system senses the environment and the software system and adapts the latter when necessary. For example, to achieve the required number of tasks within a certain time window under peak load, the robots give priority to particular types of tasks. Conceptually, the adaptation system may consist of multiple layers where the upper parts manage the underlying subsystems.

The adaptation logic can be realized with different approaches. A classic approach applied in software engineering is to model the adaptation logic in the form of four components, Monitor, Analyze, Plan, and Execute that share common Knowledge (often referred to as MAPE-K [2]). The Monitor acquires data from the managed element and the environment, and processes this data to update the content of the Knowledge element accordingly. The Analyze element uses the up-to-date knowledge to determine whether there is a need for adaptation of the managed element. To that end, the Analyze element uses representations of the adaptation goals that are available in the Knowledge element. If adaptation is required, the Plan element puts together a plan that consists of one or more adaptation actions. The adaptation plan is then executed by the Execute element that adapts the managed element accordingly.

A key aspect of self-adaptation is to provide guarantees for the compliance of the adaption goals of self-adaptive systems that operate under uncertainty. A pioneering approach that deals with this challenge is quantitative verification at runtime. [31] applies this approach in the context of managing the quality of service in service-based systems. Extensive research has shown that providing guarantees for the compliance of the adaption goals with traditional software engineering approaches (ranging from traditional testing and sanity checks to model checking) remains a challenging problem [12]. This is one of the key reasons why researchers started exploring alternative paradigms such as the application of control theory to realize self-adaptation.

4 Control Theory Background

This section introduces some background on control theory, and defines the terminology that will be used for the analysis of the studies. For further reading on control theory, the reader can refer to [52], [53], [54], [55], [56], [57].

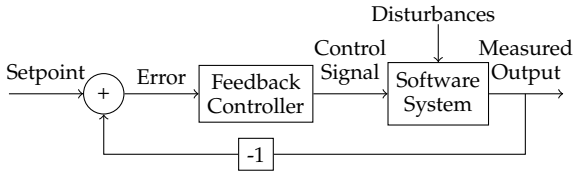


Fig. 4: Block diagram of a feedback control scheme

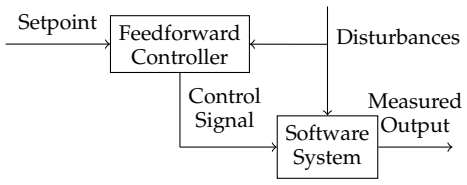


Fig. 5: Block diagram of a feedforward control scheme

4.1 Steady state and Transient Phase

In physical systems, when an input is applied to an object, this object usually reacts to the input. For example, if a person kicks a ball on a grass field, the force applied to the ball will make it move until a specific location. If one measures the position of the ball compared to the initial position, the signal will show a movement until the ball will stop (due to friction). The signal has clearly two distinct behavior. In a first phase (the transient phase), the ball will move, depending on the applied force. In a second phase, in absence of other forces, the ball position will settle to one specific location. This second phase is called steady state. A system is in steady state when the initial force applied has vanished its effects and it is in the transient phase while the effect of the initial force can still be observed. In general, the output signal of a system in the steady state is not necessarily a constant. For some systems, for example, the output can be a cyclic behavior.

As a parallelism with programming, one may think about a system in steady state as a piece of software, always repeating the same operations. If something happens in the software, some other routines can be started, to handle the interrupt. When these handling routines terminate, the software can go back to the original state of repeating the same operations.

4.2 Feedback and Feedforward Control

Figure 4 shows the basic block diagram of a feedback control scheme, applied to a software system. From left to right, the *Setpoint* represents the goal that the adaptation needs to achieve – typically a non-functional requirement such as a specific response time or a reliability value. Based on the value of the desired goal and the corresponding *Measured Output* an error is computed that is used by a *Feedback Controller* to compute the *Control Signal*. The control signal adapts the *Software System* such that the output gets as close as possible to the Setpoint. The *-1* block indicates that the value of the feedback signal is inverted, that is, the Error is computed as: $Setpoint + (-MeasuredOutput)$. During normal operating conditions the system reaches a steady state. When the measured output changes due to external *Disturbances*, the system enters a transient phase, where the feedback controller applies an appropriate control signal to handle the disturbances to bring the system back to the steady state. Figure 5 shows the basic block diagram of a feedforward control scheme. A *Feedforward Controller* takes into account the *Setpoint* and the values of external *Disturbances*, and produces a *Control Signal* that compensates for the disturbances.

To grasp the difference between feedback and feedforward control, imagine a person driving to a predefined destination. Feedforward control is the act of checking a map beforehand and memorizing it, computing the best strategy to get to the destination and applying this strategy when driving. Feedback control is the act of checking a navigation device that provides the current position and distance from the destination. A model of the map is still needed to define the direction, but this model is used during the navigation to refine the current navigation strategy.

In general, control strategies are developed to counteract the effect of disturbances on systems. In the case of software systems, these disturbances may come from the environment or from the internals of the software itself. The underlying assumption for the application of control is the ability to measure the output of the software behavior that must be kept under control. A measure of the disturbances, on the contrary, can be beneficial for the setup of a feedforward strategy, but is not necessary.

While the main purpose of feedback control has historically been disturbance rejection, the coupling of the feedforward block and the feedback one has the purpose of following a setpoint. Setpoint tracking is the other objective of the application of control.

4.3 Taxonomy of Classic and Advanced Controllers

In the blocks corresponding to feedback and feedforward controllers, one can implement different control strategies, ranging from classical control to more advanced techniques. Over the years, a lot of control techniques have been studied. The first group of techniques is generally called state-feedback controllers. These are controllers that use information about the state of the system to decide on a control signal [58]. One of the earliest strategies based on state feedback that has been developed is the bang bang controller, which consists in turning on or off a specific actuator, for example opening a valve to let water flow or closing it. In computing systems, this is usually the controller employed for admission control strategies, where requests are either admitted or rejected. Other state feedback controllers are regulators based on Pole Placement, Deadbeat Controllers and Proportional Integral and Derivative (PID) controllers. The PID controller is the most common controller and covers about 90% [58], [59] of the industrial applications of control. It is based on computing a control signal as a function of the error between the desired system behavior and the current system behavior.

The second group of techniques is called optimal control. In optimal control, the control value is obtained to minimize a cost function, possibly subject to some constraints. Typically, the objective is to maximize control performance, given prescribed guarantees [60], [61]. Whenever the cost

function is a quadratic function, and the constraints contain linear first-order dynamic constraints, the problem can be classified as a Linear Quadratic (LQ) optimal control problem. A special case is the Linear Quadratic Regulator (LQR) [62].

A particularly successful heuristic for optimal control under constraints is Model Predictive Control (MPC) [63], [64]. MPC predicts the future behavior from the current system state under a particular control action and selects the input sequence that minimizes the chosen cost function. Only the first step of that input sequence is applied and at the next time step the new system state is determined and the process repeated, according to receding horizon principle.

Together with these control strategies, there is Robust Control [60], which is based on building a control strategy that makes the system behave in a specific way, despite variation of involved parameters. In general, robustness to model inaccuracy is a property of all control strategies, but there are design techniques to develop controllers that are specifically aimed at maximizing robustness.

4.4 Composition of Controllers

Controllers can be composed by combining multiple feedback and/or feedforward controllers that interact with each other. For example, the feedback controller block may correspond to one of the following: multiple cascaded controllers; a hierarchical structure where the control signal is determined by controllers coupled together; controllers working in parallel or concurrently. When controllers are composed, the feedforward control signal is incremental with respect to any other control signal computed in the system (for example, from a feedback controller block). If no other controller is present then the feedforward control signal is applied directly to the software system. The main goal of combining feedback and the feedforward controllers is that the latter can take care of the part of disturbances that can be modeled, while the former can deal with disturbances that are not known a priori. The reader interested in composition schemes can consult [52].

5 Related Efforts

This literature review is not the first effort in trying to extract systematic knowledge from the research being conducted between the two disciplines of software engineering and control theory.

Most of the survey work on the subfield of *adaptive software* focuses on architecture-based adaptation [48], [65], where MAPE loops are usually considered the main technique to design an adaptation strategy and can be coupled with additional knowledge to reason about the software and the environment.

Motivated by the need for formal guarantees in the design of self-adaptive systems, researchers started to explore the application of principles from control theory to adapt computing systems, introducing the notion of "dynamic feedback" [66]. Seminal research in this direction is documented in the book by Hellerstein et al. [15], which highlights the potential of control theory for the adaptation of computing systems. As a result, a number of authors have further investigated the interplay between control theory and software engineering.

A pioneering article that elaborates on the application of control theory to software servers to provide guarantees for adaptation is [18]. Based on that and on subsequent works, control theory was considered as an approach that can be used in software engineering for the design of software that modifies its behavior at runtime providing formal guarantees about the mechanism used for the adaptation and about the goal satisfaction, whenever possible [14], [16].

While these studies can be useful in understanding the relationship between software engineering and control theory, they do not provide a comprehensive in-depth overview of the state of the art and they focus on adaptation at all the possible levels – as highlighted by the examples in Figure 1.

There have been a number of surveys in particular computing domains, for example [67] on mechanisms for performance management of Internet applications and [68] on quality-driven software adaptation using system properties derived from control theory to evaluate the usefulness of the adaptation. These surveys only investigated resource allocation and admission control, without delving into adaptation of the software behavior. A recent review of cloud service selection approaches did not identify any application of control theory for the adaptation of higher system layers in the cloud [69].

The work that is closely related to this survey is the systematic literature review on control-based adaptation of computing systems realized by Patikirikorala et al. [32]. The main result of that effort is a taxonomy that captures the characteristics of target and control systems, together with the types of validation performed to verify the effectiveness of the control mechanism. However, [32] does not distinguish between control-based adaptation at different layers of computing systems and treats low-level adaptation mechanism similarly to software adaptation. Low- and high-level adaptation are different in many aspects, the most important one being probably the availability of adequate physical models to guide the control design [70]. The analysis of low and high-level adaptation strategy lead to cluttered results that mix models, controllers, and guarantees of software adaptation with resource allocation, admission control, and hardware adaptation. From the results of this study it is therefore impossible to grasp the basic underlying principles that can be used for high-level software adaptation. Another problem of this survey [32] is the absence of data about a number of key characteristics and properties that are inherent to control theory. For example, the authors only collected data to classify system models based on their type – black box, first principle, queuing system –, while other essential model properties like linearity or nonlinearity and discreteness versus continuity were not examined. The same limitation applies to actuators and controller purposes – regulatory action, optimal control, disturbance rejection. The classification of controllers provided by the authors mixes control-theoretic concepts. For example, PID, LQR and MPC, which are the controller types, are mixed with cascaded, decentralized and hierarchical control, which are approaches to compose multiple controllers of one of

the mentioned types. Finally, the authors of [32] do not discuss the guarantees provided by the control-theoretical approaches, while formality is one of the main reasons to apply control theory [14], [15], [16], [53], [55].

In contrast to existing work, we perform a systematic literature review investigating control-theoretical adaptation of application software and middleware services of computing systems. We focus on adaptation based on classical or advanced control theory. This scope allows us to gain general insight and explore the use of control theory as a foundation for the design, analysis and verification of adaptive software.

6 Research Method

To conduct our systematic literature review, we followed the guidelines described in [30]. In a first stage, the team defined a protocol to be used for the review. The protocol includes (a) research questions, (b) a search string to find relevant sources, (c) inclusion and exclusion criteria to determine if a document that was found with the given string is relevant or not, and (d) relevant venues to be used as data sources. In the remainder of this section, we discuss these key elements.

Given these elements, we performed two independent searches in the documents retrieved with the search string applied to the relevant venues and compared the results and resolved the ambiguities and discrepancies arisen.

6.1 Research Questions

We first formulated the overall goal of our literature review using the Goal-Question-Metric approach [71]:

Purpose:	Understand and characterize
Issue:	the use of control theory
Object:	to adapt application software and supporting middleware services
Viewpoint:	from the standpoint of a researcher.

As control-theoretical software adaptation only recently emerged as a research field and there is currently no good overview of the field, the primary aim of this review is to create such an overview.[2] This overview will enable researchers to better compare and position specific contributions in the future.

We distilled the overall goal of the literature study in the following four research questions:

- *RQ1:* What is the current state of research on control-theoretical adaptation of software at the application and middleware level?
- *RQ2:* What are the model paradigms used for control-theoretical adaptation of software?
- *RQ3:* What are the control strategies used for control-theoretical adaptation of software?
- *RQ4:* What type of goals are achieved with control-theoretical adaptation of software and what kind of guarantees are provided?

RQ1 aims to provide a general overview of the state of the art in control-theoretical software adaptation. In particular, with RQ1 we can get insight in the trends of research on adapting software using principles from control theory. We plan to provide a deep understanding of the motivations for the use of control theory, of the viewpoint taken in its use, and of the approaches used to assess the effectiveness of the control approach.

The other questions are aligned with the "three broad areas of challenges in applying control theory to computing systems" mentioned by Hellerstein et al. [15, p.24]. These areas are: constructing models of the target system and controller, designing the feedback controller, and defining evaluation criteria to assess the results obtained.

Concretely, we formulated RQ2 to identify the models used for controlling software and their characteristics. RQ3 is related to the types of controllers and their different use, to the sensors and actuators applied, and to the methods used for building controllers. Finally, RQ4 helps us identifying the methods and metrics used to assess the effectiveness of the control solution and the guarantees provided.

6.2 Document Sources

To select the sources used for our systematic literature survey, we followed the same procedure used for other systematic studies, such as [65], [72]. The procedure starts with identifying the document sources that are used in related surveys [16], [32], [65]. The sources are then refined by consulting with researchers from both the field of control theory and software engineering.

After following the mentioned procedure, we identified the main venues for publishing research in control theory, software engineering and adaptive systems. To ensure high quality and obtain solid data to answer the research questions, we excluded a number of venues based on two parameters: the Australian Research Council (ARC) ranking[3] and the H-index[4]. Most of the included venues have high ARC rating (A*/A) and an H-index higher than 10. However, ranking alone is usually not conclusive. Therefore, we included a number of conferences and journals independent of their ratings because they are considered important in the respective communities.

In total, we included 41 venues: 15 journals and 26 conferences. For the journals we included 11 from control theory (CT), 3 from software/systems engineering (SSE), and 1 from adaptive systems (AS), see Table 2 for more detailed information. For the conferences, we included 4 from control theory, 17 from software/systems engineering, and 5 from adaptive systems, see Table 1.

6.3 Search Strategy

Our search strategy is composed by six different steps.

2. As explained above, control theory has been studied in the context of computing systems for over a decade mainly focusing on resource allocation and admission control. Control-theoretical adaptation of ***software*** on the other hand only recently emerged as a research field.

3. ARC for journals: http://research.unsw.edu.au/excellence-research-australia-era-outlet-ranking
ARC for conferences: http://103.1.187.206/core/

4. H-index for journals: http://www.scimagojr.com
For conferences: http://academic.research.microsoft.com/
Control venues: Google Scholar cat. *Automation & Control Theory*

TABLE 1: Conferences included in the search.

ID	Group	Venue	ARC	H-index
ICSE	SSE	International Conference on Software Engineering	A*	118
ICAC	SSE	International Conference on Autonomic Computing	B	32
DAC	SSE	Design Automation Conference	C	73
ICSM	SSE	International Conference on Software Maintenance and Evolution	A	56
ASE	SSE	Automated Software Engineering Conference	A	44
ESEC	SSE	European Software Engineering Conference	B	44
WADS	SSE	Workshop on Architecting Dependable Systems	n/a	30
VMCAI	SSE	Verification, Model Checking and Abstract Interpretation	B	30
WICSA	SSE	Working Conference on Software Architecture	A	25
CBSE	SSE	Symposium Component-Based Software Engineering	A	21
HASE	SSE	Symposium on High Assurance Systems Engineering	B	19
SEFM	SSE	Conference on Software Engineering and Formal Methods	B	18
ATVA	SSE	Symposium on Automated Technology for Verification and Analysis	A	14
QoSA	SSE	Conference on the Quality of Software Architectures	A	10
ECSA	SSE	European Conference on Software Architecture	n/a	8
FSE	SSE	International Symposium on the Foundations of Software Engineering	A	8
ESEM	SSE	Symposium on Empirical Software Engineering	A	n/a
CDC	CT	Conference on Decision and Control	A	45
ACC	CT	American Control Conference	n/a	40
ICARCV	CT	International Conference on Control, Automation, Robotics and Vision	A	11
ECC	CT	European Control Conference	n/a	20
SASO	AS	Self-Adaptive and Self-Organizing Systems	n/a	9
Adaptive	AS	Adaptive and Self-adaptive Systems and Applications	n/a	n/a
FeBID	AS	International Workshop on Feedback Computing	n/a	n/a
SEAMS	AS	Software Engineering for Adaptive & Self-Managing Systems	n/a	n/a
SefSAS	AS	Software Engineering for Self-Adaptive Systems	n/a	n/a

TABLE 2: Journals included in the search.

ID	Group	Journal	ARC	H-index
TSE	SSE	Transactions on Software Engineering	A*	128
JSS	SSE	Journal of Systems and Software	A	72
TOSEM	SSE	Transactions on Software Engineering and Methodology	A*	59
Automatica	CT	Automatica	A*	85
TAC	CT	Transactions on Automatic Control	A*	82
TCST	CT	Transactions on Control Systems Technology	A	54
IJRNC	CT	International Journal of Robust and Nonlinear Control	A	41
CEP	CT	Control Engineering Practice	B	38
SICON	CT	SIAM Journal on Control and Optimization	A*	36
IJC	CT	International Journal of Control	A	33
CS	CT	IEEE Control Systems	B	24
SCL	CT	Systems & Control Letters	n/a	41
ARC	CT	Annual reviews in control	n/a	27
CTA	CT	IET Control Theory & Applications	B	39
TAAS	AS	Transactions on Autonomous and Adaptive Systems	B	26

The first step is the definition and validation of the search string to be used for automated search. This process started with pilot searches on IEEE Explore and the ACM Digital Library. We combined different keywords from software engineering and control theory that are relevant for our research questions. Based on the pilot searches, we defined the following search string, that was then applied to title and abstract.

(control OR controller OR controlling) AND (adaptive OR self-adaptive OR adaptation OR self- OR autonomic OR autonomous) [AND (software)][5]

To validate the search string, we used a "quasi-gold standard" [73]. In particular, we manually searched through the proceedings of three known venues (TAAS, ICAC, and ICSE) during the past three years and found five studies that matched the selection criteria (discussed below). Then, we performed the automatic search in the proceedings of the same venues, using the search engines of the IEEE and ACM libraries. We refined the search string until the five studies were in the search results and the remaining number of the studies was minimal.

In the second step, we applied an automatic search using the previously defined search string. We use IEEE Explore, the ACM Digital Library and Google Scholar[6]. The search is performed on the venues described in Section 6.2. For venues not included in the digital libraries, we manually downloaded and searched the proceedings. After the automatic search, we collected a total of 1512 papers.

In the third step, two researchers independently read the abstracts of all studies selected in the previous step and used the inclusion and exclusion criteria described in Section 6.4 to filter out irrelevant papers. Of the 1512 papers selected with the automatic string match, only 161 papers were advanced to the next stage.

In step four, we read the complete papers to make a final decision on their inclusion in the review. Conflicts were resolved during extensive discussion. We excluded a various number of papers because they were not relevant, and had 40 studies to analyze at the end of this step. As a fifth step, we applied *snowballing*. We checked the references cited by the selected papers and included them when appropriate. We increased the number of studies to analyze to 61 papers.

Finally, in the sixth step, we identified and removed similar versions of the remaining papers. For example, when we found a conference and a journal version of the same paper, we kept only the journal version, as it is considered more complete and accurate. The final list of primary studies for our literature review consists of the following 42 references: [21], [24], [27], [28], [29], [34], [70], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108].

5. The additional keyword "software" is used only for control theory venues to improve the search results.

6. The reference search string was adjusted to match the search features provided by different electronic sources (e.g., different field codes, case sensitivity, syntax of search strings). The search string for Google Scholar was adjusted to *controller adaptive software* as the engine only allows searching on title or full text of papers.

6.4 Inclusion and Exclusion Criteria

We determined that a paper is approved for further analysis only when it satisfies all inclusion criteria and does not satisfy any of the exclusion criteria. We include studies that:

- Were published from January 2000 to June 2016. We used 2000 as starting date as adaptive systems have become subject of active research around that time [18], [66].
- Discussed the engineering of the adaptation strategy. The design or the implementation of the adaptation strategy or its parts must be included in the study.
- Matched the focus of the study, including adaptation of application software or middleware services, as shown in Figure 1.
- Applied classic or advanced control theory to design feedback loops, as shown by the grey area in Figure 2.

We excluded:

- Papers written in languages other than English.
- Tutorials, short papers, editorials because they do not contain sufficient data for our study.

6.5 Assessment of the Presentation Quality

Assessing the quality of the presentation – not necessarily related to the quality of the research – of the studies is important for the interpretation of the results.

To assess the presentation quality, we collected six quality items for each study. The quality items are listed in Table 3. These items are based on the quality assessment method for research studies initially described in [109] and adjusted in [110]. For each quality item we assign a value of 2 if the authors provide an explicit description, 1 if there is a general description, and 0 if there is no description at all. The paper quality assessment score (max 12 points) is calculated by summing up the scores for every quality item.

6.6 Extracted Data Items

Table 4 shows the data items that are extracted to answer the identified research questions. We here briefly explain the different items.

• F1-F5: These data items are used for documentation. For item F4 we additionally group venues into SSE (Software/Systems Engineering), CT (Control Theory) and AS (Adaptive Systems), as shown in Table 1. This data item is referred as F4.1.

• F6: Presentation quality score (on a total of 12), obtained as described in Section 6.5.

• F7: The engineering perspective taken by the authors of the study, which can be one of the following options: (a) *SE perspective:* The focus of these studies is on applying adaption to realize some quality requirements. The application of control-theoretical principles to adapt software is not well elaborated. For example, controller guarantees are not analyzed or the software mathematical model is not explicitly presented. (b) *CT perspective:* The focus of these studies is control theoretical aspects; software is basically used as an application domain. There is less focus on typical software engineering aspects. (c) *Integrated perspective:* These

TABLE 3: Quality items to assess the presentation quality of the studies.

Q1: Problem definition of the study	
2	Explicit problem description
1	General problem description
0	No problem description
Q2: Problem context of the study	
2	Explicit problem context supported by references
1	General problem context supported by references
0	No description of the context
Q3: Research design of the study	
2	Explicit description of how the research was organized
1	General words about the way the research was organized
0	No description of how the research was organized
Q4: Contributions/results of the study	
2	Explicit list of the study contributions
1	General words about the study contributions
0	No description of the study contributions
Q5: Insights derived from the study	
2	Explicit list of insights/lessons learned from the study
1	General words about the insights
0	No description of the insights derived from the study
Q6: Limitations of the study	
2	Explicit list of the study limitations
1	General words about the study limitations
0	No description of the study limitations

TABLE 4: Collected Data items.

Item ID	Field	Use
F1	Author(s)	Documentation
F2	Year	Documentation
F3	Title	Documentation
F4	Venue	Documentation
F5	Citations per year	Documentation
F6	Quality score	Documentation
F7	Engineering perspective	RQ1
F8	Motivation for CT	RQ1
F9	Validation	RQ1
F10	Assessment	RQ1
F11	Application domain	RQ1
F12	Claimed Generality	RQ1
F13	System model	RQ2
F14	Sensors and actuators	RQ3
F15	Triggers for adaptation	RQ3
F16	Controller type	RQ3
F17	Controller purpose	RQ3
F18	Guarantees	RQ4
F19	Software qualities	RQ4
F20	Tradeoffs	RQ4

studies employ principles from control theory to solve a software adaptation problem and exploit its mathematical foundation to analyze the system behavior and provide guarantees for quality goals.

• F8: Motivation for using control theory in a software system. The initial options are: formal guarantees, systematic approach, inefficiency of existing approaches. Additional options are derived during the review.

• F9: Validation setting is one of the following: academic effort, academic/industry collaboration, industrial effort, none.

• F10: The assessment approach used in the study. The initial options are: example application, simulation and discussion. In addition, we collect data about formal assessment (F10.1) which is one of the following: formal modeling, formal analysis, or none. By formal modeling we mean having a formal description of system model/controller, while formal analysis includes analysis of guarantees.

• F11: Applications domain for which adaptation is used or evaluated in the study. For example, e-commerce, tourism, video processing. The concrete application domains are derived during the review.

• F12: A boolean indicating whether the authors state a general applicability of the proposed approach.

• F13: The system model. Extracted data are divided into four sub-properties: (F13.1) model type, (F13.2) model linearity, (F13.3) time framework, (F13.4) model time dependency. For F13.1, a system can be denoted as (a) analytical, (b) grey box, or (c) black box. In an analytical model, the system is described by laws governing the behavior of that system (e.g., a Markov Chain). All model elements are known at design time (but parameters may change at runtime). With a grey-box model, the system is not entirely known, a certain model based on both insight in the system and experimental data can be constructed. However, the model has a number of unknown free parameters that are estimated using system identification. In the black-box case, the system is considered unknown but can receive input and produces some output, that in principle comes from unknown functions. For F13.2, a model can be either (a) linear or (b) non-linear. In the linear case, the output is directly proportional to the input. In the non-linear case, this direct proportionality is not true. F13.3 can be either (a) discrete- or (b) continuous-time. In a discrete-time model, a system is modeled using difference equations, while continuous-time models rely on ordinary differential equations. As for F13.4, the model can be either (a) time-dependant, or (b) time-invariant. In the first case, the dependency on time is explicit. The output o is computed using a function f that depends on the input i, on the state x, and on time t, $o(t) = f(i, x, t)$. In the second case, the model describes the output at some time advancement, but the relationship does not contain time $o(t) = f(i, x)$ and depends only on the state and the input.

• F14: Sensors and actuators. We separate collected data into: (F14.1) sensors: what is being measured during adaptation, (F14.2) actuators: the mechanism affecting software behavior to achieve the adaptation goals,

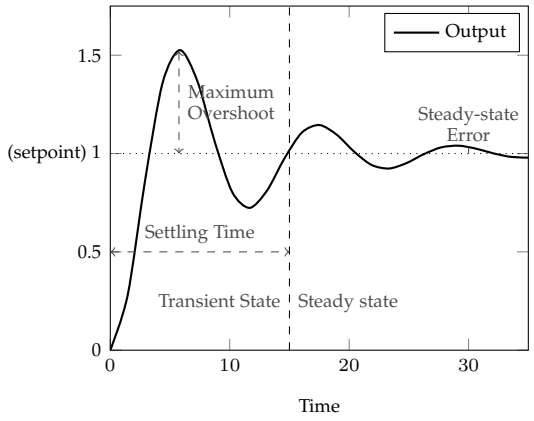


Fig. 6: Control-Theoretical Guarantees.

• F15: Triggers for adaptation. Can be one of the following: stimulations from the environment, changes in requirements/goals, changes in the software itself.

• F16: The controller type used in the feedback mechanism. Options include PID, MPC, optimal, and others. In addition, we collect the data about: (F16.1) adaptivity of controller: adaptive or non-adaptive, (F16.2) composition scheme of multiple controllers, if applicable. Options include cascaded, hierarchical, and others.

• F17: The controller purpose with options: optimization, regulatory functions (setpoint tracking), disturbance rejection, or a combinations of these purposes [15].

