Alternatives for testing of context-aware software systems in non-academic settings

Matalonga, Santiago; Amalfitano, Domenico; Doreste, Andrea; Fasolino, Anna Rita; Horta Travassos, Guilherme

Published in:
Information and Software Technology

DOI:
10.1016/j.infsof.2022.106937

E-pub ahead of print: 30/09/2022

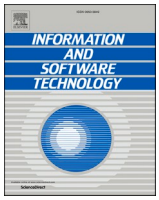
Document Version
Publisher's PDF, also known as Version of record

Information and Software Technology 149 (2022) 106937

Contents lists available at ScienceDirect

Information and Software Technology

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



Alternatives for testing of context-aware software systems in non-academic settings: results from a *Rapid Review*

Santiago Matalonga [a,*], Domenico Amalfitano [b], Andrea Doreste [c], Anna Rita Fasolino [b], Guilherme Horta Travassos [c]

[a] School of Computing, Engineering, and Physical Science. The University of the West of Scotland, Paisley, Renfrewshire, United Kingdom
[b] Department of Electrical Engineering and Information Technology, DIETI. University of Naples Federico II, Naples, Italy
[c] System Engineering and Computer Science, COPPE. Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil

ARTICLE INFO

Keywords:
Context-aware software systems
Software testing
Rapid review
Contemporary software systems

ABSTRACT

Context: Context-awareness challenges the engineering of contemporary software systems and jeopardizes their testing. The variation of context represents a relevant behavior that deepens the limitations of available software testing practices and technologies. However, such software systems are mainstream. Therefore, researchers in non-academic settings also face challenges when developing and testing contemporary software systems.
Objective: To understand how researchers deal with the variation of context when testing context-aware software systems developed in non-academic settings.
Method: To undertake a secondary study (*Rapid Review*) to uncover the necessary evidence from primary sources describing the testing of context-aware software systems outside academia.
Results: The current testing initiatives in non-academic settings aim to generate or improve test suites that can deal with the context variation and the sheer volume of test input possibilities. They mostly rely on modeling the systems' dynamic behavior and increasing computing resources to generate test inputs to achieve this. We found no evidence of test results aiming at managing context variation through the testing lifecycle process.
Conclusions: So far, the identified testing initiatives and strategies are not ready for mainstream adoption. They are all domain-specific, and while the ideas and approaches can be reproduced in distinct settings, the technologies are to be re-engineered and tailored to the context-awareness of contemporary software systems in different problem domains. Further and joint investigations in academia and experiences in non-academic settings can evolve the body of knowledge regarding the testing of contemporary software systems in the field.

1. Introduction

Contemporary software systems (CSS) refer to software systems that demand integrating devices and communications technologies [1]. Examples of CSS domains include ambient intelligence, assisted living, systems of systems, the internet of things, cyber-physical systems, autonomous systems, and industry 4.0, which are now mainstream. CSS refers to software systems that demand integrating devices and communications technologies [1]. In these systems, physical objects with embedded software are interconnected by networks to provide services to the system's actors. Sensors, computer devices, and applications should interact, exchange information, and work with distinct elements to guarantee adequate functionalities. