SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2021;e1772
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.1772

Testing of adaptive and context-aware systems: approaches and challenges

Bento R. Siqueira[1,*,†], Fabiano C. Ferrari[1], Kathiani E. Souza[1], Valter V. Camargo[1] and Rogério de Lemos[2]

[1]*Computing Department, Federal University of São Carlos, São Carlos, Brazil*
[2]*School of Computing, University of Kent, Canterbury, UK*

SUMMARY

Adaptive systems (ASs) and context-aware systems (CASs) are able to evaluate their own behaviour and to *adapt* it when the system fails to accomplish its goals or when better functionality or performance is possible. Ensuring the reliability of ASs and CASs is demanding because failures might have undesirable consequences. Testing ASs and CASs effectively is not trivial because of the inherent characteristics of these systems. The literature lacks a comprehensive review that provides a broad picture of the area; current reviews are outdated and incomplete. The objectives of this study are characterizing the state of the art in AS and CAS testing and discussing approaches, challenges, observed trends, and research limitations and directions. We performed a systematic literature review (SLR) and a thematic analysis of studies, reporting up-to-date, refined and extended results when compared with existing reviews. Based on 102 selected studies, we (i) characterized testing approaches by grouping techniques for ASs and CASs; (ii) updated and refined a characterization of testing challenges for ASs and CASs; and (iii) analysed and discussed research trends and implications for AS and CAS testing. There are recurring research concerns regarding AS and CAS testing. Examples are the generation of test cases and built-in tests. Moreover, we also identified recurring testing challenges such as context monitoring and runtime decisions. Moreover, there are some trends such as model-based testing and hybrid techniques and some little investigated issues like uncertainty and prediction of changes. All in all, our results may provide guidance for developers and researchers with respect to the practice and the future research on AS and CAS testing. © 2021 The Authors. *Software Testing, Verification & Reliability* published by John Wiley & Sons Ltd.

Received 26 June 2020; Revised 31 December 2020; Accepted 7 March 2021

KEY WORDS: software testing; adaptive systems; context-aware systems; systematic literature review; systematic mapping study

1. INTRODUCTION

This article characterizes the state of the art regarding the testing of adaptive systems (ASs) and context-aware systems (CASs). It relies on the outcomes of a systematic literature review (SLR) and reports on, mainly: (i) the identification and description of testing approaches for ASs and CASs; (ii) the characterization of testing challenges in this context; and (iii) a discussion regarding current research on testing and its implications for the future. In a prior conference paper [1], we presented preliminary results regarding (ii) (i.e. testing challenges), which are updated and reanalysed in this article. To the best of our knowledge, the literature still lacks a comprehensive

*Correspondence to: Bento R. Siqueira, Computing Department, Federal University of São Carlos, Rodovia Washington Luis, Km 235, São Carlos, São Paulo, Brazil.
†Email: bentor.siqueira@gmail.com

review that provides a broad picture of the area; currently, the reviews that are available in literature are incomplete and outdated. This article is intended to fill this gap.

We highlight the fact that the evolution of technologies has changed the way of dealing with software. Issues such as monitoring the environment and adapting the system according to its context are current – and major – software demands [2]. These demands can be fulfilled by taking into consideration concepts as well as underlying technologies that are inherent to ASs and CASs [3]. Such systems have a central characteristic that differentiates them from conventional systems: they are able to evaluate their own behaviour and to *adapt* it when the evaluation indicates that the system is not accomplishing its goals or when better functionality or performance is possible [4]. In particular, a CAS is able to recognize and react – by modifying its behaviour – to changes that occur to the system environment [5], whereas an AS is able to modify its behaviour and/or structure in response to changes that occur to the system, its environment or even its goals [6]. In other words, an AS is also a CAS.

Adaptive systems and CASs share some characteristics.[1] Typically, the architecture of ASs and CASs can be semantically divided into a managed and a managing subsystems [2,7]. The managed subsystem comprises the application logic that provides the system's domain functionality [2]. The managing subsystem, on the other hand, manages the managed subsystem and is composed of the adaptation logic that deals with concerns regarding the managed subsystem. An AS is able to modify not only its behaviour but also its own structure to deliver a better functionality or to optimize its quality of service [4]. In a CAS, changes are usually performed by a middleware that resembles a managing subsystem of ASs.

Because of the aforementioned characteristics, designing and maintaining these systems are highly challenging [3]. In particular, ensuring the reliability of ASs and CASs is demanding because failures might have undesirable consequences [8]. In this context, software testing comes into play. However, in the context of ASs and CASs, effective testing is not trivial [9] because the inherent characteristics of these systems and traditional testing approaches are ineffective [8,10,11].

In general, traditional testing approaches deal with inputs and expected outputs as fixed static values. However, requirements of ASs and CASs, and their environment, may change [10]. A traditional approach should only be applied if the *changes* that happen are taken into account [8]. In addition, in traditional systems, failed tests can show that the system violates requirements; in such cases, developers can implement the corrections (on an isolated instance of the system to prevent introducing errors into a production environment [10]) and publish a new release. In ASs and CASs, differently, the system may deal by itself with monitoring to diagnose problems and self-reconfigure at runtime to achieve the same corrective objectives [10]. Furthermore, as in the example of a smart home system described by Fredericks *et al.* [10], tests can occur when a new element (e.g. a sensor) is added to the system. In a such scenario, tests can be conducted at runtime upon the live system, often in response to changes in its operational context. Thus, it implies in dealing at first with unpredictable situations (i.e. new sensors being added at runtime). This point is also addressed in our previous studies [1,12]. In summary, the two main issues that make *AS and CAS testing* a daunting task are (i) the combinatorial explosion of the number of adaptation alternatives, many of them probably unforeseen at design time, and (ii) the fact that many adaptations are performed at runtime. Even though the running environment can constrain the number of possible adaptations [13], the number of system configurations can still be too large and impractical from testing perspective.

Concerning AS and CAS testing, the literature includes a number studies with such a focus. This article identifies and compiles such research initiatives, based on results of an SLR that we performed with the aim of (i) identifying and characterizing testing approaches for ASs or CASs and (ii) characterizing challenges for testing these types of systems. In our previous paper [1], we addressed goal (ii) by analysing studies published until 2014. In this article, we address goal

[1]Based on the selected primary studies in our SLR, it is evident that researchers of AS and CAS testing share common concerns. In total, nine from 82 studies present in our final set cited (or were cited by) studies that were classified as AS related, as well as studies that were classified as CAS related. This is discussed in Section 5.3.

(i) and revisit goal (ii), so that the results for goal (ii) are largely updated and extended. More specifically, the contributions of this article are as follows:

- an up-to-date and original *grouping, overview and analysis* of 74 studies that investigate (either by proposing or applying) testing approaches for AS or CAS;
- an updated and revised *characterization of testing challenges* we devised from the analysis of 42 studies (against 25 studies analysed in our prior research [1]); the challenges are further classified between AS and CAS categories; and
- a *discussion of research limitations and directions* for AS and CAS testing.

We emphasize that through SLRs, researchers build a body of knowledge to improve the current methodological support to the state of the art and state of the practice. Researchers may also synthesize experiences and lessons learned by other researchers with varying levels of competency of empirical research [14]. In the context of AS and CAS testing, as a way to confirm the need for an up-to-date secondary study on the topic, we applied to our prior study [1] the framework originally proposed by Garner *et al*. [15] and evaluated in the context of software engineering by Mendes *et al*. [16]. The results in our case signalled positively for an updated and extended secondary study (details can be checked in Appendix A). Therefore, we believe that our SLR may support engineers and researchers of ASs and CASs to cope with the testing challenges by means of the definition of customized testing strategies with focus on recurring and inherent properties of ASs or CASs. It also brings new findings regarding testing approaches, challenges and research trends.

This section summarized the context, motivation, goals and contributions of this article. Section 2presents basic background regarding ASs and CASs. Section 2 also briefly discusses the relationship between ASs and CASs, describes the traditional Znn.com example and shortly relates testing with key properties of ASs and CASs. Related work is summarized in Section 3. Section 4 revisits the goals of this article and presents key elements of the SLR design. Section 5 summarizes the search results and presents initial data classifications. Section 6 presents an analysis of the selected studies, with focus on testing approaches and challenges for ASs and CASs; it also brings further discussions and research implications. Section 7 discusses limitations and threats to the validity of our study, and conclusions and future work are presented in Section 8. Appendix A justifies the need for either a new or an updated SLR on the topic we addressed in this paper. Finally, Appendix B presents the complete SLR protocol.

2. BACKGROUND

This section presents foundations for ASs and CASs, describes an AS example and briefly comments on software testing in the context of ASs and CASs.

2.1. Context, configuration and environment in adaptive systems and context-aware systems

By dealing with systems that are able to self-adapt, it is usually possible to identify characteristics and scenarios in which there are *context* changes that were emanated from monitoring events from the *environment*; this may lead to the adaptation of the system by properly choosing a new system *configuration* [17]. Abowd *et al*. [5] defined the term *context* as any information that can be used to characterize the situation of an entity, where an entity can be a person, a place, or a physical or computational object. Oreizy *et al*. [18] used the term *configuration* to refer to architectures composed of components and connectors. In this sense, while dealing with ASs and CASs, Oreizy *et al*. [18] addressed the need of having dynamic architectures (i.e. sets of the system's possible configurations). Hurtado *et al*. [17] used the term *environment* when there are events generated by monitoring hardware sensors or any physical device. Xu *et al*. [19] also addressed 'environment' by comparing internal environment (i.e. state) and external environment.

2.2. Adaptive systems

Adaptive systems are systems that are able to automatically modify themselves in response to changes in their environment [18,20]. According to Krupitzer *et al*. [21], the reason for adaptation is a change in one or various system's elements, such as (i) a change in the technical resources, for example, a software or a hardware fault; (ii) a change in the environment, for example, changes in the state of a context variable; or (iii) a change regarding the user preferences. To accomplish the adaptation there are four self-* features that can be considered fundamental. These features are self-configuration, self-healing, self-optimization and self-protection [20]. In short, such features can be described as follows: *self-configuration*: the AS configures itself in order to adapt dynamically to changing environments; *self-healing*: the AS detects, diagnoses and recovers itself from problems that threat its correct operation; *self-optimization*: the AS always seeks ways and seizes opportunities to improve its operation; and *self-protection*: the AS anticipates, detects, identifies and protects itself from internal and external threats.

Adaptive systems are composed by two interrelated subsystems: the managing subsystem and the managed subsystem [2,7]. To perform an adaptation, the former monitors, analyses, plans and executes changes on the managed subsystem; in other words, there is a structure that is based on the MAPE-K reference control model [20,22]. The latter, on the other hand, implements the application domain. Regarding the constituent parts of MAPE-K, monitors probe the managed subsystem by means of sensors to obtain the current state of context variables. Analysers correlate the context values received from monitors with reference values to decide about the need for adapting the system. Based on business policies, planners define the maintenance activities to be executed to adapt or evolve the system, and executors implement the set of activities defined by planners [23]. These components in the managing system communicate between themselves and with the managed system using a fifth component, the knowledge, to transfer and store system data [22].

2.3. Context-aware systems

Context-aware systems are systems that are able to automatically collect contextual data (or simply, the *context*) in order to know the state of the running system [24]. Based on the context, the systems may be able to make decisions according to the goals of its users. Each change in a system environment generates a new context, so that a *context variable* is an attribute of a system with respect to the environment [24]. A *context instance* is an instantiated context variable. Given a timeline t, each context is located in a specific time of t. Thus, as long as the environment changes, a new context is generated with an increment of t [8]. A *context flow* is given by a time series of several context instances. *Context diversity* measures the number of context changes inherent in a context flow [25]. Usually, the context diversity can be computed by applying the total of *Hamming distance* [26] in the context flows. Despite the existence of definitions for 'context' in the literature, in software engineering activities (e.g. software testing), Santos *et al*. [27] concluded that it is difficult characterizing and using context data due to issues such as the complexity and uncertainty[2] related to these types of systems.

2.4. The relationship between adaptive systems and context-aware systems

It is clear that ASs and CASs share some characteristics, but they also have their particularities. Usually, an AS is able not only to modify its behaviour but also to modify its own structure in order to deliver a better functionality or optimize its quality of service [4]. These modifications are performed by the managing system in the managed system. A CAS, on the other hand, is usually characterized as a system focusing on the behavioural changes [29]. These changes are usually performed by a middleware that resembles a managing system of ASs. Despite these differences, as mentioned in Section 1, ASs and CASs share the characteristic of being able to evaluate their own behaviour and adapt it to properly accomplish their goals, possibly with better performance [4]. We highlight that both types of systems share the feature of data monitoring to represent the environment in which the system is inserted. In general, we can find out that the term *context*

[2]Esfahani and Malek [28] characterized *uncertainty* in terms of difficulties such as accurately express system's quality preferences and sensors employed for monitoring that often have uncontrollable noise. The authors also mentioned the need to identify the different sources of uncertainty and dealing with them as first-class attributes.

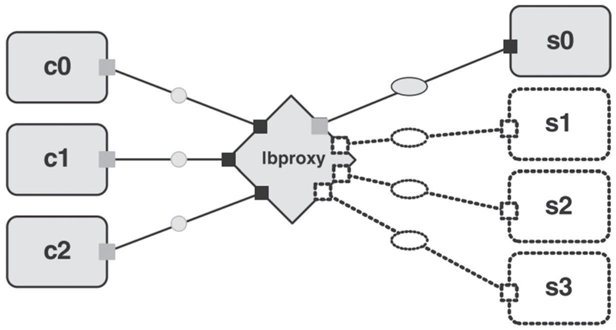

Figure 1. Znn.com system architecture [31] (c_i are clients, and s_i are servers).

awareness from CAS systems can be classified into the hierarchical levels of ASs properties [4]. Indeed, some authors even mention *context awareness* as a particular AS property [3]. As we mentioned in Section 1, a CAS is able to recognize and react to changes that occur to the system environment [5], whereas an AS is able to react to its environment to modify its behaviour and/or structure in response to changes that occur to the system, its environment or even its goals [6]. In this context, in this research, we considered both types of systems due to the interchangeable way several pieces of work in the literature deal with them. This is particularly noticed when we find cross-citations among studies from both communities (this is represented in Figure 5 and discussed in Section 5.3).

2.5. *An AS example: Znn.com*

Znn.com[3] [30] is a typical example of an AS used by the SEAMS[4] community. The system is able to reproduce the typical infrastructure for a news website. It consists of a three-tier architecture that has a set of servers providing contents from a back-end database to clients using front-end presentation interfaces. This system has a web-based client-server architecture that uses a load balancer to deal with requests across a pool of replicated servers, as shown in Figure 1. The number of servers can be adapted according to service demand. In addition, alternative levels of fidelity are provided according to the service demand (e.g. different versions and costs of resources such as usage of *text*, *images* or *videos*) to support their users having a properly experience. These attributes (i.e. scalability and levels of fidelity) allow the main goal for Znn.com to be achieved: providing content to customers within a reasonable response time while keeping the cost of the server within a range of operating budget. A short description of the scalability and fidelity adaptive properties is next presented.

2.5.1. Scalability. In case of a particular server be overload, new replicas of this server are able to be created using scales to increase resources. On the other hand, when a server replica is not demanded anymore, it can be destroyed. Thus, this attribute also brings elasticity to the system.

2.5.2. Fidelity. When the scalability is not feasible, the level of fidelity can be decreased. As an example, if the max limit number of replicas is achieved, the servers may start providing images in place of videos as a way to properly fulfil the service demand. Similar to the elasticity mentioned for scalability, once identified that the service demand has decreased, the level of fidelity can be increased.

Regarding characteristics of ASs and CASs, the service of supplying news is related to the Znn.com *managed system*. The attributes related to adaptive behaviour (i.e. scalability and fidelity) were developed as strategies and tactics inside of the Rainbow *managing system* [32].

[3]https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/model-problem-znn-com/– last checked in December 2020.
[4]https://www.self-adaptive.org/ – last checked in December 2020.

Cámara *et al*. [31] performed an evaluation study using Znn.com by focusing on the *Controller*component (i.e. the managing system). They demonstrated their concern and need for considering the system as a whole and that this would inevitably lead to new challenges during testing, such as the necessity to consider the full state of the target system. This issue is also emphasized in another study [33] of these authors. In our research, we identified a specific challenge (viz. SC-4) that is related to this issue (details are presented in Sections 5.2 and 6.2). More specifically, this regards the difficulty to test a system that does not have a clear boundary due to dynamic configurations and different contexts.

2.6. Software testing and adaptive systems/context-aware systems

Adaptive systems and CASs have been increasingly present in the peoples' social and professional lives [34]. Testing has shown to be a very effective way of improving the quality of systems in general, so it is a natural choice for assessing the quality of ASs and CASs. However, because of the likelihood of a combinatorial explosion of configurations of these types of systems, many of those unforeseen at specification and design time, testing ASs and CASs is a very difficult task for software developers (we address testing challenges in details in Section 6). As a consequence, traditional testing approaches are ineffective [8,11,32].

In software testing, there is a difference between fault, error and failure. A fault is an incorrect step, process or data definition. The execution of the fault may produce an inconsistent state (i.e. an error), which may lead to a failure. A failure occurs when the observed behaviour differs from the expected one [35]. Püschel *et al*. [36] limit this characterization to those that are relevant to a development-independent tester and modify it to ASs and CASs. So, according to Püschel *et al*., the only relevant property concerning faults is their persistence, which may be either permanent or transient. For example, the source of a fault in an AS or in a CAS can be in one of the managing subsystem's tasks, which involves monitoring, analysing, planning and executing in managed subsystems. A cyclic failure propagation can occur, for example, if a failure manifests in the system's knowledge model and influences on future decisions such that the failure becomes a fault in following cycles.

Salehie and Tahvildari [4] mentioned in their survey that 'testing' was the least focused phase in engineering ASs and CASs, with only a few studies addressing this topic. They argued that is challenging dealing with these types of systems, and they emphasized that such systems lead to several paths of execution in different scenarios (i.e. when one adds the dynamic decision making, the system will become even more complex). We highlight that inherent properties of ASs and CASs that impact on the way such systems should be tested, and the lack of studies in the literature that provide a landscape in testing ASs and CASs motivated us to perform the SLR that is presented in this article.

3. RELATED WORK

De Lemos *et al*. [6] performed a survey named *Software Engineering for Self-Adaptive Systems III Assurances*. The work is the third book of the series on *Software Engineering for Self-Adaptive Systems*. This series describes a wide range of approaches with respect to software engineering and control engineering. Comparing to our work, they aimed to give an overview about software engineering for self-adaptive systems including a software testing perspective, but not particularly focused on testing.

Matalonga *et al*. [37] performed an SLR in the same context of our SLR. Their results were split in two parts. The first one focused on identifying challenges faced in CAS testing, whereas the second one focused on mapping testing approaches to the ISO/IEC/IEEE 29119 standard.[5] Their studies [37-39] have not been selected as primary studies during our SLR because they were classified as secondary studies. In spite of that, we used them for snowballing purposes. Even

[5]Note that the study of Matalonga *et al*. [37] is a complete version of their work. Before the work of Matalonga *et al*. [37], the authors published partial results of their research [38,39].

though one of their research questions (viz. 'Which are the existing methods for testing context aware systems?') is related to ours, the main difference between both studies regards the inclusion criteria. They defined 10 criteria from two different categories. The first category focused on studies related to activities involving software testing. The second category focused on studies that involve characterizations of context in CASs and hence identifying problems with respect to human computing interaction and software systems' usability. With respect to the selection of studies from the literature, they used both categories to classify and to select the studies.

Santos *et al*. [27] performed an SLR to investigate test case design techniques for CASs. The authors addressed research questions involving techniques and challenges with respect to test case design. Comparing with our work, objectives partially overlap with respect to testing approaches and testing challenges, given that test case design techniques are one of the characteristics of testing approaches we retrieved. Focusing on test case design, the authors characterized a set of challenges that are related to the usage of context data. However, in our work, we deal with a more comprehensive scope of testing challenges and testing approaches. In addition, our search strategy takes into account not only CASs but also ASs. Note that the study of Santos *et al*. [27] was also used as seed in our snowballing search.

We also used the SLR of Matalonga *et al*. [38] for snowballing. Those authors analysed 12 studies in the context of CASs and identified two testing challenges. The first one is related to the use of devices with resource limitation, whereas the second one is related to the testing in a high variation of context data. However, in our work, we not only characterized a more comprehensive scope of challenges but also characterized the categories and testing techniques that have been used in the literature.

In another piece of related work, in a systematic mapping study, Almeida *et al*. [40] analysed 68 primary studies in order to identify Android testing tools from the literature. In addition, the authors also characterized which studies addressed the testing of Android CASs. We highlight that the authors defined research questions regarding both goals (Android testing tools and testing of Android CASs).

Comparing with our work, our study is more comprehensive with respect to deal with testing of ASs and CASs. In our characterizations, we do not focus on testing tools, whereas we identify testing approaches, their underlying techniques, problems and directions. In addition, they also do not address testing challenges as we do.

With respect to testing challenges, Fredericks *et al*. [10] defined a taxonomy of challenges for AS testing. However, the authors did not perform a broad literature search in order to build such taxonomy as we did in our work. In short, they discussed the difficulties software engineers may face with respect to the components of the MAPE-K model. We addressed different research questions, as detailed throughout this paper. Note that all testing challenges that were characterized by Fredericks *et al*. [10] were used in this work, so that these challenges are listed in Table 7.

Finally, in recent prior research [41], we characterized specific faults for ASs and CASs based on a subset of 11 studies [19,29,42-50] selected for the SLR we report in this article. As a result, we concluded that more attention in some characteristics (e.g. adaptation rules and error propagation in control loops) is necessary when devising fault-based testing approaches for these types of systems. More precisely, we presented a list of specific faults (26 in total) and fault type categories (six in total) for ASs and CASs. We noticed that (i) some faults can also clearly occur in common systems (e.g. missing static construct and component interface fault); (ii) some faults are specific to ASs and CASs (e.g. fault in data sensing and fault in adaptation rules); (iii) the most commonly discussed fault types are related to adaptation rules; (iv) the least commonly discussed fault type is related to environmental completeness; (v) some fault types are strictly related to structural and syntactic characteristics; and (vi) in general, the faults are more related to behavioural and semantical characteristics.

4. STUDY GOAL AND SETUP

This work aims to characterize the state of the art of AS testing and CAS testing. More specifically, in this article, we aim to (i) identify and characterize testing approaches for ASs or CASs; (ii)

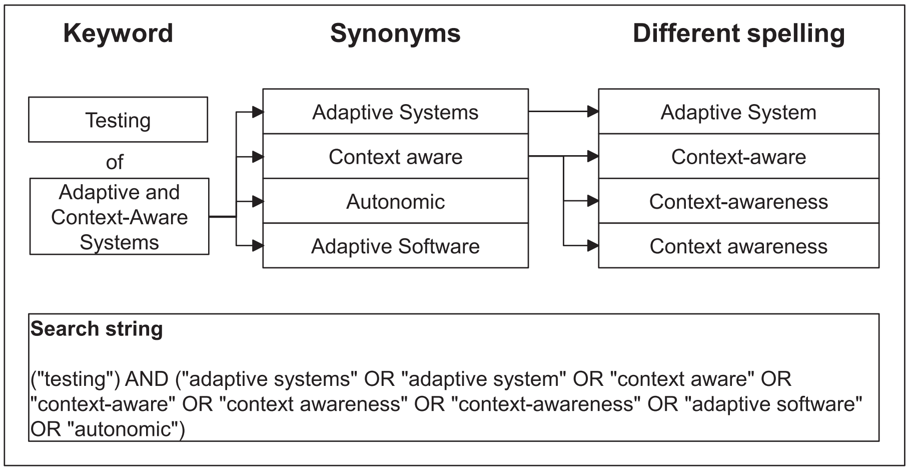
Keyword
Synonyms
Different spelling
Testing
of
Adaptive and Context-Aware Systems
Adaptive Systems
Context aware
Autonomic
Adaptive Software
Adaptive System
Context-aware
Context-awareness
Context awareness
Search string
("testing") AND ("adaptive systems" OR "adaptive system" OR "context aware" OR "context-aware" OR "context awareness" OR "context-awareness" OR "adaptive software" OR "autonomic")

Figure 2. Terms that form the search string and the snowballing.

enhance the characterization of challenges for AS testing or CASs testing; and (iii) discuss how challenges can be addressed with the identified approaches.

The process we followed to conduct the SLR is aligned with the one summarized by Kitchenham and Charters [51]. It includes three main phases: planning, conducting and reporting. Even though these phases could be seen as sequential, in general, each phase can be applied in several iterations [52]. The planning phase should evaluate the need for a new review, specify the research questions and develop the review protocol. The conducting phase should involve the selection of studies supported by a set of inclusion and exclusion criteria and deal with the data extraction of the selected studies. Finally, the reporting phase should involve data synthesis methods that support the summarization of important information to answer the research questions.

4.1. Method and research questions

We designed a study protocol that follows well-established SLR guidelines [51,52]. The full protocol can be found in Appendix B. Key points are next presented, starting with the research questions.

- *RQ1: Which are the testing approaches that are proposed for ASs or CASs?* This research question aims to identify and characterize which testing approaches have been either proposed for or applied to ASs or CASs.
- *RQ2: Which are the testing challenges for ASs or CASs?* This research question aims to identify and characterize which are the challenges faced by researchers and practitioners while developing and applying tests to ASs or CASs.

Note that in a prior conference paper [1], question RQ2 was answered with the analysis of primary studies published up to 2014; in this article, we largely increase the set of analysed studies to provide an up-to-date answer to RQ2.

4.2. Search string

We built the search string considering the three main terms: *testing*, and *ASs* or *CASs*. To define the search string – which is presented in Figure 2 – and to evaluate the retrieved studies, we used as a control group the set of studies selected by Ferrari *et al*. [12]. Figure 2 presents the variations of the terms (keywords), using synonyms and different spellings. With our string, we were also able to identify studies that apply testing on systems that share common properties with ASs and CASs (e.g. pervasive systems, ubiquitous systems and autonomous systems).

We highlight that there is a lack of standard terminology for these types of systems; consequently, one may retrieve studies of interest that employ different terms for similar concepts. In general, we noticed variations in the terminology even for basic concepts of AS or CAS (e.g. *self-adaptive system*, *self-awareness system*, *ubiquitous system* and *pervasive system*). However, it is important to mention

Table 1. Number of retrieved items with automatic search and snowballing.

ID	Source	Type of source	# retrieved		# analysed	
		Rounds	1 and 2	3	1 and 2	3
1	IEEE Xplore	Indexed base	162	329	36	221
2	ScienceDirect	Indexed base	149	269	8	230
3	SpringerLink	Indexed base	738	1081	18	977
4	ACM DL	Hybrid	195	887	39	802
5	Web of Science	Search engine	933	1712	26	1212
6	Scopus	Search engine	n/a	1098	n/a	853
	Subotal		**2177**	**5376**	**127**	**4295**
7	Snowballing	Citations and references	30	2058	17	1279
	Total		**2207**	**7434**	**144**	**5574**

that the generality and specificity [53,54] involving the definition of a search string could result in a restrict result from the literature or even retrieving a high number of irrelevant studies. Thus, to mitigate issue related to the lack of standard terminology, and to expand our search results, we have used a comprehensive snowballing approach that is next described as phase 5 in Section 4.7.

4.3. Search strategy

It comprised (i) performing automatic search using the search string; (ii) applying backward and forward snowballing [55]; and (iii) querying recent editions of specific journal issues of conference proceedings. Note that snowballing was not completely applied (or not applied at all) in the prior rounds[6] of this SLR. More specifically, in round 2 [1], we only retrieved 30 references from a small subset of selected studies, from which 17 were analysed, as shown in line 7 of Table 1. Therefore, with proper execution of snowballing in the most recent rounds, we extended the search strategy and fetched a wider set of studies when compared with our previous report [1].

4.4. Repositories[7]

The string was customized to the search engines of the selected information sources listed in Table 1, namely, IEEE Xplore,[8] Elsevier ScienceDirect,[9] Springer SpringerLink,[10] ACM Digital Library,[11] Clarivate Web of Science[12] and Elsevier Scopus.[13] The last automatic search was run in December 2019 (items 1 to 6 in the table). Note that Scopus was used only in the most recent automatic search round (round 3) and hence also contributed to fetch a wider set of studies when compared with previous research. Also note that Table 1 presents the numbers of retrieved and analysed studies for all rounds, including the ones performed in our previous work, namely, rounds 1 [12] and 2 [1].

4.5. Inclusion criteria

A study was selected if it fulfilled at least one of the three criteria listed in the sequence. Only studies that are peer reviewed, published in either a conference or a scientific journal, and written in English were analysed in the light of the inclusion criteria. As mentioned in Section 3, the dataset produced in this SLR was already used in the analysis of fault types of ASs and CASs [41]; therefore, in order to keep the dataset consistent across our publications, we also show criterion (iii) throughout this paper.

[6]In this work, we define *round* as a procedure of performing all phases of a systematic literature review in order to synthesize answers to the research questions.
[7]*Indexed base* means a source that indeed stores the study (e.g. IEEE Xplore and Elsevier ScienceDirect). *Search engine* means a tool that queries various indexed bases (e.g. Scopus and Web of Science). *Hybrid* is a combination of *indexed base* and *search engine* (e.g. ACM Digital Library).
[8]https://ieeexplore.ieee.org/ – last checked in December 2020.
[9]https://www.sciencedirect.com/ – last checked in December 2020.
[10]https://link.springer.com/ – last checked in December 2020.
[11]https://dl.acm.org/ – last checked in December 2020.
[12]https://www.webofknowledge.com/ – last checked in December 2020.
[13]https://www.scopus.com/ – last checked in December 2020.

1 It defines or applies testing approaches to ASs or CASs.
2 It characterizes challenges for AS testing or CAS testing.
3 It characterizes types of faults that are specific to ASs or CASs.

4.6. Complementary search – snowballing

Beyond performing snowballing over all selected studies, references from and citations to, secondary studies that are related to ours were also analysed. Note that, even though secondary studies usually do not produce primary evidence, they are important source of primary studies that might pass in the inclusion criteria.

4.7. Selection of studies

Initially, we reinforce that only peer-reviewed studies (from journal and proceedings) were considered in our work. This required a revision of the final selection of our second round [1], and two studies were removed from the final set of selected studies. That said, the selection comprised the phases next described. Note that the number of items and studies presented for each phase represents total number considering all rounds of the SLR. Part of the numbers are also shown in Table 1.

A recent update of our dataset is reported as phase 7 in what follows:

- **Phase 1 – search string:** customizing the base search string to be applied to the search engines. The search retrieved 7553 items[14] (i.e. subtotals 2177 and 5376).
- **Phase 2 – removing duplicated items:** this phase consisted in removing 3131 duplicated items retrieved from sources 1 to 6 mentioned earlier. Therefore, 4422 remained to be analysed (i.e. subtotals 127 and 4295).
- **Phase 3 – preselection:** preselection of studies based on title, abstract and keywords, to which the inclusion and exclusion criteria were applied. This resulted in 532 preselected studies.
- **Phase 4 – final selection:** in this final selection step, studies were fully analysed towards the final selection decision. Data of interest were extracted and stored in customized forms. This phase resulted in 78 selected studies. Once again, the inclusion and exclusion criteria were applied.
- **Phase 5 – backward and forward snowballing:** application of all iterations of backward and forward snowballing to the 78 studies selected in the previous phase. This involved retrieving additional 2088 items (i.e. subtotals 30 and 2058). After removing 779 duplicate items, 1309 studies were analysed following the steps of phase 3. We ended up adding 23 new studies to our final set; that is, we reached a total of 101 studies.
- **Phase 6 – identifying overlappings:** after analysing the 101 studies, we identified 19 overlaps among them. This includes studies (herein called *subsumed* studies) that are either updated or improved by their authors, so that new studies are published and subsume the previous ones. Therefore, the final output was a set of 82 primary studies.
- **Phase 7 – updating the search:** in this phase, which we refer to as round 4, we updated our search with focus on particular conference proceedings and journals. The choice for conferences and journals was guided by (i) the ones that were most addressed by authors of selected papers and (ii) representative venues for research on ASs or CASs testing. Table 2 presents the conference proceedings from which the studies were mostly selected, considering up to phase 6 described earlier; the most recurring are SEAMS (four studies), ICTSS and SAC (three studies each). Table 3 shows that up to phase 6, no more than one study was selected from a particular journal. Both tables were used to support the search for new studies that were published after June 2019 (i.e. after our last automatic search) and hence should be also analysed. To complement the set of analysed proceedings and journals, we used the ones that are representative for the software testing/reliability and for the ASs or CASs communities, as well

[14]We use the term *item* instead of *study* because not all search results are indeed studies. Some are proceedings front matters, talk abstracts and so forth.

Table 2. Proceedings in which selected studies were published (partial list, sorted in descending order of numbers of published studies).

Rank	Proceeding	Initials	Studies
1	International Symposium on Software Engineering for Adaptive and Self-Managing Systems	SEAMS	4
2	International Conference on Testing Software and Systems	ICTSS	3
3	Symposium on Applied Computing	SAC	3
4	International Aerospace Conference	AEROCONF	2
5	Annual International Computer Software and Applications Conference	COMPSAC	2
6	International Symposium on Foundations of Software Engineering	FSE	2
7	International Conference on Adaptive and Self-Adaptive Systems and Applications	ICAS	2
8	International Conference on Software Engineering	ICSE	2
9	International Conference on Software Engineering and Service Science	ICSESS	2
10	International Workshop on Mutation Analysis	MUTATION	2
11	Workshop on Quality Assurance for Self-Adaptive, Self-Organising Systems	QA4SASO	2
12	International Conference on Self-Adaptive and Self-Organizing Systems	SASO	2
…	…	…	…

Table 3. Journals in which selected studies were published.

Rank	Journal	Initials	Studies
1	ACM Transactions on Autonomous and Adaptive Systems	TAAS	1
2	Software Quality Journal	SQJ	1
3	IEEE Transactions on Software Engineering	TSE	1
4	International Journal of Business Research and Information Technology	IJBRIT	1
5	International Journal of Software Engineering and Knowledge Engineering	JSEKE	1
6	International Journal Software Informatics	JSI	1
7	Software & Systems Modeling	JSM	1
8	Journal of Systems and Software	JSS	1
9	Open Journal of Web Technologies	OJWT	1
10	IEEE Computer	IEEE Computer	1
11	IEEE Transactions on Dependable and Secure Computing	TDSC	1
12	IEEE Transactions on Reliability	TR	1
13	International Journal On Advances in Software	JAS	1
14	IT Professional	ITP	1
15	Journal of Computers	JC	1

software engineering in general, namely, ICST,[15] ISSTA,[16] ISSRE,[17] STVR,[18] SASO,[19] TAAS,[20] CASM,[21] FSE,[22] ICSE,[23] TSE,[24] TOSEM[25] and SQJ.[26] From these proceedings and journals, we preselected 20 studies, from which only one new study [56] was selected. After applying the snowballing techniques to it, no more studies were selected. Therefore, we ended up with a final set of 83 selected primary studies that were analysed and from which data were extracted, as presented in Tables 5 and 6. Note that the subsumed studies do not

[15]IEEE International Conference on Software Testing, Verification and Validation.
[16]ACM SIGSOFT International Symposium on Software Testing and Analysis.
[17]IEEE International Symposium on Software Reliability Engineering.
[18]Journal of Software Testing, Verification & Reliability.
[19]IEEE International Conference on Self-Adaptive and Self-Organizing Systems.
[20]ACM Transactions on Autonomous and Adaptive Systems.
[21]Journal of Complex Adaptive Systems Modeling.
[22]ACM Symposium on the Foundations of Software Engineering.
[23]IEEE/ACM International Conference on Software Engineering.
[24]IEEE Transactions on Software Engineering.
[25]ACM Transactions on Software Engineering and Methodology.
[26]Software Quality Journal.

Table 4. Examples of author's text snippets of challenges and difficulties involving testing of ASs and CASs with respect to specific challenge SC-1.

ID	Author	Text snippet
1	Truszkowski *et al*. [58]	'***The number of these combinations is exponential*** *and sometimes factorial to the number of states. Consequently, the state space is too large to test'*.
2	Schumann and Visser [59]	'***The larger size*** *and higher complexity of the valid* ***input space****, which can contain system status, environmental information, intended goals, and constraints*'.
3	Niebuhr and Rausch [60]	'***Proving the correctness*** *of ASs* ***at runtime*** *in general is not possible. You get a less valuable proof in case of an* ***incomplete specification***'.
4	Welsh and Sawyer [61]	'*It is necessary* ***not only to verify*** *that the AS operates as desired* ***in a given state, but that*** *the ASs adapts its behaviour appropriately* ***in environment changes***'.
5	Tse *et al*. [62]	'***Combinatory explosion of unforeseeable combinations of intermediate contexts*** *to trigger subsequent context-sensitive functions*'.
6	Micskei *et al*. [63]	'*The context is complex and* ***there are a large number of possible situations****: in real physical world the number and types of potential context objects, attributes and interactions that need to be specified can be large*'.
7	Vieira *et al*. [64]	'***There is a wide range of values that can be adapted****. As a result, there are* ***infinite potential test cases****. The challenge is to find a mechanism to abstract context sources and derive test cases based on the most critical values*'.

belong to our final output, given that their results can also be found in other studies in the final output.

4.8. Analysis and synthesis

To answer RQ1 and RQ2, we applied thematic analysis [57] in the selected studies. It involved identifying recurring themes in the literature and summarizing the findings based on different thematic headings. Specifically for RQ1, in Table 8, the classification and subclassification shown in columns *Technique* and *Category*, respectively, were constructed during SLR execution. With respect to testing techniques (first column of the table), we have focused on classifying the approaches as functional, structural, model based and fault based. The studies that have addressed two or more of these techniques were classified as *hybrid*, whereas the remaining studies were classified as *other* (brief descriptions of the techniques are provided in Sections 6.1.1 to 6.1.6). The categories identified for the subclassification can be observed in the second column of Table 8.

With respect to RQ2, the analysis of the primary studies helped us characterize thematic headings in the context of testing challenges involving ASs or CASs. We have also defined categories to support the data extraction and analysis of testing challenges. These categories are presented in Table 10. An example of the definition of thematic headings is presented in Table 4; in that table, we can observe the frequent usage of closely related terms: *exponential number of combinations*, *large size of input space*, *runtime with an incomplete specification*, *verify environment changes* and *infinite potential test cases*. Based on these text snippets and terms, we established the SC-1 specific challenge (listed in Table 7) as 'The impossibility to guarantee the correctness of a changing system with unpredictable and growing number of contexts and configurations'.

5. SEARCH RESULTS AND INITIAL DATA CLASSIFICATION

The final set of selected studies, respectively for each round, includes 11, 21, 50 and 1 studies (83, in total). Thus, this work largely increases the number of selected studies with 51 studies added to the set, particularly when compared with our prior paper that specifically addressed challenges for AS or CAS testing [1].

Table 5. First part of selected primary studies (in all rounds, sorted by year of publication).

Authors and ref. number	Year	R	System	Database	(i)	(ii)	(iii)	Overlapping	Information source
Kephart and Chess [20]	2003	2	AS	IEEE Xplore		x			IEEE Xplore, snowballing
Flores *et al*. [65]	2004	1	CAS	IEEE Xplore	x				IEEE Xplore, Web of Science
Tse *et al*. [62]	2004	2	CAS	IEEE Xplore	x	x			IEEE Xplore, snowballing
Truszkowski *et al*. [58]	2004	3	AS	IEEE Xplore		x			IEEE Xplore, snowballing
Lu *et al*. [24]	2006	1	CAS	ACM DL	x	x			ACM DL, Scopus, snowballing
Chan *et al*. [66]	2006	2	CAS	World Scientific	x			[67]	Web of Science, Scopus, snowballing
Merdes *et al*. [68]	2006	2	CAS	ACM DL	x				Snowballing
Schumann and Visser [59]	2006	3	AS	IEEE Xplore		x			Snowballing
Niebuhr and Rausch [60]	2007	1	AS	ACM DL	x	x			ACM DL, Scopus, snowballing
Wang *et al*. [69]	2007	1	CAS	IEEE Xplore	x	x			IEEE Xplore, ACM DL, Web of Science, snowballing
King *et al*. [70]	2007	2	CAS	Academy Publisher	x	x			Scopus, snowballing
Lu *et al*. [71]	2008	1	CAS	IEEE Xplore	x	x			IEEE Xplore, ACM DL, Scopus, snowballing
Sama *et al*. [45]	2008	1	CAS	ACM DL	x		x		ACM DL, Scopus
Jaw *et al*. [11]	2008	2	AS	IEEE Xplore	x	x			IEEE Xplore
Niebuhr *et al*. [72]	2009	1	AS	IEEE Xplore	x	x		[73,74]	IEEE Xplore, Web of Science, Scopus
Ye *et al*. [75]	2009	2	CAS	ACM DL	x				ACM DL, Scopus
Taranu and Tiemann [76]	2009	3	CAS	ACM DL	x				ACM DL, Web of Science, Scopus, snowballing
Sama *et al*. [29]	2010	1	CAS	IEEE Xplore	x	x	x		IEEE Xplore, Web of Science, Scopus, snowballing
Sama *et al*. [46]	2010	1	CAS	ScienceDirect			x		ScienceDirect, Web of Science, ACM DL, Scopus, snowballing
Wang *et al*. [48]	2010	1	CAS	IEEE Xplore	x		x	[77]	IEEE Xplore, snowballing
Welsh and Sawyer [61]	2010	1	AS	IEEE Xplore	x	x			IEEE Xplore, Scopus, snowballing
Vassev *et al*. [78]	2010	2	AS	IEEE Xplore	x	x			IEEE Xplore, Scopus
Da Costa *et al*. [79]	2010	3	AS	ACM DL	x				ACM DL, Scopus, snowballing

(*Continues*)

Table 5. (Continued)

Authors and ref. number	Year	R	System	Database	(i)	(ii)	(iii)	Overlapping	Information source
King *et al*. [80]	2011	2	AS	IEEE Xplore	x	x		[9,81-84]	IEEE Xplore, Web of Science, Scopus
Silva and De Lemos [85]	2011	2	AS	ACM DL	x				ACM DL, Scopus
Eze *et al*. [86]	2011	3	AS	ThinkMind		x			Snowballing
Bartel *et al*. [42]	2011	2	CAS	IEEE Xplore	x		x		IEEE Xplore, snowballing
Wotawa [87]	2012	2	AS	SpringerLink	x	x			SpringerLink, Web of Science, Scopus, snowballing
Micskei *et al*. [63]	2012	2	CAS	SpringerLink	x	x			SpringerLink, ACM DL, Scopus, snowballing
Püschel *et al*. [88]	2012	2	CAS	Semantic Scholar	x	x			Scopus, snowballing
Weyns [89]	2012	2	AS	ACM DL	x	x			ACM DL, Scopus
Bayha *et al*. [90]	2012	3	AS	ACM DL		x			ACM DL, snowballing
Xu *et al*. [19]	2012	3	CAS	ScienceDirect	x	x	x		Snowballing
Fredericks *et al*. [10]	2013	2	AS	IEEE Xplore	x	x			IEEE Xplore, Web of Science, ACM DL, Scopus, snowballing
Amalfitano *et al*. [91]	2013	3	CAS	IEEE Xplore	x				IEEE Xplore, Scopus, snowballing
Garvin *et al*. [92]	2013	3	AS	SpringerLink	x			[93]	SpringerLink, Scopus
Horányi *et al*. [94]	2013	3	CAS	Hal.archives-ouvertes	x			[95]	Snowballing
Nehring and Liggesmeyer [96]	2013	3	AS	ThinkMind	x	x			Snowballing
Silva and De Lemos [97]	2013	3	AS	KAR Repository	x				Snowballing
Xu *et al*. [98]	2013	3	CAS	HKUST Repository	x	x		[99]	Snowballing
Akour *et al*. [100]	2014	3	AS	Academic OneFile	x			[101]	Snowballing
Eberhardinger *et al*. [102]	2014	2	CAS	SpringerLink	x	x			SpringerLink, ACM DL, Web of Science, Scopus, snowballing
Subtotals (41 studies)					**36**	**26**	**6**	**13**	

Tables 5 and 6 list all selected studies. In both tables, the round number is shown in the column labelled with 'R'. The tables also show the author(s) and the reference number (column 'Authors and ref. number'); year of publication (column 'Year'); type of system (AS or CAS) *as originally reported by the authors* (column 'System'; more details about this classification are provided along this paper); the database from which the study was downloaded (column 'Database'); the number of the inclusion criteria fulfilled by the study (columns '(i)', '(ii)' and '(iii)'); the subsumed studies, that is, studies that were updated by those in the first column (column 'Overlapping'); and the

Table 6. Second part of selected primary studies (in all rounds, sorted by year of publication).

Authors and ref. number	Year	R	System	Database	(i)	(ii)	(iii)	Overlapping	Information source
Fredericks *et al*. [106]	2014	2	AS	ACM DL	x				ACM DL, Web of Science, Scopus, snowballing
Griebe and Gruhn [107]	2014	2	CAS	ACM DL	x	x			ACM DL, Scopus, snowballing
Wang *et al*. [8]	2014	2	CAS	ACM DL	x			[25]	ACM DL, Web of Science, Scopus, snowballing
Püschel *et al*. [108]	2014	3	AS	ThinkMind	x	x			Snowballing
Püschel *et al*. [109]	2014	3	AS	ThinkMind	x	x		[36]	Snowballing
Püschel *et al*. [44]	2014	3	AS	ThinkMind	x	x	x		Snowballing
Cámara *et al*. [110]	2015	2	AS	IEEE Xplore	x	x		[31,33]	IEEE Xplore
Vieira *et al*. [64]	2015	3	CAS	ACM DL		x			ACM DL, Web of Science, Scopus
Fredericks and Cheng [111]	2015	3	AS	IEEE Xplore	x				IEEE Xplore, ACM DL, Scopus, snowballing
Hansel *et al*. [112]	2015	3	AS	IEEE Xplore	x				IEEE Xplore, Web of Science, Scopus, snowballing
Lahami *et al*. [113]	2015	3	AS	SpringerLink	x				SpringerLink, Web of Science, Scopus
Lim *et al*. [114]	2015	3	AS	IEEE Xplore	x				IEEE Xplore, Scopus
Majchrzak and Matthias [115]	2015	3	CAS	RonPub	x				Snowballing
Markov and Fröhlich [116]	2015	3	AS	IEEE Xplore	x				IEEE Xplore, Web of Science, Scopus
Sen *et al*. [117]	2015	3	AS	IEEE Xplore	x				IEEE Xplore, Web of Science, Scopus
Song *et al*. [118]	2015	3	CAS	Semantic Scholar	x				Scopus
Tonjes *et al*. [119]	2015	3	CAS	IEEE Xplore	x				IEEE Xplore, Scopus
Al-Refai *et al*. [120]	2016	3	AS	IEEE Xplore	x				IEEE Xplore, Web of Science, Scopus, snowballing
Al-Refai *et al*. [121]	2016	3	AS	IEEE Xplore	x				Snowballing
Eberhardinger *et al*. [43]	2016	3	AS	SpringerLink	x	x	x	[103]	SpringerLink, Scopus, snowballing
Heck *et al*. [122]	2016	3	AS	IEEE Xplore		x			Snowballing
Qin *et al*. [123]	2016	3	CAS	ScienceDirect	x	x			Snowballing
Rodrigues *et al*. [124]	2016	3	CAS	ACM DL	x				ACM DL, Scopus
Wotawa [125]	2016	3	AS	IEEE Xplore	x				IEEE Xplore, Scopus
Yu *et al*. [50]	2016	3	CAS	IEEE Xplore	x	x	x	[49]	IEEE Xplore, Web of Science, Snowballing
Lindvall *et al*. [126]	2017	3	AS	IEEE Xplore	x				ACM DL
Luo *et al*. [127]	2017	3	SSC	IEEE Xplore	x	x			Scopus
Matalonga and Travassos [128]	2017	3	CAS	ACM DL	x				ACM DL, Scopus, snowballing
Mehmood *et al*. [129]	2017	3	CAS	IEEE Xplore	x				IEEE Xplore, Scopus, snowballing

(*Continues*)

Table 6. (Continued)

Authors and ref. number	Year	R	System	Database	(i)	(ii)	(iii)	Overlapping	Information source
Usaola *et al*. [130]	2017	3	CAS	IEEE Xplore	x				IEEE Xplore, Scopus, snowballing
Eberhardinger *et al*. [131]	2018	3	AS	IEEE Xplore	x				IEEE Xplore, Scopus, Web of Science, snowballing
Fredericks [132]	2018	3	AS	IEEE Xplore	x				IEEE Xplore, ACM DL, Scopus, Snowballing
Ma *et al*. [104]	2018	3	AS	SpringerLink	x	x			Snowballing
Mehmood *et al*. [105]	2018	3	CAS	IEEE Xplore	x	x			IEEE Xplore, Scopus, snowballing
Mirza and Khan [133]	2018	3	CAS	IEEE Xplore	x				IEEE Xplore, snowballing
Reichstaller and Knapp [134]	2018	3	AS	IEEE Xplore	x				IEEE Xplore, ACM DL, Scopus, Snowballing
Reichstaller *et al*. [135]	2018	3	AS	SpringerLink		x			Snowballing
Reichstaller *et al*. [136]	2018	3	AS	IEEE Xplore	x	x			Snowballing, ACM DL
Siqueira *et al*. [47]	2018	3	CAS	IEEE Xplore	x		x		ACM DL, Scopus, snowballing
Ma *et al*. [137]	2019	3	AS	SpringerLink	x	x			snowballing
Arcaini *et al*. [56]	2020	4	AS	IEEE Xplore	x				IEEE Xplore
Subtotals (42 studies)					**38**	**16**	**4**	**6**	
Totals (83 studies)					**74**	**42**	**10**	**19**	

information source from which the study was retrieved (column ‘Information source’; it may have been more than one source).

In the following, we will discuss the approaches and challenges; that is, the discussion regards inclusion criteria (i) and (ii).

5.1. Testing approaches for adaptive systems or context-aware systems

In total, 74 studies were selected with the application of criterion (i). Note that although there are 74 studies that apply at least one testing approach, this does not mean they all involve distinct approaches. As an example, Eberhardinger *et al*. [43] applied the same approach in two distinct studies [43,103]; the approach explores constraints and model-based testing to decrease the number of test cases to be generated. One of the studies [103] introduced the main concepts and used a running example, whereas the other [43] proposed a different test model – based on the first one – and presented results of an exploratory study.

We grouped the 74 studies based on their underlying testing techniques, namely, *functional testing* (12, in total), *structural testing* (6, in total), *model-based testing* (24, in total) and *fault-based testing* (8, in total). *Hybrid approaches* refer to studies that explored mixed techniques (13, in total), and 11 studies with general characteristics – for example, that focused on the definition of testing processes and frameworks – are described as *other approaches*. Figure 3 presents the number of studies for each testing technique. It shows the distribution of studies per technique before and since 2014 (note that this gives the reader a view of what has been on focus in the last 6 years). One observed trend is the application of combined (hybrid) techniques since 2014. Apart from it, model-based approaches are regularly the most explored technique. Moreover, structural and other techniques have decreased in ‘popularity’. Finally, the adoption of functional and fault-based testing has increased slightly. Details of the techniques and associated studies are provided in Section 6.1.

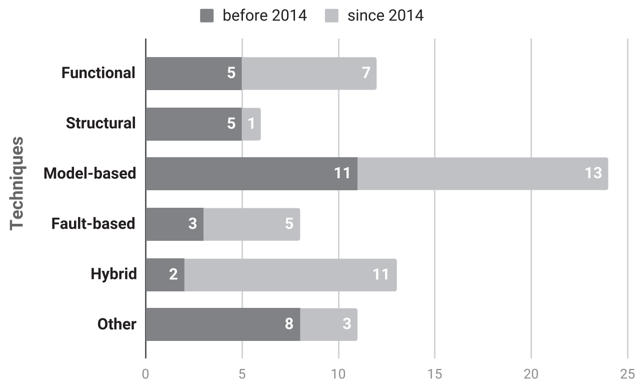

Figure 3. Number of studies per technique.

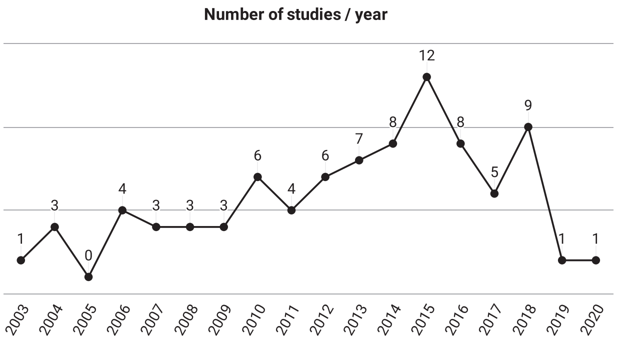

Figure 4. Number of studies per year.

The study distribution per year is shown in Figure 4. Overall, the number of publications has increased year by year, with slight variations in this trend. Moreover, in the last 2 years (i.e. 2019 and 2020), only two new primary studies were selected. Even though in round 4[27] we have not applied the six phases described in Section 4, we searched for studies in the main proceedings and journals referring to the area investigated in this work.

5.2. Challenges for adaptive system testing or context-aware system testing

From the 83 primary studies, 42 were selected with the application of criterion (ii). From these, 21 of them came from round 3. We reinforce the fact that we kept in our final set of studies only those that are peer reviewed, and this holds for subsumed studies as well. As such,we reanalysed the 25 primary studies identified in rounds 1 and 2, and this number decreased to 21 studies present in our final set.

The challenges listed in Table 7 are herein called *specific challenges*, because they are described by the original authors in particular research contexts. In other words, these challenges were mentioned or described in specific contexts, methodologies, testing approaches and so forth. Furthermore, in our prior paper [1], likewise in this article, we found out the goal of the selected primary studies was not consolidating a taxonomy of challenges; instead, in most cases, the authors of the primary studies employed either different terms for describing the same challenge or the same term for different challenges. Given that the challenges are scattered on the literature, to have a general view involving which testing challenges could be faced for these types of systems, characterizing testing challenges was one of the activities in this SLR, and we ended up with a list of 35 testing challenges for AS or CAS (Table 7).

Refinements performed in this article, in comparison with our prior paper [1], regard the inclusion of new specific challenges (SCs), the updating of references linked to some SCs and the

[27]The time frame considered for round 4 was June 2019 to June 2020.

Table 7. Specific challenges for AS and CAS testing.

SC	Description	AS	CAS
1	The impossibility to guarantee the correctness of a changing system with unpredictable and growing number of contexts and configurations.	[60] **[59]** **[58]** [61]	[63] [62] **[64]**
2	The issue of limiting (or not) the ability to adapt during the testing activity.	[10]	
3	The issue of when it is possible to stop the testing.	[87]	
4	The dynamicity of ASs, which have no clear boundary, so that the use of the configuration variants and context variants is not predictable.	[110] [72] [61]	**[19]**
5	The difficulty of defining test oracles.		**[19]**
6	The difficulty to detect and avoid possible resulting incorrect contexts and configurations of the system at runtime.	**[86] [104]** [60]	[63]
7	The difficulty of testing the triggering of adaptations with respect to the executor component at runtime.	[10]	
8	The impossibility to prove the correctness of a component wiring at runtime.	[60]	
9	The issue related to test a system that has synchronization between components and ensuring harmony between the closed control loops.	[103] **[43] [122]** [20] [80]	[29] [62]
10	The issue of testing a system built upon a layered architecture that encapsulates management and usage of context.	**[122] [109]**	
11	The issue of testing a system that let the users perform customization in how the system should adapt itself according to changes of contexts.	**[86]**	[107] [29]
12	The issue of testing a system whose contexts change all the time.		[88]
13	The need to handle the state space, which is developing in an evolutionary fashion at runtime.	**[43] [137]**	[102] **[123]**
14	The issue of identifying reliably important adaptive changes within the system and its execution environment.	[10]	[19]
15	The issue of determining the properties of the systems that should be observed.	[10]	
16	The challenge of validating design models that contain uncertain or learning characteristics.	[11] **[44] [108]**	
17	The possible changes in context that can affect the application behaviour at any time during the execution.		[107] [69]
18	The issue of testing a system that contain some kind of learning and reasoning capabilities.		[63]
19	The difficulty of testing dynamically systems whose structure and behaviour may change during its execution.	[103] **[96]**	[70]
20	The difficulty to apply data flow tests due to context-aware faults, environmental interplay and context-aware control flow.	**[104]**	[24]
21	The issue of testing how a system avoids inconsistent program states caused by 'noisy contexts'.	**[109]**	[71] **[19]**
22	**The issue of tracing the complete history of adaptations starting at a known initial state.**	**[122] [134]**	
23	The difficulty of promoting non-functional, adaptable properties into testable, first-class entities of the system.	[10]	
24	The issue of accurately accessing the impact on test cases caused by system adaptations.	[10]	
25	The difficulty to define thresholds or acceptance rates for the test data.	[10]	[63]
26	The issue of determining the frequency on which monitoring data should be gathered.	[10]	
27	The issue of determining what sensors, or sensor value aggregations, can measure desired properties.	**[90]** [10]	**[127] [105]**
28	The difficulty to build test systems that capture the size and complexity of realistic systems and workloads.	[20] **[134]**	**[127]** [63]

(*Continues*)

Table 7. (Continued)

SC	Description	AS	CAS
29	The need to reduce the number of tests that are automatically generated.	[78]	
30	The difficulty to automatically generate test cases for a changing environment.	**[104]**	**[105] [50]**
31	The issues for defining formal models for testing taking into account system properties related to adaptive behaviour.	[89]	
32	The difficulty of using mechanisms during the testing to express and formalize context-aware behaviour.		[63] [98]
33	**The high risk of an faulty autonomously acting system to damage the environment or itself during testing.**	**[90]**	
34	**The difficulty to define validation approaches that should be generic for any adaptation process of adaptive system.**	**[86] [122]**	
35	**The issue of the practicability to keep minimum the number of test executions in the context of mutation testing.**	**[135]**	

Notes. Items identified in the third and fourth rounds appear in bold and larger font size. [Correction added on 10 May 2021, after first online publication: some entries in Table 7 were mistakenly captured as regular font and has been corrected.]

classification of studies between AS or CAS. The new SCs and new studies are highlighted in bold larger font size in Table 7, likewise the references to studies selected with the search updates. Note that even though some SCs are not highlighted in bold (e.g. SC-5, SC-10 and SC-30), information related to them was also updated, particularly considering the study subsumption relationship, as well as the removal of non-peer-reviewed studies. For example, in this article, SC-30 (*The difficulty to automatically generate test cases for a changing environment*) is linked to the studies of Ma *et al*. [104], Yu *et al*. [50] and Mehmood *et al*. [105]; in our prior paper [1], SC-30[28] was linked to the study of Yu and Gao [49], which was subsumed by the study of Yu *et al*. [50], as shown in Table 6.

The classification of SCs between AS or CAS is the same classification shown in the column 'System' of Tables 5 and 6. This means that some challenges are more related to AS and other challenges are more related to CAS. Once again, we emphasize that *this classification relies on the original authors' description of their studies*. Many studies clearly describe which type of system they address. For example, Vieira *et al*. [64] stated that 'This contribution outlines challenges of testing context-aware mobile applications relating to their context …'. Another example is the study of Welsh and Sawyer [61], which contains excerpts such as (1) 'this paper focuses on a class of system that uses runtime models to drive runtime adaptations in changing environmental conditions' and (2) 'the compositional adaptation is achieved by allowing structural elements of the system to be combined and recombined at runtime'. Both descriptions (1) and (2), which involve runtime adaptations and recombining structural elements, led us to classify that study as AS related. The same rationale was applied to all studies in this SLR.

It is very important to note that by classifying a study as AS or CAS related, we do not mean such a study does not address any issue of the other type of system, specially because both types of systems share common concepts and properties, as described in the previous sections of this paper. In other words, a challenge that is classified as CAS related may also be presented in an AS, so it may be handled with a testing approach for ASs.

We also highlight that some authors use interchangeably both terms 'adaptive system' and 'context aware', whereas others make no distinction between both terms. As an example, Qin *et al*. [123] used the term 'self-adaptive system' while referring to CAS-related studies [8,29] as well as to an AS-related one [106] to conduct their work. Moreover, Qin *et al*. [123] used a running example that is commonly used in CAS-related studies; specifically, the authors used PhoneAdapter [138], which comes to be a system that was originally proposed by the community of CAS applications for exemplifying the definition of adaptation rules that depend on the system context [29].

As a last note, we emphasize that SCs are usually described using different terminology, within different contexts and in a varied level of details. Thus, the analysis not only relied on the short

[28]Cf. ID SC-32 in study [1].

descriptions of each challenge but also on the understanding of the primary studies after the full reading and data extraction. Possible relationships between testing approaches and challenges are discussed in Section 6.3.

5.3. Why considering adaptive systems and context-aware systems in this research?

As we described in Section 2, ASs and CASs may share some characteristics. Salehie and Tahvildari [4] mentioned that the term *context awareness* from CASs can be classified into the hierarchical levels of ASs properties. In this work, we took into account the definition of Salehie and Tahvildari [4], and we demonstrate that both terms have been used by some authors from the AS testing literature, likewise by some authors from the CAS testing literature. Given the set of studies listed in Tables 5 and 6, Figure 5 provides an overview on how studies have addressed ASs and CASs either exclusively or interchangeably. In the figure, studies classified as CAS are coloured in white (e.g. studies [66], [65] and [62]), while those classified as ASs are coloured in gray (e.g. studies [106], [10] and [20]). Note that there are also studies coloured in green so that such studies cited studies from both categories (ASs and CASs). An example of this last case is the study by Püschel *et al*. [44], which cited the studies by Kephart and Chess [20] and Nehring and Liggesmeyer [96] (i.e. AS-related studies), and the study by Wang *et al*. [69] (i.e. CAS-related study).

As we can note, even though there are two noticeable different groups of studies (i.e. the top and bottom areas of the figure), and hence evidence of two well-established research communities, there are some studies – that is, the ones coloured in green – that are correlated with both types of systems. Ultimately, this led us to decide to consider both types of systems in this research.

Based on the information shown in Figure 5, we are also able to identify that some studies were more cited than others. For example, the study by Kephart and Chess [20] (i.e. an AS-related study displayed in the bottom area of the figure) was cited by 18 studies, and the study by Wang *et al*. [69] (i.e. a CAS-related study displayed in top area) was cited by 20 studies In addition, there are studies that were neither cited by nor cited other studies (e.g. Lindvall *et al*. [126], Wotawa [125] and Siqueira *et al*. [117]).

6. ANALYSIS AND DISCUSSION

This section describes studies related to testing approaches and used techniques for ASs or CASs (Section 6.1). Moreover, this section analyses the challenges that are faced by ASs or CASs testing (Section 6.2). Finally, it presents discussions involving the findings and trends (Section 6.3).

6.1. Overview of studies that proposed and/or applied testing approaches for adaptive systems or context-aware systems

The 74 studies presented in this section were selected through criterion (i) (see Section 4 for more details). The studies are grouped in Sections 6.1.1 to 6.1.6 based on their underlying testing techniques. In Table 8, we list the studies involving such techniques (column *Technique*). For each technique, the table also groups the studies in categories. For example, for studies that explore functional testing, we established five categories, namely: *test case generation*, *definition of test sets for built-in and runtime*, *generation of duplicated components*, *test case prioritization* and *usage of metamorphic testing*. Note that a study can be in more than one category (e.g. studies by Merdes *et al*. [68], Niebuhr and Rausch [60] and Niebuhr *et al*. [72]). The process of grouping was similar to the characterization of challenges for AS or CAS testing (i.e. thematic analysis), which was described in Section 4. As an example, in Table 8, the second and third categories for functional testing were used to classify the study by Niebuhr and Rausch [60], in which the authors addressed the *definition of test sets for built-in and runtime* and the *generation of duplicated components*. The same procedure was performed for the other studies listed in Table 8. In the next subsections, we provide more information about each study. Note that each study description points to specific testing techniques and specific categories from Table 8. For example, the group of studies that explore the functional technique with focus on test case prioritization is labelled with *1-d*.

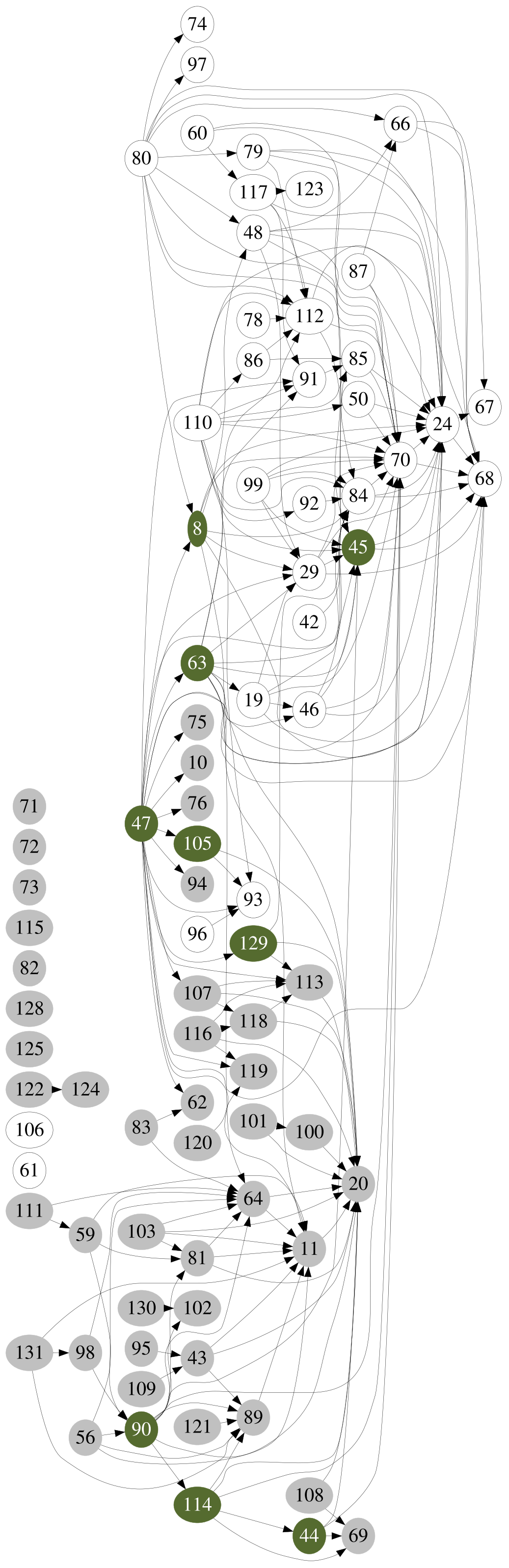

Figure 5. Citation graph between the studies.

Table 8. Classification of studies per testing technique.

Technique		Category	#	Studies
1 – Functional				
	a	Test case generation	6	[128], [105], [68]*, [124], [117], [118]
	b	Definition of test sets for built-in and runtime	4	[111], [68]*, [60]*, [72]*
	c	Generation of duplicated components	2	[60]*, [72]*
	d	Test case prioritization	2	[92], [114]
	e	Usage of metamorphic testing	1	[62]
2 – Structural				
	a	Context analysis and generation	2	[115], [76]
	b	Test case generation	4	[24], [71], [69], [75]
3 – Model-based				
	a	Definition of test sets for built-in and runtime	11	[120]*, [121]*, [56]*, [103], [131]*, [102], [112], [94]*, [61]*, [19]
	b	Test case generation	11	[56]*, [120]*, [131]*, [107], [94]*, [11], [129], [63], [29], [45]*, [87]
	c	Usage of formal specification and fault model	7	[56]*, [104], [123], [88], [134], [45]*, [98]
	d	Analysis of application tracing	2	[120]*, [121]*
	e	Usage of model similarities	1	[89]
	f	Usage of metamorphic testing	1	[126]
	g	Test minimization	1	[61]*
4 – Fault-based				
	a	Usage of mutation analysis and fault seeding	5	[42], [110], [132], [130], [48]*
	b	Definition of test sets for built-in and runtime	1	[80]
	c	Analysis of context diversity	1	[48]*
	d	Usage of formal specification and fault model	1	[109]
	e	Test minimization	1	[136]
5 – Hybrid				
	a	Usage of formal specification and fault model	5	[65], [137], [133], [44], [47]*
	b	Test case generation	5	[100]*, [47]*, [125], [91], [50]*
	c	Definition of test sets for built-in and runtime	3	[100]*, [70], [108]
	d	Analysis of context diversity	2	[47]*, [8]*
	e	Usage of data flow-based testing	2	[47]*, [50]*
	f	Test selection	2	[113], [8]*
	g	Usage of context emulators	1	[127]
6 – Other				
	a	Definition of test sets for built-in and runtime	8	[66]*, [79], [10], [106]*, [96]*, [85], [97], [78]*
	b	Test case generation	5	[66]*, [106]*, [119]*, [78]*
	c	Usage of formal specification and fault model	2	[116], [96]*
	d	Analysis of context diversity	1	[119]*
	e	Usage of metamorphic testing	1	[66]*

Notes. Studies in category overlapping are marked as '[*i*]*', where *i* is the reference number (e.g. '[68]*'.)

6.1.1. *Studies that either proposed or applied functional testing approaches.*

Functional testing considers the system under test (SUT) as a *black box* of which only input and output (without knowledge of the inside) are known. Thus, it relies on the specification of the software to derive the test cases, disregarding aspects associated with the application code [139]. Some studies retrieved in this SLR (e.g. Merdes *et al*. and Sen *et al*. [68,117]) deal with specification and component structures to apply *black box* testing in software units. Those studies are described as follows:

- **Test case generation (1-a):** Song *et al*. [118] introduced a method for systematically generating various execution contexts from permissions of mobile devices. Sen *et al*. [117] proposed an approach that explores combinatorial testing. Their approach comprises two testing criteria based on interactions between components: coverage T-wise and coverage R-wise, in which *T* and *R* concern, respectively, the number of variables and the number of reconfigurations.

  Rodrigues *et al*. [124] focused on generating test cases by taking into account variations of context. The process is composed of identifying context variables, identifying thresholds and generating test suites. In the same context, Matalonga and Travassos [128] proposed an approach based on an idea that context should vary freely during test execution as it does in the production environments. Finally, Mehmood *et al*. [105] not only addressed a difficulty involving the generation of test sets for testing contexts using sensors data (i.e. SC-27) but also proposed a process to generate test sets.

- **Test case generation (1-a) and definition of test sets for built-in and runtime (1-b):** the approach of Merdes *et al*. [68] is based on a functional technique that uses associations between specification of components to generate test data. They also focused on combining runtime testing with context-aware features.

  Fredericks and Cheng [111] proposed a framework for testing ASs at runtime, supporting the management and the adaptation of test sets. It comprises two tasks: adaptive test plan, which is responsible for defining test sets at design time, and testing cycle, which analyses the execution of the software, possibly finding new configurations.
- **Definition of test sets for built-in and runtime (1-b) and generation of duplicated components (1-c):** the approach of Niebuhr and Rausch [60] addresses secure components, which are typical elements in fault-tolerant systems. The same research group also proposed an approach to monitor messages between components [72], including a strategy for running tests that ensure the component that is about to be bound to an AS behaves as expected before the binding occurs. Regarding built-in tests with respect to component duplication, Neibuhr *et al*. [60,72] aimed to duplicate components to test them before their execution.
- **Test case prioritization (1-d):** regarding prediction and test case prioritization, Garvin *et al*. [92] proposed the management of failure history to predict possible failures at runtime. Lim *et al*. [114], on the other hand, focused on prioritizing test cases based on their importance according to the adaptation context, to properly define which tests should be executed.
- **Usage of metamorphic testing (1-e):** Tse *et al*. [62] proposed the definition of metamorphic relations to establish relationships among test cases to define domain-specific restrictions. More specifically, the main characteristic of this approach is that if a given test case *t* does not pass, all interrelated test cases (i.e. test cases in a metamorphic relation that includes *t*) are all seen as failing test cases and should be analysed.

6.1.2. Studies that either proposed or applied structural testing approaches. Structural testing, also known as *white box* (as opposed to *black box*), is based on the knowledge of the internal structure (e.g. design structure or source code structure) of the SUT [139]. As examples, some studies [24,71] addressed structural criteria (e.g. associations between internal nodes of data flows) to apply the testing by taking into account context variations. These and other studies that explored structural testing for ASs and CASs are next described.

- **Context analysis and generation (2-a):** Taranu and Tiemann [76] proposed an approach to support the generation of contexts and how to use these contexts to develop a test method (using internal and external interfaces). Majchrzak and Matthias [115] focused on the identification of code blocks that may cause context changes; this is performed through the classification of different types of context.
- **Test case generation (2-b):** for automatic test case generation, Lu *et al*. [24,71] explored data flow-based criteria based on situations and contexts of the SUT; they use associations between different flows of data to generate test cases. The same authors [71] proposed another approach to evaluate inconsistencies, but by focusing on evaluating the exchange of data between services, components and middleware of a system. Ye *et al*. [75] used customized graphs for representing scenarios of a system by analysing its source code. Their approach involves middleware modelling and test case generation. Wang *et al*. [69] focused on the automatic generation of test cases, using possible variations of context in the system supporting the generation of test cases.

6.1.3. Studies that either proposed or applied model-based testing approaches. Model-based testing leads to test cases being systematically generated from formal test models, thus decreasing the ambiguity and supporting the testing automation [140]. As examples, some studies [29,45] use finite-state machines to apply testing strategies. Once a formal model is created to represent the system specification, testing criteria can be derived using that specific model.

- **Definition of test sets for built-in and runtime (3-a):** Eberhardinger [103] and Eberhardinger *et al*. [102] addressed the use of oracles for minimizing the number of test cases to be generated. The first study [103] focused on defining a model of constraints to reduce the testing scope, while the second study [102] focused on minimizing the test set. Hansel *et al*. [112], also in the runtime perspective, investigated testing approaches for control loops. The considered approaches use architectural models at runtime. Xu *et al*. [19], on the other hand, used a model-based approach to predict failures at runtime.
- **Test case generation (3-b):** Griebe and Gruhn [107] explored model transformations to generate test cases. By using design time models, they addressed test set generation and evaluated the efficiency of test execution. Their approach comprises a four-tier process that uses (i) context information; (ii) petri net representation; (iii) technology-independent testing model; and (iv) technology-specific test case generation. Mehmood *et al*. [129] proposed a model-based testing approach to model transformations that uses activity diagrams and Petri nets. In general, they use an extended UML activity diagram by taking into account different alternatives when an activity node A is transiting to an activity node B. Their approach suggests the need of analysing the context data during these transitions. These models and transitions between nodes are used to generate test cases. Micskei *et al*. [63] defined a language to represent contexts and situations that are used to generate test cases and applying coverage metrics. Sama *et al*. [29,45] explored FSMs for simulating the behaviour of systems, supporting the generation of test cases based on transitions and sequences between states. Wotawa [87] proposed a model to support the state diagnosis of a system to generate test cases. Also to generate tests, Jaw *et al*. [11] captured contexts based on a model that represents layers of the system and test oracles.
- **Definition of test sets for built-in and runtime (3-a) and test case generation (3-b):** Eberhardinger *et al*. [131] proposed an approach based on built-in test and runtime models to generate test data using reflection to retrieve runtime data. In addition, a similar approach was proposed by Horányi *et al*. [94]. However, the latter proposed a model to capture the context and to generate test data representing stressful contexts.
- **Usage of formal specification and fault model (3-c):** Ma *et al*. [104] addressed testing executable models by defining how to simulate an AS that deals with failures and uncertain scenarios at runtime. Thus, their approach is based on a controlled environment that lets the user include modifications in the SUT. Qin *et al*. [123] proposed techniques for mocking the system environment. Their approach is based on the observable behaviour of objects and on an under-specified formal descriptions. With respect to model definition, Püschel *et al*. [88] defined a failure model for applying model-based testing. Reichstaller and Knapp [134] proposed a framework for prediction-based test design. It is inspired in the *Direct Future Prediction* technique and involves a mixture of model-based and model-free testing (the former requires an interpretable model of the environment; in the latter, the software can continuously interact with an executable environment interface). Xu *et al*. [98] evolved the work of Sama *et al*. [29] and proposed a model for representing system states that were not specified in advance, using context variables.
- **Definition of test sets for built-in and runtime (3-a), test case generation (3-b) and usage of formal specification and fault model (3-c):** in the context of dealing with FSMs, Arcaini *et al*. [56] proposed a model-based testing approach to be used at runtime that is composed of the usage of a language to formally specify the structure of an AS and its behaviour (using FSMs). The transitions of these FSMs, which are based on MAPE-K, are used to generated test sets. After that, the test sets are executed to validate the system correctness.
- **Definition of test sets for built-in and runtime (3-a), test case generation (3-b) and analysis of application tracing (3-d):** Al-Refai *et al*. [120] proposed a model-based approach for validation at runtime in which models are generated from the original tests and a tracing of

the adaptation of the system is performed, supporting the generation of test cases. Another study from the same research group [121] proposed a runtime approach to represent test cases and changes of a system by using activity diagrams; the approach also counts on the support of tracing analysis of the system execution.

- **Definition of test sets for built-in and runtime (3-a) and test minimization (3-g):**Welsh and Sawyer [61] identified specific system situations at runtime and argued that even partial testing can increase a degree of assurance, by applying test minimization, and hence can offer a degree of freedom in terms of system's autonomy.
- **Test case generation (3-b) and usage of formal specification and fault model (3-c):**the approach of Sama *et al*. [45] relies on formal models of finite-state machines (FSM) for fault detection using states and transitions involved during the adaptation of the system. These FSMs are used to support test case generation.
- **Usage of model similarities (3-e):** Weyns [89] focused on detecting similarities in a behaviour model when an initial system and an evolved one are compared with each other.
- **Usage of metamorphic testing (3-f):** Lindvall *et al*. [126] proposed a model-based approach using metamorphic principles. The approach is based on environment simulation of a drone, letting the tester add variations such as obstacles and landing pads.

6.1.4. Studies that either proposed or applied fault-based testing approaches. In fault-based testing, the test requirements are derived from the most frequent errors made during the software development process. As examples, some studies selected in this SLR [48,132] relied on the mutation testing criterion [141] or on its underlying concepts. In short, mutation testing consists of generating various slightly modified versions of a program (the *mutants*) and producing test sets that are able to show the behavioural differences between the program and its mutants.

- **Usage of mutation analysis and fault seeding (4-a):** Bartel *et al*. [42] used adaptation policies to generate mutation operators. Cámara *et al*. [110], on the other hand, proposed a set of possible faults related to components of a control loop (e.g. MAPE-K) and explored this idea in the context of components modified by the set of possible faults.

  In the same context, Usaola *et al*. [130] defined an environment to simulate Android-based system executions and proposed a set of mutation operators to reproduce common context-aware errors reported by mobile applications developers.

  Fredericks [132] conducted an empirical study investigating mutation operators and evolutionary algorithms implemented within a runtime testing framework.
- **Usage of mutation analysis and fault seeding (4-a) and analysis of context diversity (4-c):** Wang *et al*. [48] applied the mutation analysis and context diversity to test systems that have properties of adaptation. The authors presented the relationship between the analysis of context diversity with respect to test cases and mutation operators.
- **Definition of test sets for built-in and runtime (4-b):** King *et al*. [80] defined two strategies of validation for self-testing to be applied at runtime. First, RV (replication with validation) tests adaptive changes using copies of the managed resources. Then, SAV (safe adaptation with validation) tests adaptive changes in place, directly on the managed resources of the system.
- **Usage of formal specification and fault model (4-d):** Püschel *et al*. [109] focused on applying a formal fault model, error and failure by using MAPE-K, deriving 10 distinct failure scenarios that may occur in the process of adaptation.
- **Test minimization (4-e):** Reichstaller *et al*. [136] proposed a fault-based approach on test suite reduction by taking into account effects of communication faults during the reconfiguration process of a system.

6.1.5. Studies that either proposed or applied hybrid techniques. We classified studies as hybrid technique when such studies explore two or more types of the other techniques we described in the previous sections (i.e. functional, structural, model based and fault based).

- **Usage of formal specification and fault model (5-a):** Flores *et al*. [65] focused on using the functional and model-based techniques with the definition of formal abstract models for representing the context of a SUT.

  Mirza and Khan [133] proposed an approach for behaviour modelling of contextual data of a system by extending the *UML* activity diagram. Then, by applying functional and model-based testing, the authors used a formal test model for *context-coupled* analysis and execution of test.

  Ma *et al*. [137] proposed a technique named *fragility*, based on system behaviour, which includes formal model-based and fault-based techniques. Each behaviour comprises a set of test executions, and the authors investigated which sequence of test executions could be related to uncertainty.

  Püschel *et al*. [44] investigated black box (functional) and model-based testing, exploring formal models based on a UML class diagram, statecharts and equivalence classes.
- **Test case generation (5-b):** Wotawa [125] used a test model for applying combinatorial testing and used injection mechanisms to introduce constraints to minimize the test set. In general, the author showed how fault injection can be seen as a valuable tool for testing ASs.

  Amalfitano *et al*. [91] explored sets of event-patterns. Overall, their approach considers contexts of the system and its context-related events. The definition of these event-patterns is used to explore the system's behaviours and to generate test cases according to them.
- **Definition of test sets for built-in and runtime (5-c):** King *et al*. [70] investigated the development of built-in tests for simulating versions of systems with/without faults. For this, they explored functional, structural and fault-based techniques. Part of the results was subsequently used in a comparative study that applied fault-based testing [80].

  Püschel *et al*. [108] focused on simulating and capturing unspecified parts of the system or the environment at runtime (referred to as 'in-the-loop'). The authors explored the model-based and functional techniques.
- **Usage of formal specification and fault model (5-a), test case generation (5-b), analysis of context diversity (5-d) and usage of data flow-based testing (5-e):** Siqueira *et al*. [47] proposed a testing strategy based on the combination of different approaches; the approaches are selected based on addressed testing challenges, testing activities and characteristics of systems. Their study involved functional, structural and model-based testing and used formal models to generate test sets. The generation of test sets has used analysis of context diversity and data flow testing criteria.
- **Test selection (5-f):** Lahami *et al*. [113] focused on identifying reusable tests and new test cases for covering the SUT. The approach is based on specifying behavioural model using extended FSMs. The new and irrelevant test cases are identified with an analysis of similarities between the behaviour models. The authors explored model-based and functional techniques.
- **Test case generation (5-b) and definition of test sets for built-in and runtime (5-c):**Akour *et al*. [100] explored functional and structural techniques that use a runtime approach that synchronizes the generation of test data. Their approach is based on the regression testing being applied at runtime. Thus, they applied a technique of change propagation, analysing the modifications that impact on the test cases.
- **Test case generation (5-b) and usage of data flow-based testing (5-e):** To automatically generate test cases, Yu *et al*. [50] explored model-based and structural techniques (using graphs to define the test model). More specifically, their approach uses structural criteria to support to analysis of associations between data flow and control flow of programs.
- **Analysis of context diversity (5-d) and test selection (5-f):** Wang *et al*. [8] focused on optimizing the test case selection based on source-code coverage with information of the context variables using structural and fault-based techniques. Note that their approach is based on the analysis of context diversity.
- **Usage of context emulators (5-g):** Luo *et al*. [127] proposed an approach to emulate contextual data on either physical devices or emulators, applying functional and structural testing. Their approach, also available as a tool, lets the developer manage data from the execution

of a system. These data support the analysis of functional/non-functional properties that the system can demand.

6.1.6. Studies related to other approaches. Studies that could not be assigned a specific technique were classified as *other approaches*. In contrast, we were able to classify them in the categories listed in Table 8, group '6 – other'. Note that for this technique classification, every category has overlapping with at least one other category.

- **Definition of test sets for built-in and runtime (6-a):** Fredericks *et al*. [10] introduced a testing framework named MAPE-T. It is an analogy to MAPE-K model because it includes the MAPE elements plus a testing (T) component. The approach focuses on built-in test and runtime. Also regarding testing processes, Da Costa *et al*. [79] proposed a framework for applying built-in tests based on MAPE-K (referred to as *self-testing*). The framework includes a process for (i) defining tests at design level, (ii) defining tests that will be applied and (iii) using a test planning based on MAPE-K.

  Silva and De Lemos [85] aimed to define test cases for the integration level at runtime, based on the association with each component, identifying integration order, defining stubs and detecting new errors at runtime. They also proposed a framework for generating testing processes based on dynamic adaptation [97].
- **Usage of formal specification and fault model (6-c):** Markov and Fröhlich [116] coined the term 'test ecosystem' to refer to an approach that combines roles, formal methods and tools of different sources and suppliers. In short, the authors presented a platform that provides an end-to-end testing environment that the developer can instantiate and customize with respect to business goals, intended uses, interfaces and so forth.
- **Definition of test sets for built-in and runtime (6-a) and test case generation (6-b):** Fredericks *et al*. [106] proposed a runtime approach that monitors the environment for identifying changes, so that the test cases are generated and adapted. Vassev *et al*. [78] proposed a runtime approach for generating test cases driven by change impact analysis. They investigated the generation of test cases driven by a tool responsible for extracting policies, analysing impact of changes, generating test sets and minimizing the test sets. The approach aims to identify different coverage paths.
- **Definition of test sets for built-in and runtime (6-a), test case generation (6-b) and usage of metamorphic testing (6-e):** Chan *et al*. [66] proposed runtime approach using checkpoints as the starting and ending points of a test case, and then the validation is performed based on multiple test cases that are generated by analysing the checkpoints, even when errors occur during the test execution. Their approach also uses characteristics from the metamorphic testing.
- **Definition of test sets for built-in and runtime (6-a) and usage of formal specification and fault model (6-c):** Nehring and Liggesmeyer [96] proposed an approach based on a formal test process model that takes into account structural changes of a system. Data of component interactions are captured during the reconfiguration of the system. Thus, the approach that is runtime based considers the order of reconfiguration, state transfer and transaction handling of the system reconfiguration.
- **Test case generation (6-b) and analysis of context diversity (6-d):** Tonjes *et al*. [119] implemented a tool for generating test inputs. In their tool, annotations are used to create specific test data with a method for decreasing the number of test cases based on analysis of context diversity.

Table 9 summarizes the recurring concerns regarding testing approaches. Note that despite the diversity of options to apply testing to ASs or CASs, they overlap in some characteristics. As an example, *RQ1-(iii) generation of test cases* and *RQ1-(v) built-in test* are inserted in a large number of approaches, whereas *RQ1-(iv) context diversity* and *RQ1-(vi) MAPE-K* are less addressed.

Regarding RQ1 (*Which are the testing approaches that are proposed for ASs or CASs*), we identified 74 studies that applied/addressed testing for ASs or CASs. We noted different testing techniques (e.g. functional, structural and fault based) are employed in these types of systems, and

Table 9. Most recurring concerns for research on AS and CAS testing.

ID	Description	Studies
RQ1-(i)	Simulation	[29,45,94,108,126,127,130,135]
RQ1-(ii)	Data capturing	[8,11,65,94,96,108,123,132]
RQ1-(iii)	Generation of test cases	[11,24,29,45,63,68,69,71,75,87,102,105,107,118,120,124] [43,50,94,100,104,123,126,129,131] [47,56,66,78,91,106,113,117,119,125,128,133]
RQ1-(iv)	Context diversity	[8,47,77,98,119]
RQ1-(v)	Built-in test	[10,42,56,60,61,68,70,72,79,80,85,97,100,111,131,132]
RQ1-(vi)	MAPE-K	[10,56,79,88,109,110,112]

researchers have overlapping concerns when applying them, such as (i) simulation as a frequently adopted strategy, in which the environment is a compound of a set of variations in the context variables; (ii) data capturing, which is performed in the interaction between components and the environment at runtime, thus supporting the exploration of the possible test space; (iii) generation of test cases, especially when dealing with adapting test sets; (vi) context diversity, which underlies the analysis of variations in context variables and supports test set minimization; (v) built-in tests, which are used especially in runtime tests; and (vi) MAPE-K, because AS or CAS communities use it as a reference model.

6.2. *Generic challenges for adaptive system or context-aware system testing*

Figure 6 presents a list of 13 generic challenges (with a prefix 'C-') for AS or CAS testing. The generic challenges were devised from the analysis and grouping of the specific challenges (SCs) presented in Table 7.

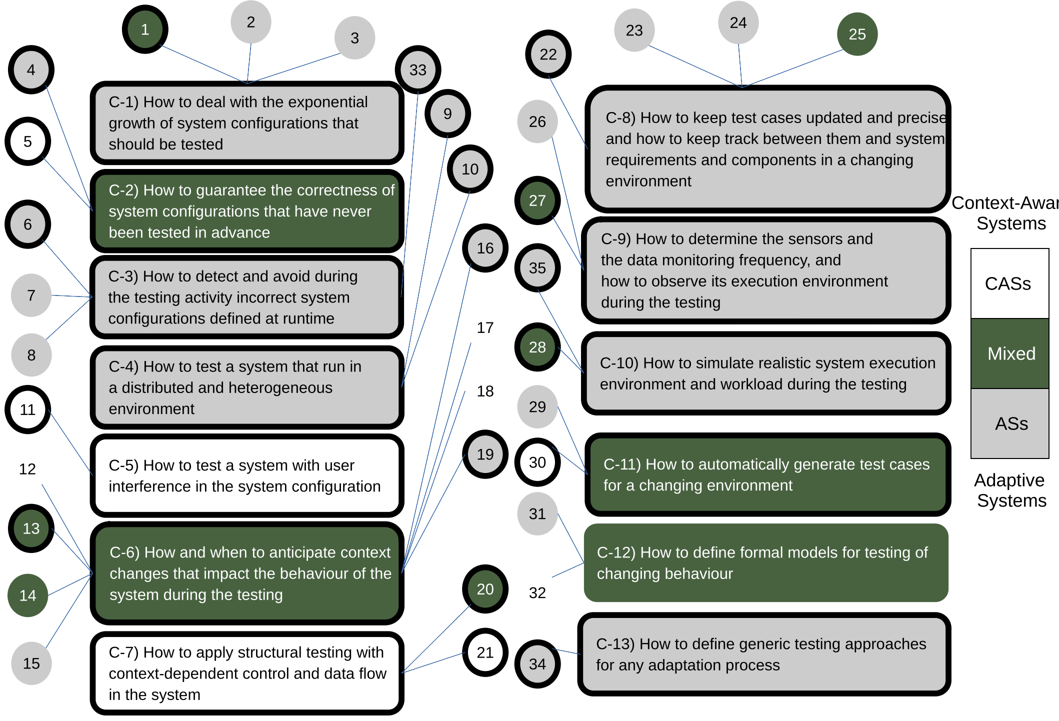
1
2
3
4
5
6
7
8
11
12
13
14
15
33
9
10
16
17
18
19
20
21
22
23
24
25
26
27
35
28
29
30
31
32
34
C-1) How to deal with the exponential growth of system configurations that should be tested
C-2) How to guarantee the correctness of system configurations that have never been tested in advance
C-3) How to detect and avoid during the testing activity incorrect system configurations defined at runtime
C-4) How to test a system that run in a distributed and heterogeneous environment
C-5) How to test a system with user interference in the system configuration
C-6) How and when to anticipate context changes that impact the behaviour of the system during the testing
C-7) How to apply structural testing with context-dependent control and data flow in the system
C-8) How to keep test cases updated and precise and how to keep track between them and system requirements and components in a changing environment
C-9) How to determine the sensors and the data monitoring frequency, and how to observe its execution environment during the testing
C-10) How to simulate realistic system execution environment and workload during the testing
C-11) How to automatically generate test cases for a changing environment
C-12) How to define formal models for testing of changing behaviour
C-13) How to define generic testing approaches for any adaptation process
Context-Aware Systems
CASs
Mixed
ASs
Adaptive Systems

Figure 6. Generic and specific challenges (for clarity, we omitted the 'SC-' prefix from the specific challenges that are represented by the circles in this figure).

We separated the studies that are more related to CASs from the studies more related to ASs. In Figure 6, the rectangles (i.e. generic challenges) and the circles (i.e. specific challenges) depicted in white background are more related to CASs than to ASs. The elements in gray background are more related to ASs. The remaining items (i.e. the ones with green background) are related to both types of systems. The classifications of the specific and generic challenges are originally presented in this article. They were defined by taking into account the number of studies related to AS or CAS, as shown in Table 7. A specific or a generic challenge was classified as related to a specific type of system when more than two-thirds of the studies associated with the challenge were related to that type of system.

Regarding our dataset update (i.e. rounds 3 and 4), in Figure 6, we highlight in thick edges those specific/generic challenges that were updated (i.e. challenges that have new studies associated with them). Note that 16 specific challenges were not updated (those in thin edges), whereas 19 specific challenges had some update, being the latter related to 12 over the 13 generic challenges we characterized.

Once again, we emphasize that *our classification relied on the original authors' description of their studies*, as previously stated in Section 5.2. In what follows, we describe the generic challenges. With respect to the studies from rounds 3 and 4, that is, the new selected studies, we highlight them with the ⋆ tag.[29]

C-1 – How to deal with the exponential growth of system configurations that should be tested: Tse *et al*. [62], Welsh and Sawyer [61] and Micskei *et al*. [63] mentioned that it is impossible to guarantee the correct behaviour of a system that changes all the time, whose number of contexts and configurations can be unpredictable and growing (SC-1). In addition, Fredericks *et al*. [10] mentioned that limiting (or not) the system ability to adapt, in order to enhance its testability, is also a hard decision (SC-2). Both approaches addressed the same problem of settings in exponential growth, which is challenging for testing of ASs or CASs. Another challenge (SC-3), reported by Wotawa [87], regards the issue of when it is necessary to stop testing by using a model-based approach.

⋆ Truszkowski *et al*. [58], ⋆ Schumann and Visser [59] and ⋆ Vieira *et al*. [64] also addressed challenges related with C-1, specifically regarding SC-1.[30] Such studies mentioned the impossibility to guarantee the correct behaviour of a changing system, whose number of configurations may be unpredictable and growing.

It is clear that a major difficulty is the task of scoping a test suite in order to execute on a system due to its number of possibilities for configuration and/or structural changes.

C-2 – How to guarantee the correctness of system configurations that have never been tested in advance: Niebuhr and Rausch [60], Niebuhr *et al*. [72], Welsh and Sawyer [61] and Cámara *et al*. [110] mentioned the difficulty of testing a system that does not have a clear boundary, because of the dynamic configurations and contexts that this system can have (SC-4). In addition, ⋆ Xu *et al*. [19] also highlighted the difficulty of defining a testing oracle (SC-5) and discussed about the dynamicity of systems, which have no clear boundary, so that the use of the configuration variants is not predictable (SC-4).

Similarly to C-1, in this scenario, a major difficulty consists in defining test cases to cover unforeseen configurations. Note that this challenge is also faced when dealing with the test oracle problem, being a common problem addressed in the software testing area [142,143].

C-3 – How to detect and avoid during the testing activity incorrect system configurations defined at runtime: Niebuhr and Rausch [60], ⋆ Ma *et al*. [104] and Micskei *et al*. [63] mentioned that the dynamicity of systems, whose boundaries are not clear, leads the system to unpredictable configurations and contexts (SC-6). Niebuhr and Rausch [60] also mentioned a specific challenge of proving the correctness of runtime component wiring (SC-8). Both challenges (SC-6 and SC-8) relate to the issue of detecting and avoiding incorrect settings at runtime.

[29]Note that the terms *Context*, *Configuration* and *Environment* found in the studies and in our characterization of challenges followed the definitions described in Section 2.1.
[30]Table 4 presented the text snippets, using the thematic analysis, from the studies that converge to SC-1.

With respect to the system boundary that is not clear, ⋆ Bayha *et al*. [90] also mentioned the high risk of a fault to damage the environment or the system itself during the testing procedure (SC-33). In addition, Fredericks *et al*. [10] mentioned the difficulty of defining adaptable entities to become testable (SC-7). ⋆ Eze *et al*. [86] mentioned the need to detect and avoid possible incorrect system configurations at runtime (SC-6).

In this scenario, as well as in C-2, it is not straightforward to dynamically define test cases to avoid incorrect settings, so that they could guarantee continuous system operation.

C-4 – How to test a system that run in a distributed and heterogeneous environment: Kephart and Chess [20], Sama *et al*. [29] and King *et al*. [80] mentioned the difficulty of testing systems that run in heterogeneous and distributed environments (SC-9) (e.g. composed of different devices and different kinds of software) and the difficulty of anticipating environment changes when the analysis involves multiple domains or enterprises, for example, in a wide-scale systems built from multiple vendor components.

Tse *et al*. [62] highlighted the difficulty of dealing with *race conditions* in context tuples between the middleware layer and the application layer. These *race conditions* mean there are concurrent operations that conflict with one another. In other words, this is related with how the asynchronous behaviour (i.e. concurrent operations) of systems impacts on the design of test cases (SC-9). Also note that the authors mention the issue of race conditions by dealing with different layers (viz. middleware and application layers). Similar characteristics, in this case more specifically heterogeneous components and asynchronous behaviour of ASs, were issues also addressed by ⋆ Heck *et al*. [122]; the authors highlighted the difficulty in identifying the difference between software events and environment events. In addition, ⋆ Püschel *et al*. [109] mentioned the issue of dealing with multilayered architecture, by encapsulating the context management from the context usage into distinct layers to ease the development of applications (SC-10).

Eberhardinger [103] mentioned the need for synchronizing components to ensure harmony between the closed control loops (SC-9). The same research group ⋆ [43,103] mentioned the difficulty of testing a system that has asynchronous behaviour. Finally, ⋆ Püschel *et al*. [109] mentioned the issue of testing a system with a common layer architecture to encapsulate context management from the usage of contexts (SC-10).

In this scenario, for instance, testers should deal with system properties like dynamism, heterogeneity and synchronization in order to define test cases to validate such systems.

C-5 – How to test a system with user interference in the system configuration: Sama *et al*. [29] and Griebe and Gruhn [107] mentioned that in scenarios in which users are given the freedom to customize the system settings, one of the key issues concerns the design of test cases to simulate such user 'interference' in the system adaptation (SC-11). Both groups of researchers mentioned the difficulty of testing a system that is customized by users. ⋆ Eze *et al*. [86] discussed the issue of granting users the freedom to perform customized system reconfigurations and changes in the system's contexts (SC-11).

Similarly to C-1 and C-2, in this scenario, the difficulty regards anticipating system configurations that can be customized by users.

C-6 – How and when to anticipate context changes that impact the behaviour of the system during the testing: Wang *et al*. [69] argued that possible changes in context can affect the system behaviour at any time after the system is put in operation (SC-17). Likewise, Fredericks *et al*. [10] presented the need to identify important adaptive changes, within the system execution environment, in a reliable way (SC-14). In addition, Griebe and Gruhn [107] mentioned the issue of anticipating changes in the context parameters (SC-17). In all these specific challenges, the authors highlighted the need of anticipating context changes and their impact on the system behaviour.

Regarding uncertain environments or systems with learning characteristics, Jaw *et al*. [11] pointed out the difficulty of validating design models (SC-16), and Püschel *et al*. [88] mentioned the difficulty of testing a system whose contexts change all the time (SC-12). In this scenario, Micskei *et al*. [63] mentioned the issue of dealing with systems that contain some kind of learning and reasoning capabilities (SC-18). The same challenge is mentioned by King *et al*. [70], who highlighted the difficulty of testing a system whose structure and behaviour may change during its execution (SC-19).

Regarding the issue of dealing with state space that the system can have, Eberhardinger *et al.* [102] mentioned the difficulty of developing a system in an evolutionary manner, such that the evolution is not all planned at design time (SC-13). Püschel *et al.* [36], on the other hand, mentioned the need to handle the behavioural decision space (i.e. the context characterization) that impacts the system structure and running processes (SC-13). In addition, Fredericks *et al.* [10] mentioned the difficulty of selecting context variables that must be observed (SC-15).

⋆ Xu *et al.* [19] highlighted the issue of identifying reliably important adaptive changes within the system and its execution environment (SC-14). ⋆ Nehring and Liggesmeyer [96] and Eberhardinger [103] mentioned the difficulty of testing systems whose structure and behaviour may change during its execution (SC-19). ⋆ Püschel *et al.* [108] and ⋆ Püschel *et al.* [44] mentioned the difficulty of validating design models that contain uncertain or learning characteristics (SC-16). Last but not least, ⋆ Qin *et al.* [123], ⋆ Eberhardinger *et al.* [43] and ⋆ Ma *et al.* [137] mentioned the need to handle the state space that is developed in an evolutionary fashion at runtime (SC-13).