• F18: Formal guarantees provided by the use of control theory and described in the study. According to [15], control theory can guarantee four main system qualities: stability, steady-state error, settling time, and maximum overshooting. Stability refers to the ability of the system to converge to a fixed point (as opposed to diverging – for example, accumulating requests in a buffer). Steady-state error refers to the difference between the fixed point to which the system converged to and the desired goal, given to the controller. The settling time of a controller is a measure of how quickly the controller is able to reach the fixed point, when it exists. Finally, the maximum overshoot determines how much the maximum difference between the measured value and the objective will be, during the transient phase. A graphical summary of these properties can be seen in Figure 6. Additionally to the properties mentioned in [15], a number of studies discuss the guarantees of systems with respect to robustness. Robustness is the ability of the system to return to the steady state in case of model inaccuracies or perturbations and disturbances.

We also collect data about experimentally verified guarantees (F18.1). The difference with the data extracted in F18 (formal guarantees) is that the evidence is based on data that is collected from experiments.

• F19: Software qualities that are affected by adaptation and described in the study. We use the specification of qualities described in the ISO/IEC 9126-1 standard[7]. According to [65], the software engineering approaches mostly concentrate on: (a) performance, the ability of the software to achieve a desired value for qualities like throughput and response time; (b) efficiency, the extent to which the software uses the appropriate resources under stated conditions and in a specific context of use; (c) reliability: the capability of software to maintain its level of performance under stated conditions for a period of time; (d) other: software qualities such as scalability, usability, security, and portability.

7. ISO/IEC 9126-1 Software Eng. - Product quality - Part 1: Quality-model, Int. Standard Organisation. (2001)

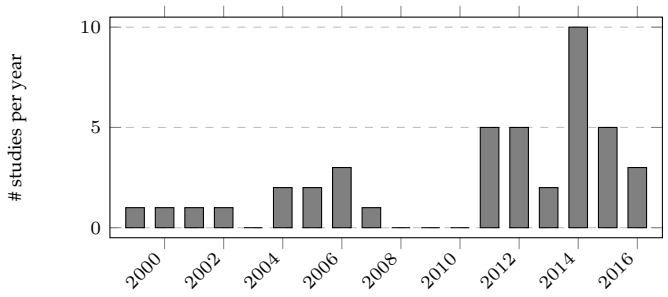


Fig. 7: F2: Number of analyzed primary studies by year.

• F20: Concerns that can be degraded as a consequence of improving other concerns. This can be one or several guarantees listed in F18 and/or qualities listed in F19.

7 Result Analysis

This section summarizes the data we collected from the identified 42 studies and presents an analysis of the results. We use descriptive statistics and plots for presenting the results. We first present demographics information and presentation quality assessment. Then, we answer the research questions stated in Section 6.1 based on the collected data.

7.1 Demographics

Figure 7 shows the frequency of primary studies per year (item F2).

Although we looked at papers from the past 15.5 years, we observed that 72% of primary studies were written in the last 5.5 years. This indicates that there is a growing interest in research on control-theoretical design of software. Several authors have argued that one important factor for this growing interest is the mathematical foundation of control theory that provides a solid basis for guaranteeing the adaptation goals under uncertainty [21], [29], [80], [83].

We also sorted studies according to the number of citations per year (item F5). Table 5 shows the primary studies with minimum 10 citations per year.

TABLE 5: Studies with minimum 10 citations per year.

Topic	Reference	Cit./year
Performance Control of Web Server	[88]	46.2
Brownout Paradigm	[70]	12.5
Push Button Methodology	[21]	11.5
DYNAMICO Reference Model	[103]	10.7

Our review revealed that the publication of primary studies is scattered over different venues: 25 studies were published at software/systems engineering venues, 7 at control theory related venues, 6 at adaptive systems

TABLE 6: Limitations reported in the primary studies

Limitations	Primary Studies
Requires specific conditions to function (pre-conditions)	[34], [84], [85], [88], [90], [98], [102], [106]
Requires additional computation resources or tools (redundancy)	[27], [84], [85], [88]
Not applicable in some cases/systems	[29], [34], [80], [84], [88], [90], [97], [98], [102]
External validity: generalising findings requires extra effort	[27], [29], [85], [94], [97]
Complexity of the proposed approach	[85]
Not able to handle new requirements	[24]

venues, and 4 at venues with other subjects. The only venue that published more than three of the studies was the Journal of Systems and Software with 6 studies. Having the majority of studies published at software/systems engineering venues, we can conclude that there is more interest in the software/systems engineering community in exploring the application of principles from control theory to realize adaptation of software as from the control theory community in applying novel research results to software applications.

Key insights from demographics:

- The interest in the research on control-theoretical adaptation raised significantly in the last 5.5 years.
- The publication of the primary studies is scattered over different venues.

7.2 Presentation Quality

The results of presentation quality assessment of the primary studies (item F6, Figure 8) show that the majority of the studies provide an in-depth description of the problem and the problem context, and most studies give a sufficiently clear description of contributions and insights. However, many studies do not describe the research design (methods, different steps, etc.) and lack a discussion of limitations of the proposed approach. This seems to be a general trend as similar results have been reported in other secondary studies and other domains, see e.g., [65], [72]. Nevertheless, the overall average score of 7.3 out of 12 points indicates a good quality of reporting in the studies, supporting the validity of the extracted data and the conclusions derived from them.

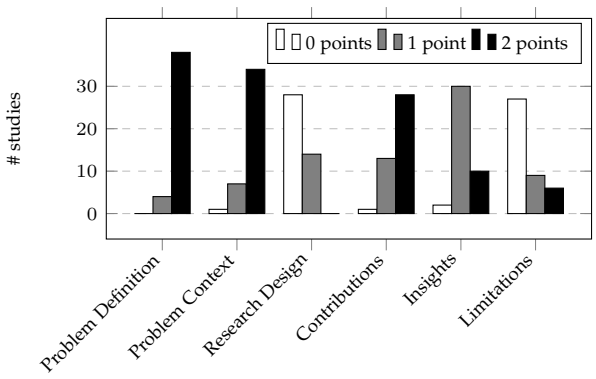


Fig. 8: F6: Presentation quality scores.

The particular limitations reported in the primary studies are summarized in Table 6. Notably, most of the limitations concern the applicability of the proposed adaptation mechanism (pre-conditions, redundancy, complexity). Only a few of the primary studies explicitly report threats to validity of the conducted study, such as internal validity, construct validity, and external validity.

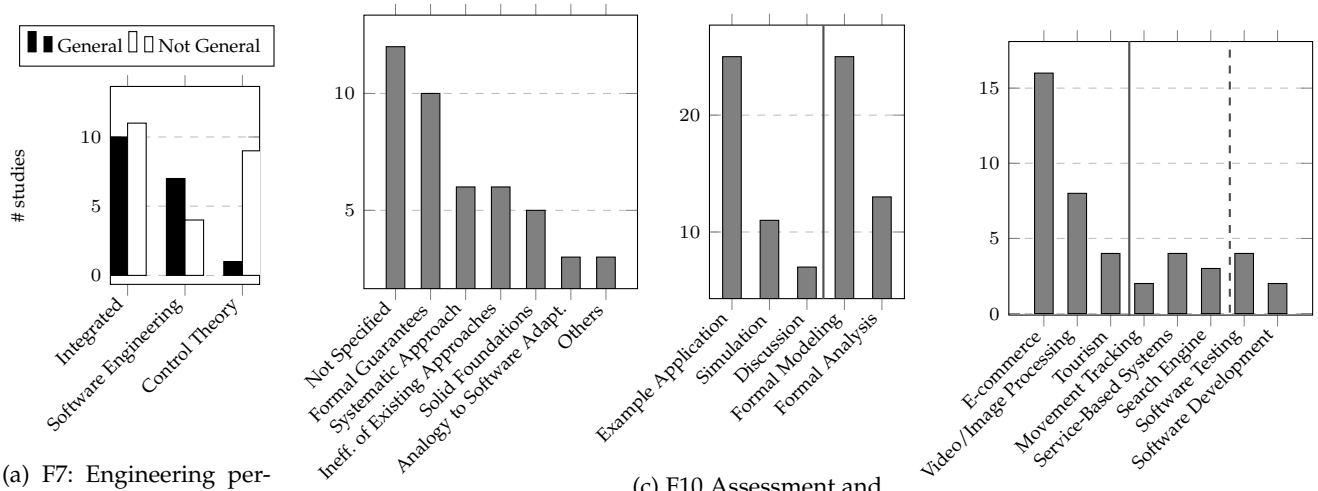
Key insights from presentation quality:

- Most of the primary studies provide a comprehensive description of problem and context, but lack a discussion of research design and limitations.

7.3 RQ1: Control-Theoretical Software Adaptation

To answer the first research question (what is the current state of research on control-theoretical adaptation of software at the application and middleware level?), we used data items F7-F12. Figure 9 provides an overview of the results. The engineering perspective taken in the studies varied (item F7, Figure 9a). 11 studies took a software engineering perspective. In these studies, particular attention was given to typical software engineering aspects, such as software qualities, design, testing, and similar concerns. The application of control theory to realize adaptation of software was not well elaborated. For example, guarantees provided by control theory were not analyzed and the software model and controller structure was not well defined. 10 studies took a control theoretic perspective. The focus of these studies contrasts to the software engineering perspective: attention was given to the formal part of the adaptation, the studies included an in-depth mathematical analysis of the model/controller. Software, in this case, was used as an application domain, typical software engineering aspects were not well elaborated. The remaining 21 studies [21], [24], [29], [34], [70], [76], [79], [80], [81], [82], [83], [88], [92], [94], [96], [98], [99], [104], [105], [106], [107] took an integrated perspective. These studies employed both software engineering and control theoretic aspects to realize adaptation of software.

The motivations to apply principles from control theory for adapting software varied (item F8, Figure 9b). The main motivations documented in the primary studies were formal guarantees, maturity ("systematic approach" and "solid foundation"), and effectiveness of control theory. These results support the rationale (discussed in Section 7.1)



(a) F7: Engineering perspective – F12: Claimed generality.

(b) F8: Motivation for using control theory.

(c) F10 Assessment and F10.1: Formal Assessment.

(d) F11: Application domain.

Fig. 9: Results for data items required to answer RQ1.

that software engineers are exploring new well-grounded approaches for engineering self-adaptive software driven by the need for guarantees. Note that 27% of the studies did not provide any motivation for applying control theory to software adaptation. We could not derive any conclusive data why the authors of these primary studies have not provided a motivation. Furthermore, there is no dominating trend in the motivations that are reported in the other studies (see Figure 9b). The motivations for applying control-theoretical adaptation of software may be an interesting topic for further investigation.

Validation of the research (item F9), except two industrial studies [107], [108], was based on academic efforts. As research of control-theoretical software adaptation is still in its early stages, most of the results have not yet found their way to practice. The most used assessment methods (item F10, Figure 9c) were example application, followed by simulation. These results are in line with the results presented in [32]. In 38 out of 42 primary studies formal modeling or analysis is conducted (item F10.1, Figure 9c). This is not surprising and confirms the appreciation of the formal underpinning of control theory to realize control-theoretical software adaptation. The concrete types of guarantees that are analyzed in the primary studies are discussed in Section 7.4.

The most popular application domains in the primary studies (Figure 9d) were web applications (E-commerce) and video/image processing software. The most used E-commerce applications were a flight reservation system described in [83] and the RUBiS benchmark[8]. Three studies used general web applications that show static content to user [88], [89], [90], while one study used recommender systems [101]. The applications in the video/image processing domain can be divided in different groups: object recognition [27], [91], video streaming [28], [104], video encoding [21], [81], [105], image/signal processing [78], [86].

Two abstract design/technology paradigms (service-based system and search engine) were included as six studies used these paradigms without describing a concrete application domain, e.g. [92], [93], [97] (Figure 9d between the dotted and full horizontal lines).

Finally, six studies applied principles from control theory not directly to adapt a running software application, but to support software development (Figure 9d). In particular, these studies applied control theory to calculate the human resources required for testing a software product [94], [95], to determine the quality of tests [96], [97], to select the appropriate types and number of test cases in order to minimize the number of software defects, to optimally distribute the development effort between construction and debugging [98], and to analyze the system lifetime based on the amount of development effort [99]. These studies may be especially interesting because they show that control-theoretical software adaptation is not only applied to end-products, but can be used in a broader way to adapt software artifacts during the development life cycle.

We observed that 18 primary studies stated general applicability of the proposed approach (item F12). It is notable that 7 out of 12 studies with software engineering focus (Figure 9a) proposed a generally applicable framework or methodology. On a contrary, only one study with a control theory perspective claimed the general applicability of the proposed approach [86]. These results support the tendency of research in the control engineering community to develop specific solutions for concrete problems, while in the software engineering community it is more common to aim for generally applicable solutions [8], [9], [10]. One of the main reasons to build controllers for specific problems in control theory is that generality comes with a tradeoff: generality of a controller typically implies some decrease in performance or robustness objectives [111]. Nevertheless, control theory offers a number of generic control structures (or patterns of controllers) and engineering techniques that enable these control structures to automatically adjust to specific scenarios [112].

As a side note, it is important to mention that the

8. Rice University Bidding System: http://rubis.ow2.org

TABLE 7: F13: System model.

Behavioral Model	Specific Model	Primary Studies	Additional Information
Analytical	Markov model	[78], [79], [96], [97]	Model used for stochastic systems for which future states depend only on the current state.
	Queuing network	[28], [80]	System is represented as a network of queues, which is evaluated analytically.
	Custom	[27], [91], [93], [98], [99], [107], [108]	Custom analytical models used for object recognition [27], [91], software development process [98], [99], search algorithm [93], ads [107], [108].
Grey box	Linear model Learned at Runtime (LLR)	[21], [29], [34], [70], [74], [75], [76], [77], [81], [82]	Model of the form: $u(k+1) = \alpha \times \eta(k) + d(k)$, where $u(k+1)$ is the system output, $\eta(k)$ is the actuation signal, α is a coefficient, $d(k)$ is a disturbance acting on the output. $u(k+1)$ and $\eta(k)$ are cases-specific; coefficient α is calculated during system identification based on a series of experiments [21], [70], [82] or using a controller [34], [81]. In some cases [21], [82] $d(k)$ is removed from the model, while α is updated during system operation to cope with system dynamics.
	Hammerstein-Wiener	[84]	Model that combines a non-linear block that captures the system non-linear behavior with a linear block responsible for all remaining system dynamics.
	Multi-Model Switching	[85]	Models of different types that can inter-replace each other during operation depending on the system goals.
	Custom	[24], [83], [86], [87], [88], [90], [94], [95], [101], [105], [106]	Custom grey-box models used for web server utilization [88], [90], software testing process [94], [95], resources allocation between software components [83], [105], recommender system [101], component interactions at the application layer [86], [87].
Black box	Custom	[89], [100], [104]	Models used with the aim to achieve generality of the proposed approach.
N/A	Not specified	[92], [102], [103]	No specification of the concrete model being used.

evidence for the general applicability of the proposed approaches in most of the primary studies is limited to the evaluation of a few examples or provided in form of discussion.