Based on this set of interrelated specific challenges, a major issue consists in building a test set that properly encompasses all relevant context variables with representative values.

C-7 – How to apply structural testing with context-dependent control and data flow in the system: ⋆ Xu *et al.* [19], ⋆ Püschel *et al.* [109], Lu *et al.* [24] and ⋆ Ma *et al.* [104] highlighted the difficulty of applying data flow testing criteria in these type of systems (SC-20) due to the interference of the environment and due to the context-aware nature of control flow and data flow-related faults (e.g. faults that are found out by applying structural testing criteria). Moreover, Lu *et al.* [71] and Püschel *et al.* [36] mentioned a specific challenge of avoiding inconsistent states inside of the system (SC-21). These inconsistent states are caused by ‘noise’ in context.

Regarding this set of overlapping specific challenges, the difficulty is related to the complexity of designing test models (e.g. control flow graphs with associated data flow information) and defining test sets to cover properties of those models (e.g. particular paths of the graphs).

C-8 – How to keep test cases updated and precise and how to keep track between them and system requirements and components in a changing environment: concerning the issue of keeping the traceability between requirements, test cases and components of these systems, Fredericks *et al.* [10] mentioned a specific challenge regarding the difficulty of defining the first-class entities that are non-functional, which typically involves adaptable properties, so that they could become testable entities (SC-23). They also highlighted the need to assess accurately the impact of system adaptation on test cases (SC-24). In the same context, ⋆ Reichstaller and Knapp [134] mentioned the issue of tracing the complete history of adaptations and their execution states (SC-22). In another study, ⋆ Heck *et al.* [122] mentioned the issue of dealing with ASs testing by taking into account the current state and previous test results. These challenges are related to the continuous updating (and impact of this updating) on the requirements, test cases and components. In addition, Fredericks *et al.* [10] also addressed the difficulty of keeping traceability links among requirements, components and test cases (SC-24), as well as defining thresholds or acceptance rates for the test data (SC-25).

In this scenario, it is difficult to keep traceability between requirements, test cases and components due to inherent, changing characteristics of these types of systems.

C-9 – How to determine the sensors and the data monitoring frequency, and how to observe its execution environment during the testing: Fredericks *et al.* [10], ⋆ Bayha *et al.* [90], ⋆ Luo *et al.* [127] and ⋆ Mehmood *et al.* [105] mentioned the challenge of determining what sensors, or aggregations of sensor values, can measure desired properties (SC-27). This issue is related to the definition of relevant input data. Additionally, Fredericks *et al.* [10] mentioned that it is also necessary to determine the frequency of data monitoring (i.e. sensor data) to be collected (SC-26).

In the scenario of preparing the environment and the set of test cases, for instance, it is difficult to determine which sensors must be enabled during the testing. In addition, it is difficult to determine the frequency at which data will be collected during the system execution.

C-10 – How to simulate realistic system execution environment and workload during the testing: Kephart and Chess [20], Micskei *et al.* [63], ⋆ Luo *et al.* [127], and ⋆ Reichstaller and Knapp [134] mentioned the difficulty of building system tests that capture the size and complexity of realistic systems and workloads (SC-28). In this context, ⋆ Reichstaller *et al*. [135] mentioned the issue of the practicability to keep minimum the number of test executions in the context of mutation testing (SC-35). This issue permeates a challenge to simulate, realistically, execution environments and workloads that real system executions can have. Particularly for these types of systems, realistic simulations and workloads are directly impacted by unpredictability and unclear system boundaries.

C-11 – How to automatically generate test cases for a changing environment: for ⋆ Yu *et al*. [50], ⋆ Mehmood *et al*. [105] and ⋆ Ma *et al*. [104], it is hard to automatically generate test cases for these types of systems (SC-30), for which changing (i.e. reconfigurable) structure and executing environment are inherent properties. In this context, Vassev *et al*. [78] addressed the issue of reducing the number of tests that are automatically generated (SC-29), specially due to the two aforementioned properties of these types of systems.

C-12 – How to define formal models for testing of changing behaviour: Weyns [89] addressed the challenge of defining formal models to validate by taking into account adaptive properties of the systems (SC-31). The authors mentioned this challenge in the context of using model checking in combination with testing techniques.

In the same context, Micskei *et al*. [63] and ⋆ Xu *et al*. [98] also highlighted the difficulty of using mechanisms to express and formalize context-aware behaviour (SC-32) to allow for verification tasks. Because model verification techniques produce verification sequences that we can use to build test cases for the final implementation, we can see the difficulty imposed by ASs or CASs to one technique as a challenge to the other as well.

C-13 – How to define generic testing approaches for any adaptation process: ⋆ Eze *et al*. [86] mentioned the difficulty of defining validation approaches that should be generic for any system adaptation process (SC-34). The same challenge is addressed by ⋆ Heck *et al*. [122] with respect to how an approach can be generalized to be applicable in the entire domain or, at least, to a certain subset of applications. This issue is related to a challenge of defining a testing approach that shall be generic for all subdomains of these types of systems. Even though a specific testing approach could be defined, the approach could be useless to test another system in a specific subdomain.

Additional note: Note that challenges C-1 to C-12 were documented in prior research [1,12], based on the analysis of 25 primary studies (Section 5.2 brings more details about it). In this article, after the analysis of additional 21 primary studies, we updated the list and extended it with a new challenge (C-13). Furthermore, the analysis was enhanced with the classification of challenges, what is reflected by the colour schema that represents a challenge that is related to either ASs, CASs or both. Last and not least, it is important to mention that the testing challenges – particularly the generic challenges – are not disjoint. It is possible to note that they, eventually, may be complementary between themselves (e.g. one can arise from another) or still be the opposite (e.g. one can be the negation of another). As an example, in C-1, we can deal with the lack of test oracles due to unpredictable inputs and outputs. In this scenario, challenges related to C-2 can emerge, regarding difficulties in guaranteeing the correctness of the system.

In Table 10, we provide a different perspective: we highlight that the challenges share some characteristics. As an example, *RQ2-(v) context monitoring* is present in all challenges, whereas

Table 10. Summary of challenges for AS and CAS testing.

		Generic challenges (GC)												
ID	Description	1	2	3	4	5	6	7	8	9	10	11	12	13
RQ2-(i)	Runtime decisions	x	x	x		x	x		x	x		x	x	x
RQ2-(ii)	Heterogeneous and distributed environment	x		x	x			x	x	x	x	x		
RQ2-(iii)	Interference of users	x		x		x	x	x	x		x	x	x	x
RQ2-(iv)	Updating and traceability	x		x	x				x	x				
RQ2-(v)	Context monitoring	x	x	x	x	x	x	x	x	x	x	x	x	x
RQ2-(vi)	Generation of test cases	x		x		x		x		x	x	x		

RQ2-(iv) updating and traceability is related with less than half of the challenges. As we mentioned in Section 4.8, the categories from Table 10 were defined during the conduction of our SLR, so that they present an overview of the testing challenges and also support to identify groups of testing challenges.

Regarding RQ2 (*Which are the challenges imposed to ASs or CASs testing*), we produced an up-to-date, complemented and enhanced list of 35 specific challenges for ASs or CASs testing and characterized them as 13 generic challenges. We noticed that the challenges are mostly related to (i) runtime decisions, which require the testing team to deal (at design time) with the unpredictability of configurations that the system may have; (ii) heterogeneous and distributed environments, consequently leading the testing team to think beyond the boundary of a single system; (iii) interference of users, which requires tests that deal with adaptations that are customized by system users; (iv) updating and traceability between varied software artefacts, which requires the maintenance of artefacts that include the uncertainty, which is inherent of ASs and CASs; (v) context monitoring, which has direct impact on the setup of the test environment; and (vi) generation of test cases, which may be ineffective because of the aforementioned uncertainty.

6.3. Further discussions and research implications

In 2009, the survey of Salehie and Tahvildari [4] established a landscape of the ASs or CASs research area. Regarding software engineering, the authors found out that ‘testing’ was the least focused phase in engineering ASs and CASs, with only a few studies addressing this topic. They argued that the challenging issue is the availability of several alternatives and parameters in the system. Thus, they emphasized that such systems lead to several paths of execution in different scenarios (i.e. when one adds the dynamic decision making, the system will become even more complex). By conducting our SLR, we were able to realize that in the last decade, several testing approaches have emerged in the literature, involving varied techniques, technologies and domains. As a result, there are several testing approaches to deal with testing challenges, even though many of them having only preliminary evaluation. In what follows, we present some general discussions involving (i) the individual and interchangeable usage of ASs or CASs terms in testing-related research and (ii) issues of analysing a possible relationship between testing approaches and testing challenges.

6.3.1. Testing approaches and challenges related to adaptive systems or context-aware systems. Even though we classified the studies as being related to ASs or CASs based on the original authors’ focus (more details in Section 5.2), we may identify testing approaches that could mitigate challenges in *both* types of systems. As an example, the approach of Niebuhr *et al*. [72] focuses on ASs, but it also handles *SC*-5 (which is more closely related to CAS, according to Table 7). In details, the approach establishes the use of a finite number of components and presents how to deal with combinations of them, thus defining test oracles (e.g. by taking as input a set of components and producing as output a possibly different set of components). In other words, the definition of test oracles for dealing with changes at runtime could help to mitigate the challenge in both types of systems. For ASs, we could define these *components* and their interconnections as a specific *configuration* of the target system obtained from a control loop based on MAPE-K. Thus, the definition of oracles could involve configuration A as input and configuration B as output. On the other hand, for CASs, it could be the same idea of ASs but by means of a middleware to generate contexts of configuration A and a middleware to execute actions to achieve configuration B. Note that the concepts of defining inputs and output for test cases could be used in both terms.

As another example, Tse *et al*. [62] focused on CASs, but the underlying approach may handle *SC*-4 (which is more widely investigated for AS, according to Table 7). Even though the authors did not explicitly mention the specific challenge SC-4 (so that the study of Tse *et al*. [62] is not listed for SC-4 in Table 7), their approach – which is based on CASs – has as a central characteristic the use of metamorphic relations to define test sets. Thus, beyond considering changes in the context of CASs, defining test sets to validate structural changes at runtime of an AS is also possible. In other words, the key element is ‘changes’, whatever the changes are, towards the definitions of the metamorphic relations among test cases. Given an example of changing the temperature of an air conditioning, a system could be able to adjust – or even replace – the device properly, and we could define test cases to validate if the adjusted – or new – device is working. By defining a metamorphic relation for the *temperature* attribute, alternative test sets could be defined in order to guarantee some unexpected scenarios (e.g. the new configuration is not able to work over 30°). The difference between an AS and a CAS, in this scenario, could be how to handle the target attribute; for ASs, the *temperature* could be related to the replacement of devices and components, whereas for CASs, the *temperature* could be related to changes in the context values.

6.3.2. Testing challenges and studies that propose and/or apply testing approaches. Given the testing challenges and the studies that propose and/or apply testing approaches for ASs or CASs, we can raise the following question: *Which are the challenges that are directly or indirectly addressed by the testing approaches?* From a different viewpoint, *would the testing approaches be able to deal with the challenges for ASs or CASs?* To answer these questions, it would be necessary to conduct empirical studies to obtain evidence, and this is beyond the scope of this work. In spite of this, we can establish some relationships according to the solutions that have been proposed in the literature for testing these types of systems. As a concrete example, Lu *et al*. [24] dealt with context-dependent data flows (e.g. variables shared for different program executions); thus, the applied approach could support the development of a testing strategy to deal with the specific challenges SC-20 and SC-21 (see Table 7 for more details) and hence with the generic challenge C-7. As another example, Sama *et al*. [29] explored finite-state machines to simulate system behaviour. Therefore, their approach may deal with SC-13. It means that even if the context of a system changes very often, one can predict the main transitions between states. Consequently, such an approach may also support testers to mitigate other challenges related to the C-6 generic challenge.

In the end of Sections 6.1 and 6.2, respectively, we summarized the concentration of effort regarding testing approaches (answer to RQ1) and the main characteristics of the testing challenges (answer to RQ2). Given those concentration of effort and main characteristics of challenges, we can point to some research directions. At first, we call the readers' attention to the fact that this research area (i.e. ASs or CASs testing), in general, lacks evidence to support strong claims about what should be performed next. That said, it is clear that some challenges have already been substantially tackled by the research community. For instance, much work has been performed involving approaches for generating test cases. More specifically, when we analysed the concentration of effort, we found out this concern as the most addressed in recent literature (e.g. since 2014, from the 40 studies addressing testing approaches, 23 of them refer to generating test cases). Therefore, researchers and practitioners have a large variety of options to deal with that challenge. On the other hand, regarding the less recurring concerns, a few approaches refer to (i) prediction of changes [19,92,98,134], which may involve models for representing system states that were not specified in advance; (ii) checkpoints and isolation [66,69,76,115], which may involve classifying and isolating source code blocks related to environment data; and (iii) metamorphic testing [47,62,77,126], which may establish relationships among different test cases. We believe these three topics need more attention from the community.

In 2012, Weyns *et al*. [144] analysed the studies from the SEAMS[31] community by focusing on research results from 2006 to 2008. The authors pointed out an increasing concern for this community with respect to testing these types of systems. In 2013, Bertolino *et al*. [145] concluded that to analyse and test ASs or CASs, it is necessary to formally model the uncertainty in modern systems due to their ubiquitous nature. Regarding this, while conducting our study, we also could note that studies addressing testing under uncertainty share the concern of 'testers should define and specify the uncertainties'. The solutions to deal with uncertainty are related to (i) evolutionary algorithms and prediction models [11,106,123]; (ii) sequences of operation invocations to reveal a fault [61,104,137]; (iii) software architecture [145]; and (iv) data monitoring from sensors in environment perspective [10,27,128,132]. These approaches are defined within (and to be applied

[31]https://www.self-adaptive.org/ – last checked in June 2020.

to) a restricted scope. Intrinsic characteristics of ASs or CASs such as unforeseen configurations and uncertainty *of the external environment* are neglected (or poorly treated). Note that uncertainty may also be related to testing challenges we characterized in this study (for instance, GC-3 – How to detect and avoid during the testing activity incorrect system configurations defined at runtime and GC-6 – How and when to anticipate context changes that impact the behaviour of the system during the testing). Therefore, we also believe that research effort should concentrate in testing under uncertainty.

Note that even though we can identify several testing approaches that propose in advance a general solution, they may work in a very specific situation; consequently, more effort is required for customizing such a solution to deal with a challenge in particular. An example refers to approaches related to *RQ1-(v) built-in test* or *RQ1-(vi) MAPE-K*, which may have been developed using a very specific domain or technology. For example, in the context of approaches for built-in test and MAPE-K, Fredericks *et al*. [10] mentioned that their approach may not achieve similar results in different system domains. Analogously, King *et al*. [70] emphasized that their solution was applied in a prototype that utilized a static structure and predefined test plans. Thus, their validation strategy might be limited to their prototype characteristics. Finally, in our SLR, we could note that there are few initiatives that reported on empirical and controlled studies. This lack of experimental studies regarding the testing of these types of systems can be an impediment for the practical adoption of the proposed solutions.

7. THREATS TO VALIDITY

In Section 4, we presented the goals of this work and key elements of the SLR design. We also mentioned the three main phases involving the SLR (i.e. planning, conducting and reporting) and addressed how these phases were performed in this work. Zhou *et al*. [146] established a set of threats involving the validity of secondary studies. These threats supported us to address which of them are related to our SLR process.

- *Planning – lack of standard terminology* (which may happen because of studies in the literature that use different terms for similar concepts; if a specific concept is not well understood, then the metrics that capture it may be inaccurate or incorrect): in our SLR, we used a narrowed list of terms that are related to AS or CAS. Nevertheless, we noticed variations in the terminology used in the selected studies, even for basic concepts of AS or CAS (e.g. *self-adaptive system*, *self-awareness system*, *ubiquitous system* and *pervasive system*). Thus, the inclusion of other terms (e.g. 'self-*', 'autonomous', 'adaptable' and 'adaptation') might have fetched other relevant studies for this SLR. To diminish this limitation, we used a comprehensive snowballing approach (backward and forward), and as such, we believe we covered a wide spectrum of published research in the investigate topic.
- *Planning – comprehensiveness of the set of venues and/or databases* (which may happen when researchers select an incomplete set of study databases and hence retrieve an incomplete set of studies): in our SLR, we used a particular set of indexed databases and search engines (viz. IEEE Xplore, ScienceDirect, SpringerLink, ACM Digital Library, ISI Web of Science, Scopus and Google Scholar). Querying other databases might also have led to the selection of other relevant studies for this SLR. Likewise in the previously described threat, snowballing helped us diminish this limitation.
- *Conducting – bias in study selection* (which may happen because of missing relevant studies, inaccuracy of data and incorrect classification of publications): in our SLR, particularly in last study selection round, to avoid these threats, we (i) combined automatic search with manual search; (ii) evolved the search string and (iii) searched for additional information sources. Moreover, we contacted authors to eventually obtain studies that were not publicly available for download.
- *Conducting – subjective interpretation about the extracted data* (which may happen because of the incorrect classification of extracted data): in our SLR, we applied a *thematic analysis*

[57] in the studies of the SLR, which involves the identification of recurring themes in the literature, as well as the summarization of the findings based on different thematic headings. In addition, at least two authors of this work participated in all phases and steps involving the interpretation and classification of studies. Thus, the results were naturally double checked during the process of the SLR.

- *Conducting – misclassification of primary studies* (which may be related to biased data extraction): the reader should note that in this study, we did not classify results for several different application domains (e.g. ubiquitous systems, autonomous (unmanned) vehicles and mobile applications). Instead, the studies were analysed and discussed taking into account general properties of ASs or CASs. Therefore, the results provide the reader with a basis for more in-depth analysis. For instance, one can pay particular attention to a subset of testing challenges to identify testing approaches with more specific goals.
- *Conducting – subjective quality assessment* (which may be related to authors' assessment of the criteria based on their own judgement): a threat may be the lack of clear criteria and evaluation, which may lead to bias. In our SLR, we took into account only peer-reviewed, previously assessed studies. This same approach of selecting only peer-reviewed studies was adopted in other SLRs [55,147,148].
- *Reporting – lack of expert evaluation* (regarding the understanding of selected primary studies, which may lead to erroneous conclusions because of the lack of expertise of the reviewer in the subject under investigation): regarding this issue, and hence with the reliability of the results, other research groups could have different keywords involving the application domain for ASs or CASs. Is spite of this, we counted on the expertise of specialists from our research group, and we used multiple type of sources (cf. Table 1). This led us evaluate different sources and different keywords for ASs or CASs.

8. CONCLUSIONS, IMPLICATIONS AND FUTURE WORK

The goal of this article was to characterize the state of the art of AS or CAS testing, by identifying and characterizing testing approaches for AS or CAS and by enhancing the characterization of challenges for AS or CAS testing currently available in the literature. The method we followed to achieve these goals was an SLR.

Even though only experimentation provides evidence on the actual effectiveness of the testing approaches, we believe this kind of study is useful for the research community, as well for practitioners, because it is possible to use our work to cope with the testing challenges by means of the definition of customized testing strategies with focus on recurring and inherent properties of AS or CAS. The literature still lacks a comprehensive review that provides a broad picture of the area. To the best of our knowledge, only in our prior work [1] the reader can find a focused characterization of challenges for testing these types of systems. Moreover, our prior study is outdated and incomplete, because the last analysed study dates from 2016, and we did not analyse testing approaches for ASs or CASs. Therefore, in this article, we provided an up-to-date, refined and extended analysis of the literature.

Regarding the concentration of effort, the generation of test sets was the most recurring research concern. Apart from it, observed trends are the application of combined (hybrid) techniques and the popularity in using model-based approaches.

The characterized challenges and the studies discussed in this paper may help engineers of ASs or CASs have an overview of the issues they will face while developing their systems. Our results also provide initial characteristics of testing approaches based on issues present in particular systems. As an example, while conducting a study (or developing a system) that requires the *generation of test cases* for ASs or CASs, the researcher (or practitioner) may use our results to identify related approaches (e.g. usage of interfaces to generate test cases [76]) and also apply particular techniques (e.g. usage of data flow criteria to define test sets [24,71]). As another example, by dealing with a *heterogeneous and distributed environment*, it is possible to identify in this work (e.g. in Table 10) the challenges related to it and the studies that addressed such challenge (e.g. Kephart and Chess, Sama *et al.*, Tse *et al.* and Püschel *et al.* [20,29,62,109]).