RQ1: Control-Theoretical Software Adaptation

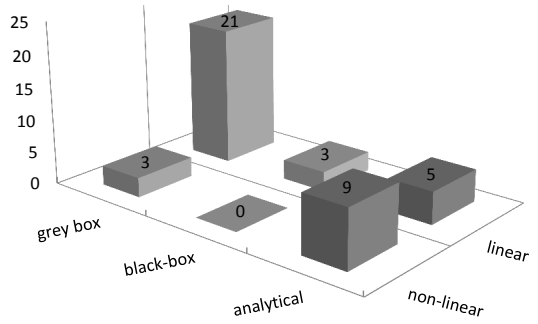
- The main motivations to use control theory in software adaptation are the maturity of the field and its formal foundation as a basis to provide guarantees.
- The most used application domains for control-theoretical software adaptation are E-commerce and video/image processing.
- Assessment of research contributions is based on (simple) example applications and simulations. There is a need for involving industry partners to evaluate control-theoretical solutions in practical settings.
- The studies with a software engineering focus typically propose a generally applicable methodology/framework, while studies focusing on control theory solve specific problems.

7.4 RQ2: Software Models

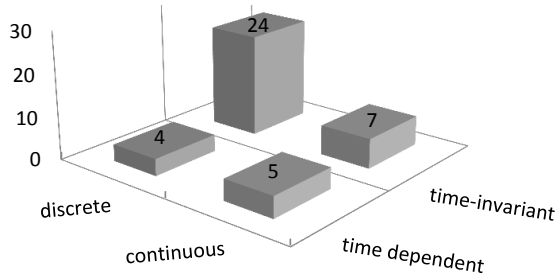
To answer the second research question (what are the model paradigms used for control-theoretical adaptation of software?), we used data item F13, see Figure 10.

We observed that different types of system models are used, but the dominating type is a linear, time-invariant, discrete grey-box model that is built using system identification techniques. The studies apply linear grey-box models for three reasons. First, these models can be easily designed, see for example [21], [79]. Black or grey-box models are preferred as it is often difficult to create a detailed analytical model of software since it is not governed by physical laws. And even if such model can be created, it may become inaccurate after the first software update. Moreover, the parameters of an analytical model must be updated at runtime to deal with changing operating conditions. Second, black or grey-box models offer a generic solution to system modeling. Whereas at the infrastructural layer CPU cores, memory and virtual machines can be easily abstracted for many systems types, at the software level it is problematic (or challenging) to find general elements that can be modeled. Each middleware software or each application has its own technology- and domain-specific software elements. Third, as stated in [21], [34], although linear grey-box models are not as accurate as complex non-linear models at design time, they are more effective at runtime due to a low level of complexity and a higher degree of guarantees that can be obtained using them. A common view on using a linear grey-box model is that as long as the model captures the general system dynamics, the inherent non-linearities of the system can be compensated by endowing the feedback controller with an adaptation or online model update mechanism [85], [89]. Table 7 provides an overview of models used in different studies.

Although most studies refer to the complexity of software systems and their non-linear behavior, there are only 11 studies that look at software as a non-linear system [78], [84], [86], [87], [93], [96], [97], [98], [99], [107], [108]. It is

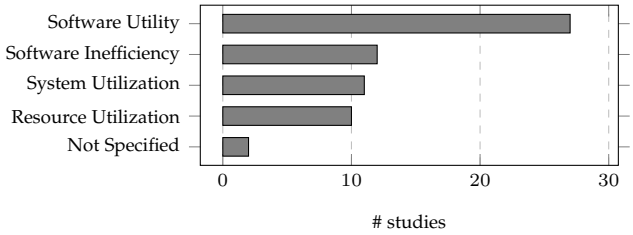


(a) F13.1: Model type vs F13.2: Model linearity.

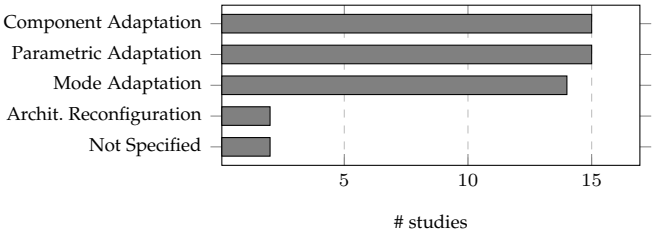


(b) F13.3: Time framework vs F13.4: Model time dependency.

Fig. 10: Results for different properties of system models.



(a) F14.1: Sensors.



(b) F14.2: Actuators.

Fig. 11: Sensors and Actuators.

notable that most of the non-linear models are analytical (Figure 10a). An explanation for this is that the identification of non-linear models is extremely challenging in terms of engineering effort; and there are almost no tools available to support the identification of non-linear models [54], [113].

When software applications undergo sudden changes in their behavior at runtime (for example a component failure), we observed two types of reactions in the primary studies: (1) updating model parameters [21], [27], [34], [82], [94], [95], [97] or even switching the model [85], and (2) updating parameters of the control law, which means using adaptive or model predictive control (further discussed in the following Section). In some cases, an update of the model may be followed by an update of control law as well [21], [82], [97]. Other approaches use a separate linear corrector to compensate for model changes [95], allow human operators to make a decision [94], or completely change the control law [27], [85].

RQ2: Software Models

- Linear, time-invariant, discrete grey-box models are mostly used in control-theoretical software adaptation.
- Although most of the authors discuss complexity and non-linear behavior of software, only 11 out of 42 primary studies employed non-linear models, most of which are analytical.
- Eight primary studies deal with behavioral changes at runtime by updating model parameters.

7.5 RQ3: Control Strategies

To answer the third research question (what are the control strategies used for control-theoretical adaptation of software?), we look at: monitoring mechanisms (sensors), effecting mechanisms (actuators), triggers for adaptation and controller types.

Sensors: When extracting data about monitoring mechanisms (sensors) and effecting mechanisms (actuators), we observed that the actual implementation of sensors (for example how the values are technically measured) and actuators (for example how the actuation mechanisms implement changes of the application) are discussed in only 7 of the 42 primary studies [34], [70], [88], [90], [92], [103], [105]. [88], [90] describe how bandwidth and request rate of an Apache Web server are measured, [70] employs PHP scripts to control the amount of optional content served to users and calculate user perceived latency, and [92] compares the influence of actuator realizations on the output of the target software.

In the rest of the primary studies, the authors refer to sensors and actuators as the monitored variables and variables effecting the application respectively. Consequently, we analyze only these variables in this literature review (and refer to them as sensors and actuators), due to the lack of data concerning the actual implementation of sensors/actuators in the primary studies.

We observed the use of various types of sensors in the primary studies (item F14.1, Figure 11a and Table 8a). The sensors can be classified in two main classes: sensors that monitor the software that is subject of adaptation and sensors that monitor elements that are external to the software application. We further distinguish two types of sensors that monitor the software application: those that monitor

TABLE 8: Specific sensors and actuators applied in primary studies.

(a) F14.1: Sensors.

Sensor Type	**Monitored Variable: Primary Studies** (the variables are representative examples)
Software utility	Software response to requests: [34], [77], [81], [82], [83], [84], [85], [104]
	Probability of correct object recognition: [27], [91]
	User perceived latency of application: [70], [74], [75], [76]
	Video quality and processing speed: [21], [86], [87], [105]
	Profit gained from the application: [99], [102]
Software inefficiency	Percentage of software failures: [21], [34], [78], [79], [82]
	Detection of software defect: [95], [96], [97]
	Number of errors in software: [94], [98], [102]
Resource utilization	Energy consumption: [34], [81]
	Cost of using external services: [34]
	Bandwidth: [88], [89], [90]
	CPU usage: [100], [105]
	Memory usage: [100], [104]
System utilization	Request arrival rate: [77], [86], [88], [89], [90]
	Length of requests queue: [28], [76], [77], [80], [87]
	Data to be processed by the system: [101]
	Amount of new user registrations: [102]

(b) F14.2 Actuators.

Actuator Type	**Changed Variable: Primary Studies** (the variables are representative examples)
Parametric adaptation	Length of a queue with pending requests: [76]
	Degree of video compression: [21]
	Quality parameters of a filter: [81], [105]
	Parameters to enhance testing quality: [94], [95]
Component adaptation	Load of software services: [21], [34], [78], [79], [82], [83], [84], [85]
	Distribution of incoming requests: [86], [87]
	Test case of application: [96], [97]
	Degree of parallelism to process requests: [104]
	Number of service instances: [28]
Mode change	Increment/decrement of content being served: [70], [74], [75], [76], [77]
	Change in search strategy: [93]
Mode switch	Video buffering scheme: [88], [89], [90]
	Quality of content representation of website: [21]
	Operating mode of system: [81], [105]
	Preference given to each service level: [80]
Architecture reconfigur.	Components change to handle variations in the task load: [100]
	Modules selected for execution to deal with changing goals: [102]

software utility and those that monitor software inefficiency. Sensors that monitor software utility measure the usefulness of the software to achieve its goals, such as the quality of video and the profit gained from the software application. Sensors that monitor software inefficiency measure the lack of ability of the software to achieve its goals, such as detection of software defects and errors in the software application. We also distinguish two types of sensors that monitor elements external to the software application: those that monitor resource utilization and those that monitor system utilization. The sensors that monitor resource utilization measure the amount of resources consumed by the software application to realize its goals, such as energy consumption and memory usage. Sensors that monitor system utilization on the other hand measure the degree of load on the application, for example as the length of request queues or the request arrival rate. Table 8a lists other examples of the different types.

The first class of sensors – those that monitor software utility and software inefficiency – are specific to control-theoretical software adaptation. These sensors have to be implemented by the software application or middleware services, for example using supporting functionality (framework API, component model, programming abstractions, and similar ones) or through a dedicated software interface. The second class – sensors that monitor elements that are external to the software application – are conventional types of sensors that are commonly used in control-based adaptation of computing systems at lower levels of the technology stack (physical resources and infrastructural software).

Control-theoretical software adaptation requires two types of sensors that respectively monitor the software application and the execution environment. These correspond to the types of sensors that are typically required for architecture-based adaptation of application software. Two studies that elaborate on this are [103] and [10].

Actuators: As for the actuators, we observed that the studies use a wide variety of effecting mechanisms to realise adaptation (item F14.2, Figure 11b and Table 8b). We identified four main types of actuators that operate at different levels of granularity: parametric, component, and mode adaptation, and architecture reconfiguration.

Parametric adaptation refers to changing the values of variables of the application software or middleware services. These types of actuators are typically domain-specific; examples are the degree of video compression and the length of a queue with pending requests that need to be processed. Component adaptation refers to changes at the level of software components, such as the load of services and the degree of parallelism that components process requests. Mode adaptation refers to a variation in the mode of operation, which can be either mode change or mode switch. An example of a mode change is an increment in the quality of content that is being served by a video application; an example of mode switch is an alteration of the buffering schema of a video application. Finally, architecture reconfiguration refers to a runtime adaptation of the architectural structure or behavior of the application. We only observed two instances of this type of effecting mechanism: changing components to handle variations in

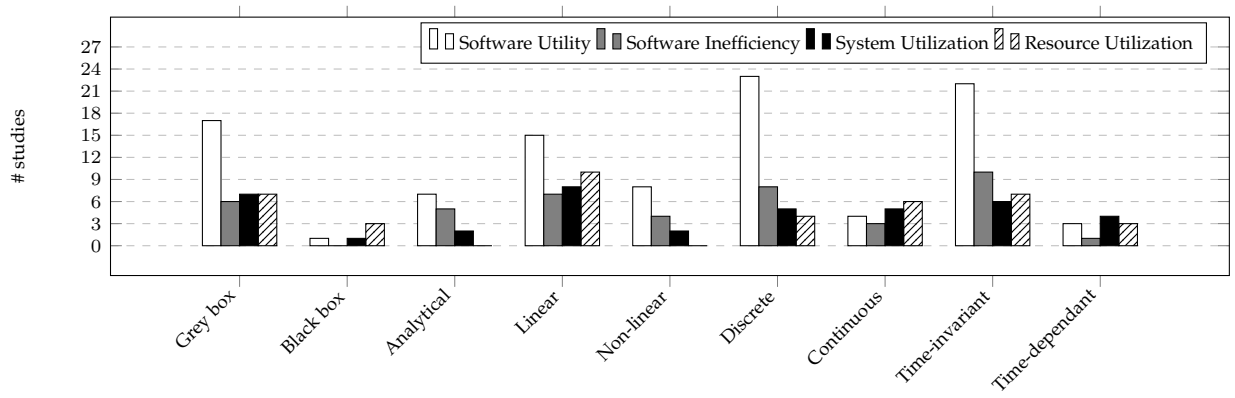


Fig. 12: Relation between F14.1 Sensors and F13 System model.

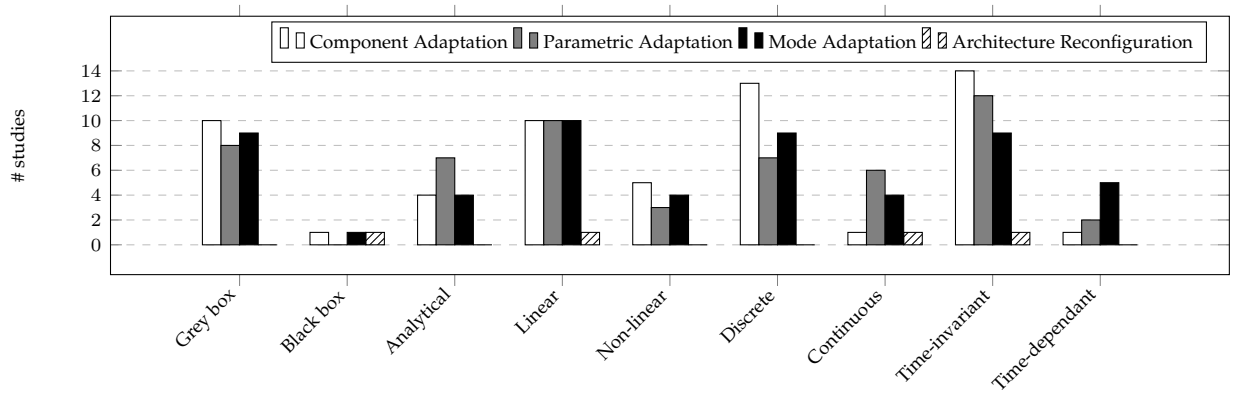


Fig. 13: Relation between F14.2 Actuators and F13 System model.

the task load and selecting modules for execution to deal with changing goals.

As the actuators directly effect the application software and/or middleware services, they are all specific to control-theoretical software adaptation. Similar to the implementation of sensors, actuators can be implemented by supporting functionality (framework API, component model, and similar) or through a dedicated software interface. A particular aspect of effecting mechanisms is ensuring locality and consistency of the software adaptation. This means adapting the system properly without stopping or disturbing the operation of the parts of the system unaffected by the adaptation, which is more challenging for coarse-grained types of adaptations, such as architecture reconfigurations. A typical approach to handle this is by adapting the system or parts of it in quiescent states [114]. We noticed that consistency of adaptation is to a large extent ignored in the primary studies of the survey. A related aspect of effecting mechanisms is that some adaptations may require more invasive changes, such as a partial or even complete reboot of the software system. Such kinds of adaptations are critical for controllers with a short adaptation period. A possible approach to address this aspect is suggested in one primary study that takes a software engineering perspective [90]. In this study, the authors encourage engineers to make sensors and actuators modifiable at runtime. Two other primary studies address this aspect by taking into account the controller overhead [89], [105]; a fourth study deals with it by minimizing the number of system reconfigurations [104].

We also checked whether there are any correlations between sensors/actuators and the system model (Figures 12 and 13). The analysis results give some indication that software utility sensors are the dominating type of sensors used, in particular for linear grey-box and time invariant models. Resource utilization is not used in analytical and non-linear models. Parametric adaptation and component adaptation are the dominating type of used actuators. Parametric adaptation is particularly preferred in analytical and continuous models; component adaption is the preferred actuator for discrete, time invariant models. However, as the figures show, the data for both sensors and actuators is scattered over different model elements, so it is difficult to derive clear conclusions.

Finally, we gathered data about the **triggers for adaptation** (item F15). In 36 out of 42 primary studies adaptation is triggered by changes in the environment. In 29 primary studies adaptation is also triggered by changes in requirements. Only 11 studies present experiments with changing requirement at runtime [21], [27], [29], [34], [78], [79], [81], [82], [93], [100], [104], and only a single study [27] supports removing or adding new requirements on the fly. Finally, in 7 studies, adaptation is triggered by changes in the software itself. These studies are mainly related to software development and testing, where software is often the only source that provides feedback.

TABLE 9: F16 Controller type.

Controller Category	Specific Controller	Primary Studies	Additional Information
PID	Proportional-integral	[21], [29], [70], [74], [75], [76], [77], [79], [82], [85], [88], [89], [90]	A classical controller that is easy to implement and tune. Consists of 3 components (P, I, D) responsible for different controller characteristics [53]. In 13 out of 18 studies, PID controllers are also adaptive as it helps to compensate for inaccuracy and errors in the system model [21], [70], [79]
	Proportional-integral-derivative	[93], [102]	
	Proportional	[80], [94].	
	Integral	[92], [107]	
MPC	Model predictive	[24], [83], [84], [95], [104], [105], [106]	Controller that uses a system model to predict its future behavior and selects adaptation actions that minimize the cost for achieving this behavior. MPC was used to optimally deal with multiple requirements in primary studies from different domains.
	Limited lookahead (LLC)	[28], [86], [87]	Controller that is conceptually similar to MPC: creates a set of future system states up to a certain horizon and selects a trajectory between these states such that its cost is minimal.
Feed-forward	Pure feedforward	[101]	Controller that computes adaptation actions based on the system model; the system output is not taken into account, i.e., there is no feedback.
	Feedback+ feedfoward	[74]	Applied in a single study, where feedforward and feedback controllers are paired and compared to other types of controllers.
Optimal	Custom optimal	[96], [97]	The goal of this controller is to minimize/maximize a cost function subject to certain constrains, e.g. maximize performance using a pool of limited resources. This controller was used in the context of software testing.
	Bang-bang	[98], [99]	This controller is also known as on-off controller because the control signal can take only two values, e.g., 0 or 1. This controller was used in the context of software development.
Deadbeat	Deadbeat	[34], [74], [81]	Controller created using a pole placement technique [15], [53]. Although being less robust to disturbances than PID, this controller has a very low settling time and limited overshooting.

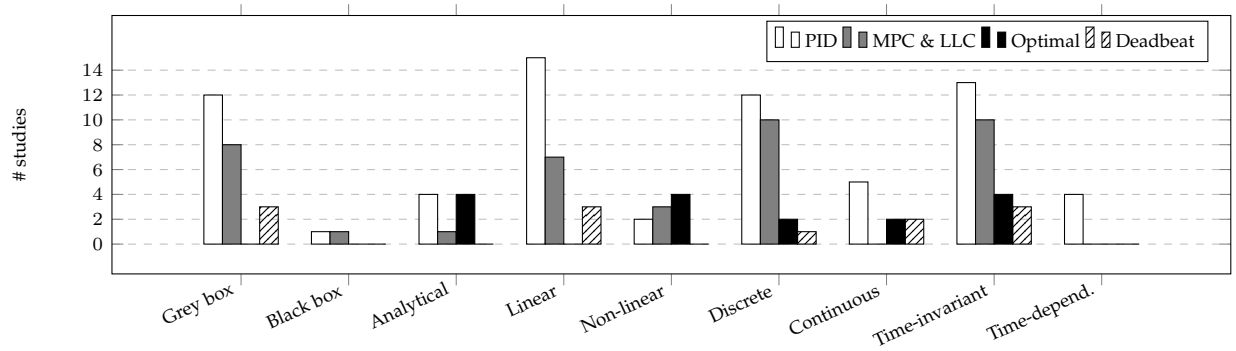


Fig. 14: Relation between F16 Controller type and F13 System model.

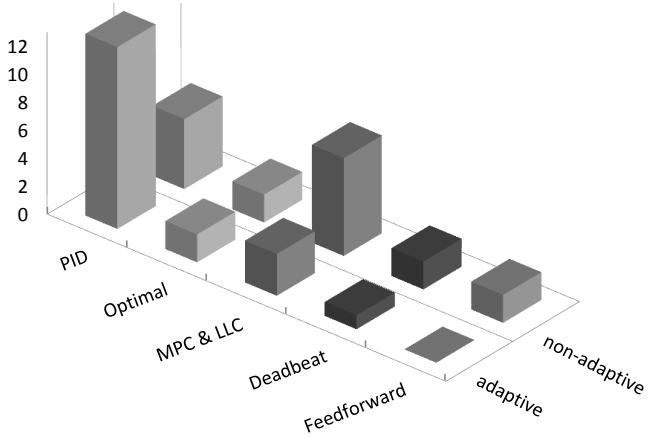


Fig. 15: F16: Controller type and F16.1: Adaptivity.

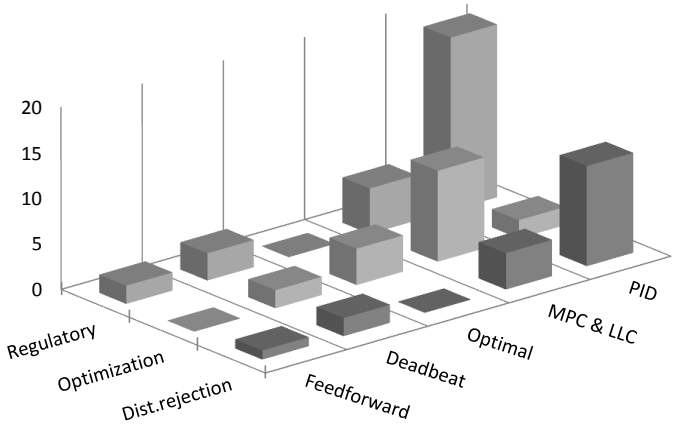


Fig. 16: F17: Controller purpose

Controllers: The results for the data extracted for controller types (item F16, Table 9) shows that 5 types of controllers have been used for control-theoretical adaptation in the primary studies. The dominant type of controller is the PID controller (50% of the primary studies). While being the most applied type of controller in the primary studies, it is less dominant as in industrial practice where PID controllers are used in around 90% of the control applications. MPC is used in 26% of the primary studies. MPC is the preferable choice for systems with multiple objectives. Other types of controllers used are Feedforward (5%), Optimal (10%) and Deadbeat (8%). Table 9 gives additional information about the controller types with examples how they are used in primary studies. It is notable that 4 primary studies do not specify a concrete controller, but instead refer to "any kind of feedback mechanism that makes a software fulfill its requirements" [78], [91], [100], [103]. We also underline a specific case of adaptation, where multiple feedback loops at different levels of computing systems interact with each other to address the adaptation goals. This case mostly occurred in primary studies that apply hierarchical control (e.g., [75], [76], [86], [87]), where a higher level controller of a software application provided goals for lower level controllers that manage resources such as CPU and memory.

Regarding the adaptivity of controllers (Figure 15), in 13 out of 18 studies, PID controllers are also adaptive as it helps to compensate for errors that may result from the linearizion in the modeling phase [21], [70], [79]. The other types of controllers used in the primary studies are mostly non-adaptive.

The data extracted for Controller purpose (item F17, Figure 16) shows that PID is the preferred solution for regulatory control (setpoint tracking) and disturbance rejection. However, PID controllers do not scale easily, so their use is typically limited to single-input, single-output systems. The need to support adaptation for multiple objectives (for example: performance and failure rate), while functioning under constraints (like resource limitations) or requiring the system to optimize for some parameter (like minimizing the operational cost), led to the use of model predictive and optimal control [83], [105]. A well-known drawback of optimal controllers is that they are sensitive to modeling errors and runtime disturbances. This can be also observed in Figure 16 where optimal controllers are used solely for optimization purpose.

An interesting topic for analysis are possible *correlations between controller types and system models*. We observed the following tendencies (Figure 14):

• 15 out of 17 primary studies use PID controllers with linear models. In addition to the complexity of building or identifying non-linear models, PID is not very effective in controlling processes that are non-linear and time-invariant [115, p.52]. A common practice from industrial control is combining a complex controller with a simple linear time-invariant model, and this approach seems to be adopted for software adaptation as well.

• All 10 studies with MPC controllers use discrete time-invariant models, 8 of which are grey-box models. The motivation for using discrete time-invariant models is similar to the use of linear models combined with PID control: it is a simple model to work with. Hence, it is preferred over complex non-linear or adaptive models. The motivation to combine grey-box models with MPC is based on the adaptation requirements of the software systems under study, which often have multiple inputs and outputs. As it is challenging to build a model of such systems without identification, a grey-box model is a preferred choice.

• The 4 primary studies focusing on optimal control used non-linear time-invariant analytical models. The motivations for using optimal control are similar to MPC, so it is not surprising to see a preference for time-invariant models. Nevertheless, the fact that all 4 studies use non-linear analytical models is surprising, and it worthwhile to see whether future studies will confirm this trend.

Finally, we looked at the *composition of multiple controllers* into a single feedback mechanism. The extracted data yields the following insights:

• Six of the primary studies apply hierarchical control. In 4 of these studies [75], [76], [86], [87], a high-level controller solves global software adaptation tasks and provides input for controllers at a second level that solve intermediate tasks and provide input for controllers of lower level that solve local adaptation tasks. A reversed two-level hierarchical control approach is studied in [29], [82], where multiple controllers at the top level provide inputs to a single controller at the bottom level.

• Two studies apply switching control [27], [85], where different control laws interchange with one another, depending on the actual software adaptation tasks.

• Two studies apply cascaded control [34], [81], where the output signal of a high level controller becomes an input for a lower level controller.

• Finally, one study applies cooperative control [104], where multiple controllers work in parallel, contributing to achieve a global software adaptation task.

RQ3: Control Strategies

- Software adaptation requires specific sensors for measuring software utility and software inefficiency (along with conventional sensors to measure elements at lower levels of the technology stack and environment).
- The actuators directly effect the application software and/or middleware services, hence they that are all software-adaptation specific. Consistency of adaptation is largely ignored in the primary studies.
- PID and MPC are the dominating types of controllers used in software adaptation. The use of PID (50% of studies) is not as dominant as in current industrial practice.
- Studies using PID control, prefer to combine this with linear models, while studies that use MPC control prefer discrete time-invariant grey-box models.
- PID control is mostly used for regulatory functions and disturbance rejection in single-input, single-output systems. MPC and optimal control is mostly used to achieve optimality in systems with multiple goals.