The reader should note that the challenges herein described consider general and specific properties of ASs or CASs that are not necessarily present in all systems. Moreover, in general, the identified testing approaches are proposed taking into account a complete control of the environment; that is, the uncertainty element is disregarded. As an example, Sama *et al*. [29] proposed a model-based approach in which the tester should define all possible states that the system must have. However, the system may be prone to the occurrence of unpredictable values, so an unpredictable state can be generated. In cases like this, intrinsic characteristics of ASs or CASs such as unforeseen configurations and uncertainty of the external environment are neglected (or poorly treated).

Finally, we noted a lack of experimental studies regarding ASs or CASs testing. Despite the existence of some specific repositories related to these types of systems (e.g. SEAMS's source code repository),[32] we believe this limitation is due to the lack of artefacts for specific approaches, technologies or domains. This means that the research questions involving these types of systems are challenging to be evaluated, and consequently, many remain unanswered.

As future work, we intend to contribute for planning experimental studies, promoting an *experimental baseline approaches and artefacts* that would consist of sets of techniques and artefacts that could be used. Moreover, we could indicate possible results that would be achieved with our experimental baseline. We also intend to keep this SLR up to date, so that we can keep track of the progress made by the communities of ASs testing and CASs testing, as well as evolving our search with the inclusion on new related terms.

ACKNOWLEDGEMENTS

This work was partly supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – finance code 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) (grant no. 306310/2016-3) and Federal University of São Carlos. In addition, Valter V. Camargo would like to thank Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (process number 2016/03104-0).

REFERENCES

1. Siqueira BR, Ferrari FC, Serikawa MA, Menotti R, Camargo VV. Characterisation of challenges for testing of adaptive systems. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing (SAST)*. ACM: Maringá, PR, Brazil, 2016; 110–120.
2. Weyns D, Iftikhar M, Soderlund J. 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 (SEAMS)*. IEEE: San Francisco, CA, USA, 2013; 3–12.
3. Lalanda P, McCann JA, Diaconescu A. *Autonomic Computing* (1st edn.) Springer: Springer-Verlag London, 2013.
4. Salehie M, Tahvildari L. Self-adaptive software: landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*. 2009; **4**: 14:1–14:42.
5. Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. Springer: Karlsruhe, Germany, 1999; 304–307.
6. De Lemos R, Garlan D, Ghezzi C, Giese H. *Software Engineering for Self-adaptive Systems III Assurances*, vol. 9640 LNCS. Springer: Dagstuhl Castle, Saarland, Germany, 2017.
7. Shaw M. Beyond objects: a software design paradigm based on process control. *SIGSOFT Software Engineering Notes*. 1995; **20**: 27–38.
8. Wang H, Chan WK, Tse TH. Improving the effectiveness of testing pervasive software via context diversity. *ACM Transactions on Autonomous and Adaptive Systems*. 2014; **9**: 1–28.
9. King TM, Allen AA, Cruz R, Clarke PJ. Safe runtime validation of behavioral adaptations in autonomic software. In *Proceedings of the 8th International Conference Autonomic and Trusted Computing (ATC)*, vol. 6906 LNCS. SpringerLink: Banff, Canada, 2011; 31–46.
10. Fredericks EM, Ramirez AJ, Cheng BHC. Towards run-time testing of dynamic adaptive systems. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, vol. 1. IEEE Press: San Francisco, CA, USA, 2013; 169–174.

[32]https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/ – last checked in June 2020.

11. Jaw LC, Homan D, Crum V, Chou W, Keller K, Swearingen K, Smith T. Model-based approach to validation and verification of flight critical software. In *Proceedings of the 29th International Aerospace Conference (AEROCONF)*. IEEE: Big Sky, Montana, USA, 2008; 1–8.
12. Ferrari FC, Cafeo BBP, Noppen J, Chitchyan R, Rashid A. Investigating testing approaches for dynamically adaptive systems. In *Lightning Talk in the 2nd International Workshop on Variability & Composition (VARICOMP) – In Conjunction with AOSD*. Semantic Scholar: Porto de Galinhas, PE, Brazil, 2011; 1–2.
13. Shevtsov S, Iftikhar MU, Weyns D. SimCA vs ActivFORMS: comparing control- and architecture-based adaptation on the TAS exemplar. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering (CTSE)*. ACM: Bergamo, Italy, 2015; 1–8.
14. Babar MA, Zhang H. Systematic literature reviews in software engineering: preliminary results from interviews with researchers. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE: Washington, DC, USA, 2009; 346–355.
15. Garner P, Hopewell S, Chandler J, MacLehose H, Schnemann HJ, Akl EA, Beyene J, Chang S, Churchill R, Dearness K, Guyatt G, Lefebvre C, Liles B, Marshall R, García LM, Mavergames C, Nasser M, Qaseem A, Sampson M, Soares-Weiser K, Takwoingi Y, Thabane L, Trivella M, Tugwell P, Welsh E, Wilson EC. When and how to update systematic reviews: consensus and checklist, 2016. BMJ, 1–10.
16. Mendes E, Wohlin C, Felizardo K, Kalinowski M. When to update systematic literature reviews in software engineering. *Journal of Systems and Software (JSS)*. 2020.
17. Hurtado S, Sen S, Casallas R. Reusing legacy software in a self-adaptive middleware framework. In *Proceedings of the 12th International Workshop on Adaptive and Reflective Middleware*. ACM: New York, NY, USA, 2011; 29–35.
18. Oreizy P, Gorlick MM, Taylor RN, Heimhigner D, Johnson G, Medvidovic N, Quilici A, Rosenblum DS, Wolf AL. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*. 1999; **14**(3): 54–62.
19. Xu C, Cheung SC, Ma X, C. C, Lv J. Adam: identifying defects in context-aware adaptation. *Journal of Systems and Software (JSS)*. 2012; **85**: 2812–2828.
20. Kephart JO, Chess DM. The vision of autonomic computing. *IEEE Computer*. 2003; **36**(1): 41–50.
21. Krupitzer C, Roth FM, VanSyckel S, Schiele G, Becker C. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*. 2015; **17**: 184–206.10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.
22. de la Iglesia DG, Weyns D. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*. 2015; **10**: 15:1–15:31.
23. Mens T, Serebrenik A, Cleve A. *Evolving Software Systems* (1st edn.) Springer Publishing Company, Incorporated, 2014.
24. Lu H, Chan WK, Tse TH. Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation. In *Proceedings of the 14th International Symposium on Foundations of Software Engineering (FSE)*. ACM: Portland, OR, USA, 2006; 242–252.
25. Wang H, Chan WK. Weaving context sensitivity into test suite construction. In *Proceedings of the 24th International Conference on Automated Software Engineering (ASE)*. IEEE: Auckland, New Zealand, 2009; 610–614.
26. Hamming RW. Error detecting and error correcting codes. *Bell System Technical Journal*. 1950; **29**(2): 147–160.
27. Santos IS, Andrade RMC, Rocha LS, Matalonga S, Oliveira KM, Travassos GH. Test case design for context-aware applications: are we there yet?. *Information and Software Technology*. 2017; **88**: 1–16.
28. Esfahani N, Malek S. 2013. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-adaptive Systems II – A Second Road Map*. Springer: Berlin, Heidelberg; 214–238.
29. Sama M, Elbaum S, Raimondi F, Rosenblum DS, Wang Z. Context-aware adaptive applications: fault patterns and their automated identification. *IEEE Transactions on Software Engineering*. 2010; **36**(5): 644–661.
30. Cheng S, Garlan D, Schmerl B. Evaluating the effectiveness of the Rainbow self-adaptive system. In *The 4th International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009)*: Vancouver, 2009; 132–141.
31. Cámara J, De Lemos R, Laranjeiro N, Ventura R, Vieira M. Testing the robustness of controllers for self-adaptive systems. *Journal of the Brazilian Computer Society*. 2014; **20**(1): 1–14.
32. Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P. Rainbow: architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*. 2004; **37**: 45–54.
33. Cámara J, De Lemos R, Laranjeiro N, Ventura R, Vieira M. Robustness evaluation of controllers in self-adaptive software systems. In *Proceedings of the 6th Latin-American Symposium on Dependable Computing (LADC)*. IEEE Press: Rio de Janeiro, RJ, Brazil, 2013; 1–10.
34. Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J, Andersson J, Becker B, Bencomo N, Brun Y, Cukic B, Serugendo GDM, Dustdar S, Finkelstein A, Gacek C, Geihs K, Grassi V, Karsai G, Kienle HM, Kramer J, Litoiu M, Malek S, Mirandola R, Müller HA, Park S, Shaw M, Tichy M, Tivoli M, Weyns D, Whittle J. Software engineering for self-adaptive systems: a research roadmap, 2009; 1–26.
35. IEEE Press. IEEE Press 829 standard for software and system test documentation. *Standard*. In 610.12, Institute of Electric and Electronic Engineers, New York, NY, USA, 1990.
36. Püschel G, Götz S, Wilke C, Aßmann U. Towards systematic model-based testing of self-adaptive software. In *Proceedings of the 5th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE)*. IARIA Computer Society: Valencia, Spain, 2013; 65–70.

37. Matalonga S, Rodrigues F, Travassos GH. Characterizing testing methods for context-aware software systems: results from a quasi-systematic literature review. *Journal of Systems and Software (JSS)*. 2017; **131**: 1–21.
38. Matalonga S, Rodrigues F, Travassos GH. Challenges in testing context aware software systems. In *Proceedings of the 9th Brazilian Workshop on Systematic and Automated Software Testing (SAST)*. ACM: Belo Horizonte, MG, Brazil, 2015; 51–60.
39. Matalonga S, Rodrigues F, Travassos GH. Matching context aware software testing design techniques to ISO/IEC/IEEE 29119. In *Proceedings of the 15th International Conference Software Process Improvement and Capability Determination (SPICE)*. Springer: Gothenburg, Sweden, 2015; 33–44.
40. Almeida DR, Machado PatriciaDL, Andrade WL. Testing tools for android context-aware applications: a systematic mapping. *Journal of the Brazilian Computer Society*. 2019; **25**(1): 12.
41. Siqueira BR, Ferrari FC, Souza KE, Santibáñez DSM, Camargo VV. Fault types of adaptive and context-aware systems and their relationship with fault-based testing approaches. In *Proceedings of the 15th International Workshop on Mutation Analysis (MUTATION)*. IEEE Press: Porto, Portugal, 2020; 284–293.
42. Bartel A, Baudry B, Munoz F, Klein J, Mouelhi T, Le-Traon Y. Model driven mutation applied to adaptative systems testing. In *Proceedings of the 6th International Workshop on Mutation Analysis (MUTATION)*. IEEE Press: Berlin, Germany, 2011; 408–413.
43. Eberhardinger B, Habermaier A, Seebach H, Reif W. Back-to-back testing of self-organization mechanisms. In *Proceedings of the 28th International Conference of Testing Software and Systems (ICTSS)*. Springer: Graz, Austria, 2016; 18–35.
44. Püschel G, Götz S, Wilke C, Piechnick C, Aßmann U. Testing self-adaptive software: requirement analysis and solution scheme. *International Journal on Advances in Software*. 2014; **2628**: 88–100.
45. Sama M, Rosenblum DS, Wang Z, Elbaum S. Model-based fault detection in context-aware adaptive applications. In *Proceedings of the 16th International Symposium on Foundations of Software Engineering (FSE)*. ACM: Atlanta, GA, USA, 2008; 261–271.
46. Sama M, Rosenblum DS, Wang Z, Elbaum S. Multi-layer faults in the architectures of mobile, context-aware adaptive applications. *Journal of Systems and Software (JSS)*. 2010; **83**(6): 906–914.
47. Siqueira BR, Costa Júnior M, Ferrari FC, Santibáñez DSM, Menotti R, Camargo VV. Experimenting with a multi-approach testing strategy for adaptive systems. In *Proceedings of the 17th Brazilian Symposium on Software Quality (SBQS)*. ACM: Curitiba, PR, Brazil, 2018; 111–120.
48. Wang H, Chan WK, Tse TH. Correlating context-awareness and mutation analysis for pervasive computing systems. In *Proceedings of the 10th International Conference on Quality Software (QSIC)*. IEEE: Zhangjiajie, China, 2010; 151–160.
49. Yu L, Gao J. Generating test cases for context-aware applications using bigraphs. In *Proceedings of the 8th International Conference on Software Security and Reliability (SERE)*. IEEE Press: San Francisco, CA, USA, 2014; 137–146.
50. Yu L, Tsai WT, Perrone G. Testing context-aware applications based on bigraphical modeling. *IEEE Transactions on Reliability*. 2016; **65**: 1584–1611.
51. Kitchenham BA, Charters S. Guidelines for performing systematic literature reviews in software engineering. *Technical Report TR*. In (EBSE) 2007-001, Keele University and Durham University Joint Report, Keele, Staffs, UK, 2007.
52. Fabbri SCPF, Felizardo KR, Ferrari FC, Hernandes ECM, Octaviano FR, Nakagawa EY, Maldonado JC. Externalising tacit knowledge of the systematic review process. *IET Software*. 2013; **7**: 298–307.
53. Dieste O, Grimán A, Juristo N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*. 2009; **14**(5): 513–539.
54. Santos JAM, Santos AR, Mendonça M. Investigating bias in the search phase of software engineering secondary studies. In *18th Ibero-American Conference on Software Engineering (ESELAW 2015)*. UCSP, URP,SPC,UCSP, 2015; 488–501.
55. Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM: London, EN, UK, 2014; 1–10.
56. Arcaini P, Mirandola R, Riccobene E, Scandurra P. Model-based testing for MAPE-K adaptation control loops. In *Proceedings of the 16th Workshop on Advances in Model Based Testing (A-MOST)*, 2020; 43–51.
57. Dixon-Woods M, Agarwal S, Jones D, Young B, Sutton A. Synthesising qualitative and quantitative evidence: a review of possible methods. *Journal of Health Services Research & Policy (JHSRP)*. 2005; **10**: 45–53.
58. Truszkowski W, Hinchey M, Rash J, Rouff C. NASA's swarm missions: the challenge of building autonomous software. *IT Professional*. 2004; **6**: 47–52.
59. Schumann J, Visser W. Autonomy software: V&V challenges and characteristics. In *Proceedings of the 26th International Aerospace Conference (AEROCONF)*. IEEE: Big Sky, MT, USA, 2006; 1–6.
60. Niebuhr D, Rausch A. A concept for dynamic wiring of components: correctness in dynamic adaptive systems. In *Proceedings of the 6th Workshop on Specification and Verification of Component-Based Systems (SAVCBS) – Held in Conjunction with ESEC/FSE*. ACM Press: Dubrovnik, Croatia, 2007; 101–102.
61. Welsh K, Sawyer P. Managing testing complexity in dynamically adaptive systems: a model-driven approach. In *Proceedings of the 1st International Workshop on Validation and Verification of Dynamic Systems (VIDAS)*. IEEE: Paris, France, 2010; 290–298.

62. Tse TH, Chan WK, Yau SS, Lu H, Chen TY. Testing context-sensitive middleware-based software applications. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC)*. IEEE: Hong Kong, China, 2004; 1–9.
63. Micskei Z, Szatmári Z, Oláh J, Majzik I. A concept for testing robustness and safety of the context-aware behaviour of autonomous systems. In *Proceedings of the 6th International Conference of Agent and Multi-Agent Systems (KES-AMSTA)*, vol. 7327 LNAI. SpringerLink: Dubrovnik, Croatia, 2012; 504–513.
64. Vieira V, Holl K, Hassel M. A context simulator as testing support for mobile apps. In *Proceedings of the 30th Symposium on Applied Computing (SAC)*. ACM: New York, NY, USA, 2015; 535–541.
65. Flores A, Augusto JC, Polo M, Varea M. Towards context-aware testing for semantic interoperability on PvC environments. In *Proceedings of the 17th International Conference on Systems, Man and Cybernetics (SMC)*, vol. 2. IEEE: The Hague, The Netherlands, 2004; 1136–1141.
66. Chan WK, Chen TY, Lu H, Tse TH, Yau SS. Integration testing of context-sensitive middleware-based applications: a metamorphic approach. *International Journal of Software Engineering and Knowledge Engineering*. 2006; **16**(5): 677–704.
67. Chan WK, Chen TY, Lu H. A metamorphic approach to integration testing of context-sensitive middleware-based applications. In *Proceedings of the 5th International Conference on Quality Software (QSIC)*. IEEE Press: Melbourne, Australia, 2005; 241–249.
68. Merdes M, Malaka R, Suliman D, Paech B, Brenner D, Atkinson C. Ubiquitous RATs: how resource-aware run-time tests can improve ubiquitous software system. In *Proceedings of the 6th International Workshop on Software Engineering and Middleware (SEM)*. ACM: New York, NY, USA, 2006; 55–62.
69. Wang Z, Elbaum S, Rosenblum DS. Automated generation of context-aware tests. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*. IEEE: Minneapolis, MN, USA, 2007; 406–415.
70. King TM, Ramirez A, Cruz R, Clarke P. An integrated self-testing framework for autonomic computing systems. *Journal of Computers*. 2007; **2**(9): 37–49.
71. Lu H, Chan WK, Tse TH. Testing pervasive software in the presence of context inconsistency resolution services. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. IEEE Press: Leipzig, Germany, 2008; 61–70.
72. Niebuhr D, Rausch A, Klein C, Reichmann J, Schmid R. Achieving dependable component bindings in dynamic adaptive systems – a runtime testing approach. In *Proceedings of the 3rd International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE: San Francisco, CA, USA, 2009; 186–197.
73. Niebuhr D, Rausch A. Guaranteeing correctness of component bindings in dynamic adaptive systems. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE: Patras, Greece, 2009; 454–457.
74. Niebuhr D, Rausch A. Guaranteeing correctness of component bindings in dynamic adaptive systems based on runtime testing. In *Proceedings of the 4th International Workshop on Services Integration in Pervasive Environments (SIPE)*. ACM Press: New York, NY, USA, 2009; 7–12.
75. Ye C, Cheung SC, Wei J, Zhong H, Huang T. A study on the replaceability of context-aware middleware. In *Proceedings of the 1st Asia-Pacific Symposium on Internetware (Internetware)*. ACM Press: Beijing, China, 2009; 4:1–4:10.
76. Taranu S, Tiemann J. General method for testing context aware applications. In *Proceedings of the 6th International Workshop on Managing Ubiquitous Communications and Services (MUCS)*. ACM Press: New York, NY, USA, 2009; 3–8.
77. Wang H, Kong H, Chan WK. On the construction of context-aware test suites. *Technical Report*. In TR-2010-01, Laboratory for Computer Science, University of Hong Kong, Hong Kong, China, 2010.
78. Vassev E, Hinchey M, Nixon P. Automated test case generation of self-managing policies for NASA prototype missions developed with ASSL. In *Proceedings of the 4th International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE: Taipei, Taiwan, 2010; 3–8.
79. Da Costa AD, Nunes C, Da Silva VT, Fonseca B, De Lucena CJP. JAAF+T: a framework to implement self-adaptive agents that apply self-test. In *Proceedings of the 25th Symposium on Applied Computing (SAC)*. ACM Press: New York, NY, USA, 2010; 928–935.
80. King TM, Allen A, Wu Y, Clarke P, Ramirez AE. A comparative case study on the engineering of self-testable autonomic software. In *Proceedings of the 8th International Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASE)*, vol. 0. IEEE Press: Los Alamitos, CA, USA, 2011; 59–68.
81. King TM, Babich D, Alava J, Clarke PJ, Stevens R. Towards self-testing in autonomic computing systems. In *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS)*. IEEE Press: Sedona, Arizona, USA, 2007; 51–58.
82. King TM, Ramirez A, Clarke PJ, Quinones-Morales B. A reusable object-oriented design to support self-testable autonomic software. In *Proceedings of the 23th Symposium on Applied Computing (SAC)*. ACM Press: Fortaleza, CE, Brazil, 2008; 1664–1669.
83. Ramirez AJ, Morales B, King TM. A self-testing autonomic job scheduler. In *Proceedings of the 46th ACM Annual Southeast Regional Conference (ACMSE)*. ACM Press: New York, NY, USA, 2008; 304–309.
84. Stevens R, Stevens R, Parsons B, Parsons B, King TM. A self-testing autonomic container. In *Proceedings of the 45th Annual Southeast Regional Conference (ACM-SE)*. ACM Press: Winston-Salem, NC, USA, 2007; 1–6.