TABLE 10: F18 Formal guarantees and F18.1 Experimentally verified guarantees

Guarantee	Formally Analyzed	Achieved by	Verified Experiment.	Measured by
Stability	[70], [78], [80] [21], [29], [34], [82]	Keeping the pole of the controller in a certain interval. A notable exception is [80] that analyzes routing probabilities for network nodes.	[83], [84] [85], [86] [97], [102]	Ability of the system to achieve its goals. [102] measures stability as the number of system reconfigurations that occur during adaptation.
Settling time	[21], [29], [34], [70], [78], [80], [82]	Analyzing the pole of the controller.	[77], [102] [83], [84] [85], [108]	The time required to reach the setpoint after a goal change.
Overshoot	[29], [34], [80], [82]	Keeping the pole of the controller in a certain interval.	[83], [84], [85], [108]	Spikes in the system output for different adaptation options.
Steady-state error	[29], [82]	Analysing the output equation of the system.	[77], [83], [84], [85]	Oscillations in the response time of the software for different adaptation options.
Robustness	[21], [29], [70], [78], [82]	Analyzing the feedback loop transfer function.	[80], [108]	Deviations in the system output under disturbances.
Optimality	[34], [95], [96], [98], [99]	Mathematically solving an optimization problem.	[104]	The tasks completed and the resources used by the software for different adaptation options.
Cost of control	[28], [104]	Analyzing a separate cost function.	[81], [89], [90], [105]	The the amount of resources consumed by the adaptation mechanism to achieve the goals.

7.6 RQ4: Goals and Guarantees

To answer research question four (what type of goals are achieved with control-theoretical adaptation of software and what kind of guarantees are provided?) we used data items F18-F20. The data extracted for software qualities (item F19, Figure 17) shows that the primary focus is on performance, efficiency, reliability, and business value[9] of the application.

The data extracted for guarantees (item F18) shows that 13 studies provide formal guarantees for required properties (item F18), while 13 primary studies provide empirical evidence for guarantees of required properties (item F18.1). Table 10 provides an overview of the different types of guarantees. Each type is illustrated with examples from studies that provide formal guarantees and studies that provide empirical evidence for guarantees.

The extracted data for quality tradeoffs (item F20) shows that most of the primary studies do not mention any tradeoffs. Only 3 primary studies consider tradeoffs between software qualities, namely, performance versus accuracy or reliability [86], [98], [101]. Seven studies discuss the tuning of a controller to trade different guarantees, typically robustness for settling time [21], [29], [70], [77], [78], [80], [102].

9. Business value refers to the profit earned with the application.

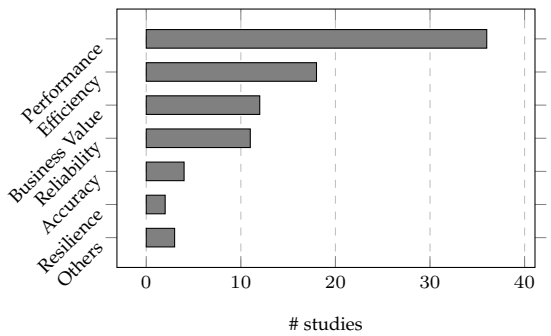


Fig. 17: F19 Software qualities.

An interesting topic of analysis is the *correlation between software qualities and achieved guarantees*. Unfortunately, most studies do not provide a clear description of how the software qualities (adaptation goals) relate to the analyzed guarantees. Hence, we had to infer this information:

- Stability indirectly relates to all software qualities that are subject of adaptation and shows the ability of an adaptation mechanism to converge to the goals. However, guarantees for stability are different for different qualities; e.g. lack of stability for a performance goal may imply fluctuations in the throughput of the software application, while lack of stability for a security goal may imply periods with higher vulnerability of the system.
- Settling time is also related to all qualities to be satisfied by the adaptation and shows the time it takes for an adaptation mechanism to bring measured quality properties close to their goals. It is generally acknowledged that the settling time should not be too small as this would compromise stability/robustness, but not too big as this decreases the quality being satisfied [70], [102]. Notably, 7 out of 11 primary studies discussing settling time guarantees are concerned with performance, in particular response time.
- Similarly, overshooting relates to all software qualities that are subject of adaptation and shows how the measured output exceeds the goal during the transient phase. Guarantees for overshoot have a different interpretation for different qualities, e.g., having overshoots on the system response time leads to violation of performance quality. Avoiding overshooting avoids penalties on the respective software qualities [34].
- Steady-state error relates to all software qualities that are subject of adaptation as well. It shows how big is the amplitude of oscillations of measured output around the setpoint during steady state. For example, in [77] the authors calculate the steady-state error as the mean of the absolute error on a response time requirement. The authors conclude that a higher steady-state error decreases performance.
- Robustness relates to reliability in all primary studies that analyze this property. Indeed, the amount of distur-

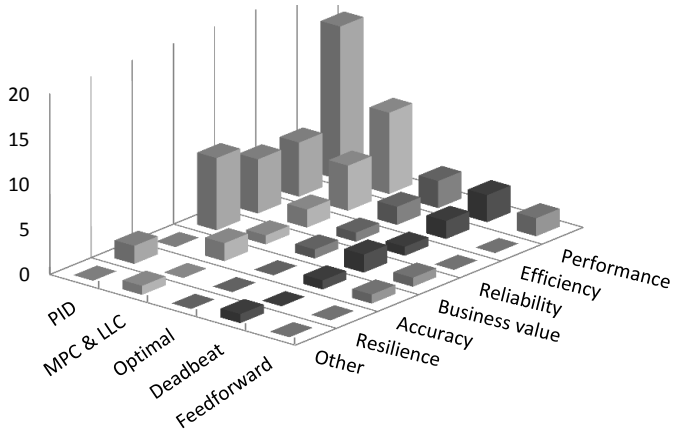


Fig. 18: F16: Controller type vs F19: Software qualities.

bance the system can withstand directly influences its reliability. One approach to analyze this relation is by adding white noise to the system inputs [21]. Having a more robust software can enhance performance by maintaining low latencies, or increase business value by serving more advertisements on web sites [80].

• Optimality is another control property that relates to any type of software quality. Examples in the primary studies are performance [96], security [34], reliability [98], and business value [99]. Lack of optimality implies that there are no guarantees that the adaptation mechanism achieves the most favorable output for the software quality under consideration.

• Control cost and overhead relates to efficiency in 6 primary studies that analyze these properties. In these studies the authors look at resources that are spent on satisfying the adaptation goals and on performing adaptation actions. In two cases, controller cost affects system performance as well [81], [104].

To conclude, we look at a number of additional correlations between guarantees and other data items. Correlating the main motivations for control-theoretical software adaptation (item F8) with guarantees shows that 7 of 10 primary studies that stated "formal guarantees" as a main motivation also provide formal guarantees. When correlating software qualities to sensors (item F14, see also Figure 11a), we obtained the following results: the most frequently used sensors for measuring performance – the primary software quality that is subject of adaptation – are of the software utility and system utilization type. Reliability on the other hand is measured by sensors of the software inefficiency type, efficiency is measured by either resource or system utilization, while business value correlates to software utility and resource utilization. Comparing software qualities with controller types (Figure 18), we observe that PID is the dominating type of controller used for all qualities considered in software adaptation, except for accuracy, for which MPC controllers are mostly used. On the other hand, performance is handled by all types of controllers that are applied in software adaptation.

RQ4: Control Guarantees

- Research of software adaptation is primarily focused on software qualities and does not exploit the full potential of control theoretical guarantees.
- Software performance, efficiency, and reliability are the most frequently applied adaptation goals. Business value is an emerging quality goal for software adaptation.
- Most of the primary studies do not provide a tradeoff analysis of system qualities or guaranties.
- Robustness, optimality, and cost are commonly analyzed properties, together with classical control-theoretical guarantees like stability and settling time.
- Guarantees for required properties are provided either by means of formal analysis or by collecting empirical evidence.
- The relation between software qualities and control theoretic guarantees remains largely implicit. We inferred that stability, settling time, overshooting, steady-state error and optimality relate to all quality properties, while robustness relates to reliability, control cost and overhead relate to efficiency.

8 Discussion

In this Section, we reflect on the results of the survey focusing on two topics: comparison with the results of the surveys of Patikirikorala et al. [32] and Brun et al. [14], and open challenges for future research in control-theoretical software adaptation.

8.1 Comparison with Patikirikorala et al. [32].

Before we compare the results of our survey with results reported in [32], it is important to emphasis that the scope of the survey of Patikirikorala et al. [32] is different from our survey: while we concentrate on the adaptation of software, in particular application software and middleware services, [32] does not distinguish between control-based adaptation at different layers of computing systems. Furthermore, a large part of the results of our survey cannot be compared since [32] does not consider important aspects of control-based adaptation, including model properties such as model linearity, time framework, model time dependencies, actuators, controller purpose, guarantees, among others items that we collected and analyzed.

Nevertheless, we can compare the following:

1) *Model type*. The ratio between black-box plus grey-box models and analytical models in our survey is similar with the results of [32] (about 65/35). However, a notable distinction concerning model type is that [32] does not distinguish between black-box and grey-box models. As shown in our review, the difference is very relevant. In our survey, black-box models are used rarely (3 studies compared to 23 studies that use grey-box models) and, in most cases, black-box models are used as a part of generic frameworks. As for types of analytical models, [32] reports that almost

half of the analytical models are queuing network models. Our survey, on the other hand, found that different types of analytical models are used, with only 2 of 11 studies using queuing networks.

2) *Sensors* (referred as "performance variables" in [32]). The most frequently used types of sensors reported in [32] are response time, resource utilization and system utilization, and "hit or miss ratio." Resource utilization and system utilization directly map to the same sensor types in our survey. Response time and "hit or miss ratio" fit under sensor types software utility/inefficiency in our survey. However, other sensor variables specific to software adaptation, as listed in Table 8a are not reported in [32].

3) *Controller type*. As [32] classified controllers together with composition schemes, the reported results are hard to compare with the results of our literature review. However, we can still see that PID controllers are the dominant type of controllers that emerged in both surveys. On the other hand, MPC was much more used in primary studies of our survey compared to [32]. Optimal control (see LQR in [32]) and feedforward control were used in a small number of analyzed studies in both surveys. As for the controller composition schemes, both our review and [32] found studies that use hierarchical, cascaded, and switching control. However, the number of such studies was relatively low in both surveys.

4) *Controller adaptivity and composition scheme*. In our survey, adaptive controllers were used in almost 50% of the primary studies, while [32] reported only 15% for this data item. Explaining such a mismatch is not difficult because [32] classified adaptive controllers in a separate group, without identifying which types of controllers (PID/MPC/etc.) were adaptive.

5) *Assessment Approach*. The ratio of studies that used example applications and simulation as assessment approach compared to other assessment approaches is approximately equal in both our survey and [32]. As a side note, [32] refers to example application as "case study with a test bed." According to our observations, almost none of the primary studies applies a scientifically valid case study approach, but rather provide results of one or two adaptation scenarios. Moreover, some of the studies justify their approach only with discussion.

6) *Application Domain*. Although [32] does not specify the precise application domains (e.g., middleware, data storage, and virtual machine are technologies rather than application domains), the authors noted that many analyzed approaches deal with managing web/application servers. In our review we observed a similar trend with studies from the e-commerce domain, where content was optimized on the server side of the application. It is also notable that RuBIS was one of the most used benchmark in both surveys.

8.2 Comparison with Brun et al. [14].

Although the article by Brun et al. [14] is not based on a systematic analysis of the state of the art and has a broader focus as this systematic literature review, we can find a number of commonalities and differences compared to the results of our review.

[14] discusses the role of feedback loops in self-adaptive systems in general and from a control engineering perspective in particular. The authors state that a key reason for using feedback control is to reduce the effects of uncertainty which appear in different forms as disturbances or noise in variables or imperfections in the models of the environment used to design the controller. The main motivations for applying control theory to software adaption derived from the primary studies of our review are formal guarantees, the maturity of the field of control theory, and the effectiveness of control theory. Inline with [14], uncertainty is a basic underlying reason for applying self-adaptation, however, our survey provides concrete arguments why authors have applied control theory to realise adaptation.

The part of [14] that focusses on control theory in particular is on adaptive control. The authors discuss Model Identification Adaptive Control (MIAC) and Model Reference Adaptive Control (MRAC) that can be considered as two reference models of how adaptive control can be realised. As explained above, the results of our review show that roughly half of the primary studies apply adaptive control. Rather than providing information about what kind of reference model has been used to realise adaptive control, the review results pinpoint: (i) which types of controllers are used in adaptive control, with PID being the dominant type; and (ii) which adaptation techniques are used, which include updating model parameters or switching the model, updating parameters of the control law or changing the law, and involving human operators to make a decision. Some of these approaches realise structural changes that go beyond adaptive control as in MIAC and MRAC.

[14] does not consider many aspects that we studied in our systematic literature review (which was not the particular aim of [14]). These aspects including the formal guarantees that can be provided by applying control theory to software adaptation, system models and their properties, the types of sensors and actuators used, concrete controller types and purposes, and the link between controller properties and software qualities.

8.3 Challenges for Future Research

To conclude, we outline a number of challenges that we identified during data analysis and answering the research questions. We clarified particular challenges for software engineers, for control engineers, and for both.

System models. The review results show that researchers prefer to work with simple linear time-invariant discrete models. This contrasts with the inherent complexity and non-linear nature of software stated in most of the primary studies. One challenging aspect of linear time-invariant discrete models is their ineffectiveness when the software application is subject to drastic disturbances (for example a sudden change in available resources, or software components that fail). The common solution to handle such situations as used in the primary studies is changing model and/or the controller parameters online. While this solution has shown great potential in traditional control applications, there is a need for substantial evidence to demonstrate its usefulness for handling adaptation of software applications, which is a particular challenge for software engineers.

TABLE 11: Software qualities versus control theoretic guarantees

Control Guarantee	Quality Properties	Note
Stability	All, indirectly	Guarantees on the ability of the system to converge to the goals. This connection is one-directional, i.e., a system can be stable without goals, but a goal cannot be achieved in an unstable system. Different interpretation for different qualities.
Settling time	All	Guarantees on time it takes to bring measured quality property close to its goal. Settling time should not be too small (for stability/robustness) but also not be too high (decrease of quality).
Overshoot	All	Guarantees on the degree the measured output exceeds the goal in transient phase. Different interpretation for different qualities.
Steady-state error	All	Guarantees on the amplitude of oscillations of measured output around the setpoint during steady state. Different interpretation for different qualities.
Robustness	Reliability	Guarantees on the amount of disturbance a system can withstand; relates directly to reliability of the system.
Optimality	All	Guarantees that the system reaches the most favourable output for the given quality.
Control cost and overhead	Efficiency	Guarantees on the resources used for satisfying goals and performing adaptation actions.

Complementary to that, an important challenge for software engineers to apply control-theoretical adaptation is to create a mathematical model of the software. [78] suggests exploring known analytical models used in control theory (such as Markov models and queuing networks) to fill the semantic gap between architecture-based and control-based adaptation of software. Along this line, [80] outlines a general control design methodology for queuing networks. Currently there is little research on using non-linear or continuous models to deal with adaptation of software. It would be interesting to investigate whether such models would work better, however, they are complex to build and require sufficient background in control theory. Consequently, software engineers may involve control engineers when tackling this challenge.

As for the model type, grey-box models were applied in almost 60% of the software systems of the primary studies. As these models reflect only particular parameters of the system, an open question is: how to choose the system parameters to be modeled and what techniques to use in order to identify those parameters? One generally applicable grey-box model was found during this review, see the LLR model in Table 7. However, most of the grey/black-box models used in the primary studies were developed to handle a specific case. Both software and control engineers should devote more efforts on identifying generic grey/black-box models for different types of software systems.

Sensors and actuators. The review results show that software adaptation requires new types of software sensors, as well as actuators that have a direct effect on the application software. We observed that in systems where actuation time is critical, the authors suggest taking the cost for adaptation into account when designing the adaptation mechanism. While we were able to provide a broad classification of the types of sensors and actuators used in software adaptation, there is currently no clear view on how sensing and actuating of software for control-theoretical adaptation can be supported in a systematic way, both from an architecture and implementation point of view. Hence, challenging questions both for software and control engineers are: (a) how to translate software qualities, such as security and resilience, to setpoints? (b) what sensors could be used to measure particular software qualities? (c) how to translate controller outputs to actuators that effect the software? (d) how to ensure locality and consistency of adaptation, how to support quiescence for control-theoretical adaptation of software?

Controllers. We observed that the choice of particular controllers depends on the problem at hand. For software with a single adaptation goal, adaptive PI controller is the preferred choice in the primary studies. For software with multiple adaptation goals preference is given to MPC and optimal controllers. However, several aspects regarding the choice of controllers remain open for further research. Open questions both for software and control engineers include: Are the current solutions scalable to real-world systems? Or even stronger: for what types of real software systems are controllers applicable? What are appropriate controllers to deal with priorities and tradeoffs among quality goals in software adaptation? What controllers are suitable for handling uncertainties in software systems that can only be resolved at runtime? Can we utilize the reusability and portability techniques from software engineering to design reusable controllers?

Guarantees for adaptation goals. Our review shows that control-theoretical software adaptation is concerned with addressing typical software goals, in particular performance, efficiency, and reliability. As modern software systems often need to be designed with partial knowledge, providing guarantees is essential. However, we observe that formal analysis of guarantees is poorly exploited in most of the primary studies. One challenging aspect of software adaptation that we tried to address in this literature review is connecting software qualities to control theoretical guarantees. Table 11 summarizes the results. As most authors do not provide an explicit connection between control theoretic guarantees and quality properties, it would be interesting to further investigate this connection with future primary studies. Such study would benefit from joint efforts of software and control engineers. An open challenge that comes from the implicit connection between software qualities to control theoretical properties is to select the proper control techniques in order to satisfy the quality properties specified by the stakeholders.

9 Threats to validity

To increase the quality and soundness of the review results, we followed a systematic approach. However, we point to possible threats to validity.

Internal validity: the extent to which a causal conclusion based on a study is warranted. The topic of this literature review lays at the intersection of two very different disciplines: control theory and software engineering. The disciplines have a different culture and use different vocabulary. Even the term "adaptive" has a different meaning in these two communities (see clarification in the introduction of the paper). To address this threat, the research team involved in this survey was balanced with an equal number of researchers from both disciplines. The researchers had comparable experience and worked closely together during all phases of the review process. In addition, our particular focus was on software adaptation that uses classical or advanced control techniques. Deciding whether a study should be included or not, was not always straightforward, in particular regarding the adaption of software at application and middleware level, and inclusion of some areas of control theory, such as discrete event control. To mitigate this threat, the decision on study inclusion was always based on agreement between at least two researcher that independently checked the papers. In case of disagreement, a third researcher was consulted and after discussion, a decision was made in consensus.

External validity: the extent to which the findings can be generalized to all control-theoretical software adaptation research. We acknowledge that limiting the automatic search to selected venues and applying an automatic search strategy using a selection of search engines, we may have missed some primary studies. To preempt this threat, we took several measures. First, during the selection of the venues we followed a thorough process in which the review team worked closely together and consulted with experts of the two disciplines to crosscheck and identify missed target venues. In this process, we followed an inclusive policy, without compromising on the expected quality of primary studies. Second, we started the search process with pilot searches to define and tune the search string, cross-checked the data using both general-purpose and scientific search engines, actively involved expertise of colleagues in the selection process when needed. Thirdly, we performed snowballing to find potentially missed material.

Construct validity: the extent to which we obtained the right measure and whether we defined the right scope in relation to what is considered research on software adaptation. The definition of control-theoretical software adaptation we used in this survey (see Section 2) may be biased and the list of extracted data items (Section 6.6) may be incomplete. Regarding the scope on software (application software and middleware services), we relied on well-established insights from the field of software engineering. Regarding the scope of adaptation mechanisms, we acknowledge that there is not a general consensus on what is considered control-based adaptation. Our choice to limit the scope to classic and advanced control theory is motivated by the very different nature of realising adaptation with other related paradigms. To address this threat, we consulted with researchers from both software engineering and control theory domains, as well as utilized experience of related surveys, such as [32]. Finally, there may be threat regarding the quality of reporting of studies that may have affected both the selection of papers and the extraction of data. To anticipate this threat, we extracted data about the quality of reporting. We found out that many primary studies reported only results from successful experiments and did not acknowledge threats to validity. Hence, our review may not show particular limitations of control-theoretical software adaptation. But in general, the reporting quality of the primary studies was good, which provides a basis to make conclusions about the validity of extracted data.

Reliability: extent to which we can ensure that our results are the same if our study would be conducted again. The researchers involved in this survey may have been biased when collecting and analyzing data of studies. To address this threat, the team defined a detailed protocol [116] for the survey that provides an explanation of the survey goals, the data items that are collected, the analysis performed, and the techniques applied to classify results. In particular, data extraction and analysis was done by two researchers in parallel and further discussed in case of differences in opinions to increase confidence. Nevertheless, the background and experience of the researchers may have created some bias, and introduced some level of subjectivity in some cases. This threat is also related to conclusion validity, which is concerned with the ability to replicate the same findings.

10 Conclusion

In this paper, we reported the results of a systematic literature review that aimed to shed light on the use of control theory as a paradigm for designing adaptive software. The study results show that control-theoretical software adaptation research is still in a preliminary stage. The number of studies is still low, but we observe a rapid growing interest in the field over the last years. We also found a number of studies where control theory was applied to the software artifacts in the development life cycle, which indicates about the research interest in a broader use of control theory for self-adaptation.

Despite software is usually considered highly non-linear, the majority of the studies use simple linear models. Most of the studies evaluated their work with simple applications or simulations. This raises questions about how well the current approaches, in particular with simple linear models, will scale to real-world applications, or whether other approaches need to be explored. To achieve the quality goals of software applications, these goals have to be translated into control goals (setpoints). Furthermore, to adapt the software and measure the effects of the controller actions, the software applications need to be instrumented with sensors and actuators. There is currently no clear view on how this translation can be done in a systematic manner and how sensors and actuators for control-theoretical software adaptation can be realized in an effective way. Finally, the key driver to explore control-theoretical software adaptation reported in the studies is the formal underpinning of control theory as a basis to provide guarantees for adaptation goals.

This survey shows that classic controller guarantees are poorly exploited when engineering control-based solutions. Explicitly linking control theoretic guarantees to software qualities is a challenging topic for future research.

To conclude, we would like to emphasize that research on control-theoretical software adaptation is situated at the crossing of two disciplines: software engineering and control theory. Traditionally, these disciplines operate in different worlds, but progress in these fields requires that both disciplines take an open position to one another. Without the joint effort of researchers from both disciplines this survey would not have been possible. We hope that the outcome of this joint effort may be a stimulus for new research in this exciting area.

REFERENCES

[1] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Runtime software adaptation: Framework, approaches, and styles," in *Companion of the 30th International Conference on Software Engineering*, ser. ICSE Companion '08. New York, NY, USA: ACM, 2008, pp. 899–910. [Online]. Available: http://doi.acm.org/10.1145/1370175.1370181

[2] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[3] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann, "POET: a portable approach to minimizing energy under soft real-time constraints," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium, Seattle, WA, USA, April 13-16, 2015*, 2015, pp. 75–86. [Online]. Available: http://dx.doi.org/10.1109/RTAS.2015.7108419

[4] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "(Requirement) evolution requirements for adaptive systems," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 155–164. [Online]. Available: http://dl.acm.org/citation.cfm?id=2666795.2666820

[5] B. H. Cheng, R. de Lemos, and et al., "Software engineering for self-adaptive systems: A research roadmap," B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1–26. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_1

[6] R. de Lemos et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science, R. de Lemos, H. Giese, H. Muller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 1–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35813-5_1

[7] P. Oreizy, N. Medvidovic, and R. Taylor, "Architecture-based runtime software evolution," in *20th International Conference on Software Engineering*, 1998.

[8] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, 2004.

[9] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *Future of Software Engineering*, 2007.

[10] D. Weyns, S. Malek, and J. Andersson, "FORMS: Unifying Reference Model for Formal Specification of Distributed Self-adaptive Systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 1, pp. 8:1–8:61, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2168260.2168268

[11] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, Sep. 2012. [Online]. Available: http://doi.acm.org/10.1145/2330667.2330686

[12] D. Weyns, N. Bencomo, R. Calinescu, J. Cámara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli, "Perpetual Assurances in Self-Adaptive Systems," *Software Engineering for Self-Adaptive Systems: Assurances (Dagstuhl Seminar 13511)*, 2014. [Online]. Available: http://homepage.lnu.se/staff/daweaa/papers/2015Dagstuhl.pdf

[13] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1516533.1516538

[14] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Engineering Self-Adaptive Systems Through Feedback Loops, pp. 48–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_3

[15] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[16] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, V. Papadopoulos, Alessandro, S. Ray, M. Sharifloo, Amir, S. Shevtsov, M. Ujma, and T. Vogel, "Software Engineering Meets Control Theory," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Firenze, Italy, May 2015. [Online]. Available: https://hal.inria.fr/hal-01119461

[17] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What does control theory bring to systems research?" *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 62–69, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1496909.1496922

[18] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *Control Systems, IEEE*, vol. 23, no. 3, pp. 74–90, June 2003.

[19] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, 2002, pp. 219–234.

[20] X. Zhu, Z. Wang, and S. Singhal, "Utility-driven workload management using nested control design," in *American Control Conference, 2006*, June 2006.

[21] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 299–310.

[22] J. Andersson, R. de Lemos, S. Malek, and D. Weyns, "Modeling dimensions of self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems*, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 27–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02161-9_2

[23] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit, "A design space for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 33–50. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35813-5_2

[24] K. Angelopoulos, A. V. Papadopoulos, and J. Mylopoulos, "Adaptive predictive control for software systems," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, ser. CTSE 2015. ACM, 2015, pp. 17–21.

[25] K.-Y. Cai, J. Cangussu, R. A. DeCarlo, and A. Mathur, "An overview of software cybernetics," in *Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on*, Sept 2003, pp. 77–86.

[26] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 79–88. [Online]. Available: http://doi.acm.org/10.1145/1809049.1809065

[27] Y. A. Eracar and M. M. Kokar, "An architecture for software that adapts to changes in requirements," *Journal of Systems and Software*, vol. 50, pp. 200–0, 1998.

[28] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, and S. Abdelwahed, "Enabling self-managing applications using model-based online control strategies," in *Proceedings of the 2006*

IEEE International Conference on Autonomic Computing, ser. ICAC '06, 2006, pp. 15–24.

[29] S. Shevtsov and D. Weyns, "Keep it SIMPLEX: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 2016 11th Joint Meeting on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016.

[30] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.

[31] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, 2011.

[32] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, June 2012, pp. 33–42.

[33] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289–302, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/12.990127

[34] A. Filieri, H. Hoffmann, and M. Maggio, "Automated multi-objective control for self-adaptive software design," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786833

[35] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva, "Comparison of decision-making strategies for self-optimization in autonomic computing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 4, pp. 36:1–36:32, Dec. 2012. [Online]. Available: http://doi.acm.org/10.1145/2382570.2382572

[36] P. Jamshidi, A. Ahmad, and C. Pahl, "Autonomic resource provisioning for cloud-based software," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 95–104.

[37] P. Jamshidi, A. M. Sharifloo, C. Pahl, A. Metzger, and G. Estrada, "Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution," *CoRR*, vol. abs/1507.00567, 2015.