85. Silva CE, De Lemos R. Dynamic plans for integration testing of self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM: Honolulu, HI, USA, 2011; 148–157.
86. Eze T, Anthony RJ, Walshaw C, Soper A. The challenge of validation for autonomic and self-managing systems. In *Proceedings of the 7th International Conference on Autonomic and Autonomous Systems (ICAS)*. ThinkMind: Venice, Italy, 2011; 128–133.
87. Wotawa F. Adaptive autonomous systems – from the system's architecture to testing. *Communications in Computer and Information Science*. 2012; **336 CCIS**: 76–90.
88. Püschel G, Seiger R, Schlegel T. Test modeling for context-aware ubiquitous applications with feature Petri nets. In *Proceedings of the 2nd Workshop on Model-Based Interactive Ubiquitous Systems (MODIQUITOUS)*, vol. 947. ACM: Copenhagen, Denmark, 2012; 37–40.
89. Weyns D. Towards an integrated approach for validating qualities of self-adaptive systems. In *Proceedings of the 9th International Workshop on Dynamic Analysis (WODA)*. ACM: Minneapolis, MN, USA, 2012; 24–29.
90. Bayha A, Grüneis F, Schätz B. Model-based software in-the-loop-test of autonomous systems. In *Symposium on Theory of Modeling and Simulation – DEVS Integrative M&S Symposium (TMS/DEVS)*. Society for Computer Simulation International: San Diego, CA, USA, 2012; 30:1–30:6.
91. Amalfitano D, Fasolino AR, Tramontana P, Amatucci N. Considering context events in event-based testing of mobile applications. In *Proceedings of the 4th International Workshop on Testing Techniques and Experimentation Benchmarks for Event-Driven Software (TESTBEDS)*. IEEE Press: Luxembourg City, Luxembourg, 2013; 126–133.
92. Garvin BJ, Cohen MB, Dwyer MB. 2013. Failure avoidance in configurable systems through feature locality. In *Assurances for Self-adaptive Systems*. Springer: Springer-Verlag Berlin Heidelberg; 266–296.
93. Garvin BJ, Cohen MB, Dwyer MB. Using feature locality: can we leverage history to avoid failures during reconfiguration?. In *Proceedings of the 8th Workshop on Assurances for Self-Adaptive Systems (ASAS)*. ACM: Szeged, Hungary, 2011; 24–33.
94. Horányi G, Micskei Z, Majzik I. Scenario-based automated evaluation of test traces of autonomous systems. In *Workshop on Dependable Embedded and Cyber-Physical Systems (DECS) – Held in Conjunction with the 32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP)*. Inria: Toulouse, France, 2013; NA.
95. Horányi G, Majzik I. Automated evaluation of the test traces of autonomous systems. *Technical Report*, BME, Budapest, Hungary, 2013.
96. Nehring K, Liggesmeyer P. Testing the reconfiguration of adaptive systems. In *Proceedings of the 5th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE)*. CiteSeerX: Valencia, Spain, 2013; 14–19.
97. Silva CE, De Lemos R. Dynamic management of integration testing for self-adaptive systems. In *Proceedings of the 1st Workshop on Dependable in Adaptive and Self-Managing Systems (WDAS)*. KAR Repository: Rio de Janeiro, RJ, Brazil, 2013; 3–11.
98. Xu C, Cheung SC, Ma X, Cao C, Lu J. Detecting faults in context-aware adaptation. *International Journal Software Informatics (JSI)*. 2013; **7**: 85–111.
99. Xu C, Cheung SC, Ma X, Cao C, Lu J. Dynamic fault detection in context-aware adaptation. In *Proceedings of the 4th Asia-Pacific Symposium on Internetware (Internetware)*. ACM Press: Qingdao, China, 2012; 1:1–1:10.
100. Akour M, Akour I, King TM. Runtime test case synchronization in adaptive software. *International Journal of Business Research and Information Technology (IJBRIT)*. 2014; **1**(1): 10–29.
101. Akour M, Jaidev A, King TM. Towards change propagating test models in autonomic and adaptive systems. In *Proceedings of the 18th International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS)*. IEEE Press: Las Vegas, NV, USA, 2011; 89–96.
102. Eberhardinger B, Seebach H, Knapp A, Reif W. Towards testing self-organizing, adaptive systems. In *Proceedings of the 26th International Conference Testing Software and Systems (ICTSS)*. Springer: Madrid, Spain, 2014; 180–185.
103. Eberhardinger B. Testing self-organizing, adaptive systems. In *PhD Symposium – Held in Conjunction with the 9th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE: Cambridge, MA, USA, 2015; 140–145.
104. Ma T, Ali S, Yue T. Modeling foundations for executable model-based testing of self-healing cyber-physical systems. *Software & Systems Modeling (JSM)*. 2018; **1**(1): 1–31.
105. Mehmood MA, Khan MNA, Afzal W. Automating test data generation for testing context-aware applications. In *Proceedings of the 9th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE Press: Beijing, China, 2018; 104–108.
106. Fredericks EM, DeVries B, Cheng BHC. Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, vol. 1. IEEE: Hyderabad, India, 2014; 17–26.
107. Griebe T, Gruhn V. A model-based approach to test automation for context-aware mobile applications. In *Proceedings of the 29th Symposium on Applied Computing (SAC)*. ACM: New York, NY, USA, 2014; 420–427.
108. Püschel G, Piechnick C, Götz S, Seidl C, Richly S, Schlegel T, Aßmann U. A combined simulation and test case generation strategy for self-adaptive systems. *International Journal on Advances in Software*. 2014; **7**(3 & 4): 686–696.

109. Püschel G, Piechnick C, Götz S, Seidl C, Richly S, Schlegel T, Aßmann U. A black box validation strategy for self-adaptive systems. In *Proceedings of the 6th International Conference on Adaptive and Self-Adaptive Systems and Applications (ICAS)*. CiteSeer: Chamonix, France, 2014; 111–116.
110. Cámara J, De Lemos R, Laranjeiro N, Ventura R, Vieira M. Robustness-driven resilience evaluation of self-adaptive software systems. *IEEE Transactions on Dependable and Secure Computing*. 2015; **6**(1): 1–1.
111. Fredericks EM, Cheng BHC. Automated generation of adaptive test plans for self-adaptive systems. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, vol. 1. IEEE Press: Florence, Italy, 2015; 157–167.
112. Hansel J, Vogel T, Giese H. A testing scheme for self-adaptive software systems with architectural runtime models. In *Proceedings of the 2nd Workshop on Quality Assurance for Self-Adaptive, Self-Organising Systems (QA4SASO)*. IEEE Press: Cambridge, MA, USA, 2015; 134–139.
113. Lahami M, Krichen M, Barhoumi H, Jmaiel M. Selective test generation approach for testing dynamic behavioral adaptations. In *Proceedings of the 27th International Conference on Testing Software and Systems (ICTSS)*. Springer: New York, NY, USA, 2015; 224–239.
114. Lim YJ, Jee E, Shin D, Bae DH. Efficient testing of self-adaptive behaviors in collective adaptive systems. In *Proceedings of the 39th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE: Taichung, Taiwan, 2015; 216–221.
115. Majchrzak TA, Matthias S. Context-dependent testing of applications for mobile devices. *Open Journal of Web Technologies (OJWT)*. 2015; **2**: 27–39.
116. Markov G, Fröhlich J. An ecosystem approach for testing self-organizing, adaptive systems. In *Proceedings of the 2nd Workshop on Quality Assurance for Self-Adaptive, Self-Organising Systems (QA4SASO)*. IEEE Press: Cambridge, MA, USA, 2015; 120–121.
117. Sen S, Alesio SD, Marijan D, Sarkar A. Evaluating reconfiguration impact in self-adaptive systems – an approach based on combinatorial interaction testing. In *Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE: Funchal, Madeira, Portugal, 2015; 250–254.
118. Song K, Han AR, Jeong S, Cha S. Generating various contexts from permissions for testing Android applications. In *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Semantic Scholar: Pittsburgh, PA, USA, 2015; 87–92.
119. Tonjes R, Reetz ES, Fischer M, Kuemper D. Automated testing of context-aware applications. In *Proceedings of the 82nd Vehicular Technology Conference (VTC Fall)*. IEEE Press: Boston, MA, USA, 2015; 1–5.
120. Al-Refai M, Cazzola W, Ghosh S, France R. Using models to validate unanticipated, fine-grained adaptations at runtime. In *Proceedings of the 17th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE Press: Orlando, FL, USA, 2016; 23–30.
121. Al-Refai M, Ghosh S, Cazzola W. Model-based regression test selection for validating runtime adaptation of software systems. In *Proceedings of the 9th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Press, 2016; 288–298.
122. Heck H, Rudolph S, Gruhl C, Wacker A, Hähner J, Sick B, Tomforde S. Towards autonomous self-tests at runtime. In *Proceedings of the 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. IEEE: Augsburg, Germany, 2016; 98–99.
123. Qin Y, Xu C, Yu P, Lu J. SIT: sampling-based interactive testing for self-adaptive apps. *Journal of Systems and Software (JSS)*. 2016; **120**: 70–88.
124. Rodrigues F, Matalonga S, Travassos GH. CATS design: a context-aware test suite design process. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing (SAST)*. ACM Press: Maringá, PR, Brazil, 2016; 9:1–9:10.
125. Wotawa F. Testing self-adaptive systems using fault injection and combinatorial testing. In *Proceedings of the 1st International Workshop on Verification and Validation of Adaptive Systems (VVASS)*. IEEE: Vienna, Austria, 2016; 305–310.
126. Lindvall M, Porter A, Magnusson G, Schulze C. Metamorphic model-based testing of autonomous systems. In *Proceedings of the 2nd International Workshop on Metamorphic Testing (MET)*: Buenos Aires, Argentina, 2017; 35–41.
127. Luo C, Kuutila M, Klakegg S, Ferreira D, Flores H, Goncalves J, Mäntylä M, Kostakos V. TestAWARE: a laboratory-oriented testing tool for mobile context-aware applications. *The Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*. 2017; **1**(3): 80:1–80:29.
128. Matalonga S, Travassos GH. Testing context-aware software systems: unchain the context, set it free!. In *Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES)*. ACM: Fortaleza, CE, Brazil, 2017; 250–254.
129. Mehmood MA, Khan MNA, Afzal W. Transforming context-aware application development model into a testing model. In *Proceedings of the 8th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE Press: Beijing, China, 2017; 177–182.
130. Usaola MP, Rojas G, Rodrguez I, Herndez S. An architecture for the development of mutation operators. In *Proceedings of the 12th International Workshop on Mutation Analysis (MUTATION)*. IEEE Press: Tokyo, Japan, 2017; 143–148.
131. Eberhardinger B, Ponsar H, Siegert G, Reif W. Case study: adaptive test automation for testing an adaptive hadoop resource manager. In *Proceedings of the 18th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE Press: Lisbon, Portugal, 2018; 513–518.

132. Fredericks EM. An empirical analysis of the mutation operator for run-time adaptive testing in self-adaptive systems. In *Proceedings of the 11th International Workshop on Search-Based Software Testing (SBST)*. IEEE Press: Gothenburg, Sweden, 2018; 59–66.
133. Mirza AM, Khan MNA. An automated functional testing framework for context-aware applications. *IEEE Computer*. 2018; **6**(1): 46568–46583.
134. Reichstaller A, Knapp A. Risk-based testing of self-adaptive systems using run-time predictions. In *Proceedings of the 12nd International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE Press: Trento, Italy, 2018; 80–89.
135. Reichstaller A, Gabor T, Knapp A. Mutation-based test suite evolution for self-organizing systems. In *Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods (ISOLA)*. Springer International Publishing: Limassol, Cyprus, 2018; 118–136.
136. Reichstaller A, Eberhardinger B, Ponsar H, Knapp A, Reif W. Test suite reduction for self-organizing systems: a mutation-based approach. In *Proceedings of the 13th International Workshop on Automation of Software Test (AST)*. ACM Press: Gothenburg, Sweden, 2018; 64–70.
137. Ma T, Ali S, Yue T, Elaasar M. Testing self-healing cyber-physical systems under uncertainty: a fragility-oriented approach. *Software Quality Journal (SQJ)*. 2019; **27**(1): 1–35.
138. Liu Y. PhoneAdapter, 2013. Online http://sccpu2.cse.ust.hk/afchecker/phoneadapter.html – accessed in May 2020.
139. Myers GJ, Sandler C, Badgett T. *The Art of Software Testing* (3rd edn.) Wiley, 2011.
140. Simão A, Petrenko A. Generating checking sequences for partial reduced finite state machines. In *Testing of Software and Communicating Systems*, Suzuki K, Higashino T, Ulrich A, Hasegawa T (eds). Springer Berlin Heidelberg: Berlin, Heidelberg, 2008; 153–168.
141. DeMillo RA, Lipton RJ, Sayward FG. Hints on test data selection: help for the practicing programmer. *IEEE Computer*. 1978; **11**: 34–43.
142. Chen TY, Kuo F, Tam WK, Merkel RG. Testing a software-based PID controller using metamorphic testing. In *Proceeding of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems*. SciTePress: Vilamoura, Portugal, 2011; 387–396.
143. Weyuker EJ. On testing non-testable programs. *The Computer Journal*. 1982; **25**(4): 465–470.
144. Weyns D, Iftikhar MU, Malek S, Andersson J. Claims and supporting evidence for self-adaptive systems: a literature study. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012; 89–98.
145. Bertolino A, Inverardi P, Muccini H. Software architecture-based analysis and testing: a look into achievements and future challenges. *Computing*. 2013; **95**(8): 633–648.
146. Zhou X, Jin Y, Zhang H, Li S, Huang X. A map of threats to validity of systematic literature reviews in software engineering. In *Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE Press: Hamilton, New Zealand, 2017; 153–160.
147. MacDonell S, Shepperd M, Kitchenham B, Mendes E. How reliable are systematic reviews in empirical software engineering? *IEEE Transactions on Software Engineering*. 2010; **36**(5): 676–687.
148. Pizzoleto AV, Ferrari FC, Offutt AJ, Fernandes L, Ribeiro M. A systematic literature review of techniques and metrics to reduce the cost of mutation testing. *Journal of Systems and Software*. 2019; **157**: 1–39.

APPENDIX: IS AN UPDATED SECONDARY STUDY NEEDED?

In the application of the framework proposed by Garner et al. [15] and evaluated by Mendes *et al*. [16], we used as a baseline our prior SLR report presented in a conference paper [1] and executed the proposed checklist. The checklist and responses are shown in the table below. The table reveals that the responses for the three questions in step 1 are YES, which enables us to proceed to the next step. At least one YES response in step 2 enables us to move on to the last step. In the third step, at least one YES response gives us confirmation to proceed with the study update and extension.

Framework step	Response
Step 1.a – Does the published SLR still address a current question?	YES
Step 1.b – Has the SLR had good access or use?	YES
Step 1.c – Has the SLR used valid methods and was well-conducted?	YES
Step 2.a – Are there any new relevant methods?	YES
Step 2.b – Are there any new studies, or new information?	YES
Step 3.a – Will the adoption of new methods change the findings, conclusions or credibility?	YES
Step 3.b – Will the inclusion of new studies/information/data change findings, conclusions or credibility?	YES

APPENDIX: SYSTEMATIC LITERATURE REVIEW PROTOCOL

Testing of Adaptive Systems and Context-aware Systems

Team: Bento R. Siqueira (Federal University of São Carlos, São Carlos, Brazil); Fabiano C. Ferrari (Federal University of São Carlos, São Carlos, Brazil); Kathiani E. Souza (Federal University of São Carlos, São Carlos, Brazil); Daniel S. M. Santíbãnez (Federal University of São Carlos, São Carlos, Brazil); Valter V. Camargo (Federal University of São Carlos, São Carlos, Brazil); Rogério de Lemos (University of Kent, Canterbury, UK).

Primary study selection: Bento R. Siqueira; Fabiano C. Ferrari.

Description

This document consists in a protocol to be followed over the course of the conduction of our systematic literature review. To establish this protocol, we followed the guidelines for conducting secondary studies proposed by Kitchenham and Charters [51] and Wohlin [55]. The next sections describe how we followed these guidelines to answer the research questions posed by this study.

Question Formulation

Question focus (objectives). Identifying testing approaches for adaptive systems (ASs) or context-aware systems (CASs); characterizing challenges to test these types of systems; and discussing research directions and recommendations for the area.

Note: Previous initial results have already been reported in our prior studies [1,12,41].

Questions.

- RQ1: Which are the testing approaches that are proposed for ASs or CASs?
- RQ2: Which are the testing challenges for ASs or CASs?

Keywords and synonyms.

- Group 1: ‘testing’;
- Group 2: ‘adaptive system’, ‘adaptive software’, ‘context-aware’ and ‘autonomic’.

Base search string. (‘testing’) AND (‘adaptive systems’ OR ‘adaptive system’ OR ‘context aware’ OR ‘context-aware’ OR ‘context awareness’ OR ‘context-awareness’ OR ‘adaptive software’ OR ‘autonomic’).

Procedure, Control and Search

Procedure. In our work, we looked for the set of studies that propose or apply testing approaches to ASs or CASs.

Control and search. In our work, we defined the control group as the set of studies selected in the very preliminary round (herein named round 1) of this SLR [12]. However, we could have a limited search scope by using only search strings [55]. Therefore, we also applied the backward and forward snowballing techniques.

Selection of Sources

Criteria for selection of sources. We selected traditional repositories of scientific literature in the field of computer science. Besides that, the Google Scholar search engine was included to apply the snowballing techniques.

List of selected sources.

- IEEE Xplore,
- Elsevier ScienceDirect,
- Springer SpringerLink,

- ACM Digital Library,
- Clarivate Web of Science,
- Elsevier Scopus and
- Google Scholar (for snowballing).

Selection of Studies

This section defines the inclusion and exclusion criteria that must be used for study selection.

Inclusion (I) criteria.

I1: It defines or applies testing approaches to ASs or CASs.
I2: It characterizes challenges for AS testing or CAS testing.
I3: It characterizes types of faults that are specific to ASs or CASs.

A study is selected if it passes (*I*1 OR *I*2 OR *I*3).

Exclusion (E) Criteria.
E1: It does not fulfil any inclusion criterion.
E2: It is not written in English.
E3: It is non-peer reviewed.

Studies that fall into at least one of the earlier categories are not selected.

Study selection process. The search strategy comprises performing automatic search using our base search string with eventual customizations required by the specific search engines. The search strategy also includes backward and forward snowballing techniques. To identify evidence that enables us to provide answers to the research questions, the search string must be built by taking into account the two main domains (i.e. testing and (adaptive systems or context-aware systems)).

During study selection, the inclusion and exclusion criteria are applied. Initial filtering of studies (also known as *preselection* step) consists in applying the criteria to the title and abstract of the retrieved studies. However, it may be the case the selection requires a more thorough analysis of the studies. During the second step of the search process, one reviewer read the candidate studies in their entirety. Indecision on whether a paper should be selected or not is resolved by discussion with, at least, one other reviewer.

Regarding the application of the snowballing techniques, references and citations to secondary studies that are related to our study must be also be analysed.

General procedures:

- In any selection step, duplicated entries must be discarded.
- Up-to-date studies must replace prior versions of the same study. Examples are studies that update a technique previously published or studies that extend a prior publication. Studies removed in the replacement process are said to be *subsumed* by more recent studies.

Synthesis of Results

Extracted data from selected studies must be analysed in two phases:

1 characterization of testing challenges and
2 analysis of testing approaches.

Phase 1 encompasses the full reading and analysis of the selected primary studies, particularly the ones that mention difficulties and problems with respect to AS testing or CAS testing (see inclusion criterion (ii)). It may be the case that different authors describe a same challenge by using different words, terms, context or level of detail. Thus, the studies that mention challenges must be organized in groups, thus allowing us to devise more general categories of challenges for AS testing, CAS testing or both.

Phase 2 consists of investigating the testing approaches and identifying the main characteristics and techniques applied by the authors.

Additional information

Main data extraction fields.

- a summary of the study,
- testing techniques (e.g. functional, structural and fault based),
- test selection criteria (e.g. equivalence partitioning, all nodes and mutation testing),
- types of faults,
- challenges for testing ASs or CASs, and
- type of studies (e.g. controlled experiment and case study).