[38] Y. Wang, H. K. Cho, H. Liao, A. Nazeem, T. P. Kelly, S. Lafortune, S. Mahlke, and S. A. Reveliotis, "Supervisory control of software execution for failure avoidance: Experience from the gadara project," *IFAC Proceedings Volumes*, vol. 43, no. 12, pp. 259 – 266, 2010, 10th IFAC Workshop on Discrete Event Systems. [Online]. Available: //www.sciencedirect.com/science/article/pii/S1474667015324666

[39] A. Girault and E. Rutten, "Automating the addition of fault tolerance with discrete controller synthesis," *Form. Methods Syst. Des.*, vol. 35, no. 2, pp. 190–225, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1007/s10703-009-0084-y

[40] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel, "Assured and correct dynamic update of controllers," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16. New York, NY, USA: ACM, 2016, pp. 96–107. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897056

[41] S. Geetha, V. Ramalakshmi, S. Bhuvaneeswari, and B. RameshKumar, "Evaluation of the performance analysis in fuzzy queueing theory," in *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, Jan 2016, pp. 1–5.

[42] M. Maggio, E. Bini, G. Chasparis, and K.-E. Årzén, "A game-theoretic resource manager for rt applications," in *25th Euromicro Conference on Real-Time Systems*, July 2013, pp. 57–66.

[43] D. Weyns, "Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges," in *Handbook of Software Engineering*, K. Kyo Chul Kang and S. Cha, Eds. Springer, 2017.

[44] B. Cheng, R. de Lemos, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems I*. Lecture Notes in Computer Science vol. 5225, Springer Verlag, 2009.

[45] R. de Lemos, H. Giese, H. Mller, and M. Shaw, Eds., *Software Engineering for Self-Adaptive Systems II*. Lecture Notes in Computer Science vol. 7475, Springer Verlag, 2013.

[46] N. Bencomo, R. France, B. Cheng, and U. Amann, Eds., *Models at Runtime*. Lecture Notes in Computer Science vol. 8378, Springer Verlag, 2014.

[47] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, Dec. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186778.1186782

[48] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 7:1–7:28, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1380584.1380585

[49] D. Weyns, M. U. Iftikhar, and J. Söderlund, "Do external feedback loops improve the design of self-adaptive systems? a controlled experiment," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 3–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487336.2487341

[50] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, ser. RE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 125–134. [Online]. Available: http://dx.doi.org/10.1109/RE.2010.25

[51] M. U. Iftikhar and D. Weyns, "ActivFORMS: Active formal models for self-adaptation," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 125–134. [Online]. Available: http://doi.acm.org/10.1145/2593929.2593944

[52] W. Levine, *The Control Systems Handbook, Second Edition: Control System Advanced Methods, Second Edition*, ser. Electrical Engineering Handbook. Taylor and Francis, 2010.

[53] K. J. Åstrőm and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.

[54] M. Krstic, P. V. Kokotovic, and I. Kanellakopoulos, *Nonlinear and Adaptive Control Design*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1995.

[55] K. Ogata, *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[56] L. Ljung, *System Identification: Theory for the User*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[57] B. C. Kuo and F. Golnaraghi, *Automatic Control Systems*, 8th ed. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[58] B. Wittenmark, K. Åström, and K.-E. Årzén, "Computer control: An overview," Tech. Rep., 2002.

[59] K. Åström and T. Hägglund, *Advanced PID control*. Research Triangle Park, NC 27709: ISA-The Instrumentation, Systems, and Automation Society, 2006.

[60] M. Morari and E. Zafiriou, *Robust Process Control*. Prentice Hall, Englewood Cliffs, 1989. [Online]. Available: http://www.google.se/books?id=HEcbgfyZEFoC

[61] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[62] S. Skogestad and I. Postlethwaite, *Multivariable feedback control: analysis and design*. Wiley New York, 2007, vol. 2. [Online]. Available: http://www.nt.ntnu.no/users/skoge/book/

[63] J. Maciejowski, *Predictive control: with constraints*. Pearson education, 2002.

[64] E. Camacho and C. Alba, *Model Predictive Control*, ser. Advanced Textbooks in Control and Signal Processing. Springer London, 2013.

[65] D. Weyns and T. Ahmad, "Claims and evidence for architecture-based self-adaptation: A systematic literature review," in *Proceedings of the 7th European Conference on Software Architecture*, ser. ECSA'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 249–265. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39031-9_22

[66] P. C. Diniz and M. C. Rinard, "Dynamic feedback: An effective technique for adaptive computing," in *Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation*, ser. PLDI '97. New York, NY, USA: ACM, 1997, pp. 71–84.

[67] J. Guitart, J. Torres, and E. Ayguad, "A survey on performance management for internet applications," *Concurrency*

and Computation: Practice and Experience, vol. 22, no. 1, pp. 68–106, 2010. [Online]. Available: http://dx.doi.org/10.1002/cpe.1470

[68] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 80–89. [Online]. Available: http://doi.acm.org/10.1145/1988008.1988020

[69] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, and E. Chang, "Cloud service selection: State-of-the-art and future research directions," *Journal of Network and Computer Applications*, vol. 45, pp. 134 – 150, 2014.

[70] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 700–711.

[71] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*. Wiley, 1994.

[72] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou, "Variability in software systems: A systematic literature review," *Software Engineering, IEEE Transactions on*, vol. 40, no. 3, pp. 282–306, March 2014.

[73] H. Zhang and M. Ali Babar, "On searching relevant studies in software engineering," in *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'10. Swinton, UK, UK: British Computer Society, 2010, pp. 111–120. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227057.2227071

[74] M. Maggio, C. Klein, and K.-E. Årzén, "Control strategies for predictable brownouts in cloud computing," in *IFAC Proceedings Volumes*, vol. 47, no. 3, 2014, pp. 689 – 694.

[75] J. Durango, M. Dellkrantz, M. Maggio, C. Klein, A. Papadopoulos, F. Hernandez-Rodriguez, E. Elmroth, and K.-E. Arzen, "Control-theoretical load-balancing for cloud applications with brownout," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, Dec 2014, pp. 5320–5327.

[76] C. Klein, A. Papadopoulos, M. Dellkrantz, J. Durango, M. Maggio, K.-E. Arzen, F. Hernandez-Rodriguez, and E. Elmroth, "Improving cloud service resilience using brownout-aware load-balancing," in *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*, Oct 2014, pp. 31–40.

[77] D. Desmeurs, C. Klein, A. Papadopoulos, and J. Tordsson, "Event-driven application brownout: Reconciling high utilization and low tail response times," in *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, Sept 2015, pp. 1–12.

[78] A. Filieri, C. Ghezzi, A. Leva, and M. Magio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'11. IEEE Computer Society, 2011, pp. 283–292.

[79] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Reliability-driven dynamic binding via feedback control," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 43–52.

[80] D. Arcelli, V. Cortellessa, A. Filieri, and A. Leva, "Control theory for model-based performance-driven software adaptation," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, ser. QoSA '15. New York, NY, USA: ACM, 2015, pp. 11–20.

[81] H. Hoffmann, "CoAdapt: Predictable behavior for accuracy-aware applications running on power-aware systems," in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, July 2014, pp. 223–232.

[82] S. Shevtsov, M. U. Iftikhar, and D. Weyns, "SimCA vs Activ-FORMS: Comparing control- and architecture-based adaptation on the tas exemplar," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, ser. CTSE 2015. New York, NY, USA: ACM, 2015, pp. 1–8.

[83] T. Patikirikorala, L. Wang, and A. Colman, "Towards optimal performance and resource management in web systems via model predictive control," in *Australian Control Conference (AUCC)*, 2011, pp. 469–474.

[84] T. Patikirikorala, L. Wang, A. Colman, and J. Han, "Hammerstein-wiener nonlinear model based predictive control for relative QoS performance and resource management of software systems," *Control Engineering Practice*, vol. 20, no. 1, pp. 49 – 61, 2012.

[85] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "An evaluation of multi-model self-managing control schemes for adaptive performance management of software systems," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2678 – 2696, 2012, self-Adaptive Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212001628

[86] S. Abdelwahed, N. Kandasamy, and S. Neema, "Online control for self-management in computing systems," in *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, May 2004, pp. 368–375.

[87] N. Kandasamy, S. Abdelwahed, and M. Khandekar, "A hierarchical optimization framework for autonomic performance management of distributed computing systems," in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, 2006, pp. 9–9.

[88] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 1, pp. 80–96, 2002.

[89] F. Křikava, P. Collet, and R. Rouvoy, "Integrating adaptation mechanisms using control theory centric architecture models: A case study," in *11th International Conference on Autonomic Computing (ICAC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 25–32.

[90] F. Křikava, P. Collet, and R. B. France, "Actress: Domain-specific modeling of self-adaptive software architectures," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ser. SAC '14. ACM, 2014, pp. 391–398.

[91] M. M. Kokar, K. Baclawski, and Y. A. Eracar, "Control theory-based foundations of self-controlling software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 37–45, May 1999. [Online]. Available: http://dx.doi.org/10.1109/5254.769883

[92] N. Fescioglu-Unver and M. M. Kokar, "Self controlling tabu search algorithm for the quadratic assignment problem," *Computers and Industrial Engineering*, vol. 60, no. 2, pp. 310 – 319, 2011.

[93] Y. A. Eracar and M. M. Kokar, "Self-control of the time complexity of a constraint satisfaction problem solver program." *Journal of Systems and Software*, vol. 85, no. 12, pp. 2697–2706, 2012.

[94] J. Cangussu, A. Mathur, and R. DeCarlo, "Feedback control of the software test process through measurements of software reliability," in *12th International Symposium on Software Reliability Engineering, ISSRE*, Nov 2001, pp. 232–241.

[95] S. D. Miller, R. A. DeCarlo, A. P. Mathur, and J. W. Cangussu, "A control-theoretic approach to the management of the software system test phase." *Journal of Systems and Software*, vol. 79, no. 11, pp. 1486–1503, 2006.

[96] K.-Y. Cai, Y.-C. Li, and W.-Y. Ning, "Optimal software testing in the setting of controlled markov chains," *European Journal of Operational Research*, vol. 162, no. 2, pp. 552 – 579, 2005.

[97] K.-Y. Cai, B. Gu, H. Hu, and Y.-C. Li, "Adaptive software testing with fixed-memory feedback," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1328 – 1348, 2007.

[98] Y. Ji, V. S. Mookerjee, and S. P. Sethi, "Optimal software development: A control theoretic approach," *Information Systems Research*, vol. 16, no. 3, pp. 292–306, 2005.

[99] Y. Ji, S. Kumar, V. S. Mookerjee, S. P. Sethi, and D. Yeh, "Optimal enhancement and lifetime of software systems: A control theoretic analysis," *Production and Operations Management*, vol. 20, no. 6, pp. 889–904, 2011.

[100] J. W. Cangussu, K. Cooper, and C. Li, "A control theory based framework for dynamic adaptable systems," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04. New York, NY, USA: ACM, 2004, pp. 1546–1553. [Online]. Available: http://doi.acm.org/10.1145/967900.968209

[101] V. Zanardi and L. Capra, "Dynamic updating of online recommender systems via feed-forward controllers," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 11–19.

[102] X. Peng, B. Chen, Y. Yu, and W. Zhao, "Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2707 – 2719, 2012, self-Adaptive

Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412121200132X
[103] N. Villegas, G. Tamura, H. Müller, L. Duchien, and R. Casallas, "DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science, R. de Lemos, H. Giese, H. Müller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 265–293.
[104] G. Mencagli, M. Vanneschi, and E. Vespa, "A cooperative predictive control approach to improve the reconfiguration stability of adaptive distributed parallel applications," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 1, pp. 2:1–2:27, 2014.
[105] G. Cao and A. A. Ravindran, "Energy efficient soft real-time computing through cross-layer predictive control," in *9th International Workshop on Feedback Computing (Feedback Computing 14)*. Philadelphia, PA: USENIX Association, 2014.
[106] K. Angelopoulos, A. V. Papadopoulos, V. E. Silva Souza, and J. Mylopoulos, "Model predictive control for software systems with CobRA," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16. New York, NY, USA: ACM, 2016, pp. 35–46. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897054
[107] N. Karlsson and J. Zhang, "Applications of feedback control in online advertising," in *2013 American Control Conference*, June 2013, pp. 6008–6013.
[108] "Control problems in online advertising and benefits of randomized bidding strategies," vol. 30, 2016, pp. 31 – 49, 15th European Control Conference, ECC16.
[109] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, no. 9-10, pp. 833–859, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2008.01.006
[110] D. Weyns, M. Iftikhar, S. Malek, and J. Andersson, "Claims and supporting evidence for self-adaptive systems: A literature study," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, June 2012, pp. 89–98.
[111] O. Garpinger, T. Hägglund, and K. J. Åström, "Performance and robustness trade-offs in PID control," *Journal of Process Control*, vol. 24, no. 5, pp. 568 – 577, 2014. [Online]. Available: //www.sciencedirect.com/science/article/pii/S0959152414000730
[112] K. J. Åström, T. Hägglund, C. C. Hang, and W. K. Ho, "Automatic tuning and adaptation for PID controllers - a survey," *Control Engineering Practice*, vol. 1, no. 4, pp. 699 – 714, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/096706619391394C
[113] M. French, E. Rogers, and C. Szepesvari, *Performance of Nonlinear Approximate Adaptive Controllers*. New York, NY, USA: John Wiley & Sons, Inc., 2002.
[114] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, 1990.
[115] P. Zhang, *Advanced Industrial Control Technology*. Elsevier Science, 2010.
[116] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, *The literature review supporting material*. [Online]. Available: https://people.cs.kuleuven.be/danny.weyns/material/2016TSE/

Stepan Shevtsov is a doctoral student at the Department of Computer Science of Linnaeus University, Sweden. His main focus is on engineering self-adaptive software systems using principles from control theory.

Mihaly Berekmeri received his PhD in Automatics from the University of Grenoble, France in 2015. His research focuses on the development of control-theoretical tools for Big Data, distributed computing systems and cloud services.

Danny Weyns is a professor at the Department of Computer Science of the Katholieke Universiteit Leuven, Belgium and part time affiliated with Linnaeus University, Sweden. His main research interest is in software engineering of self-adaptive systems.

Martina Maggio is an assistant professor in the Department of Automatic Control, Lund University, Sweden. Her main research topic is the application of control theory to the design and implementation of computing systems.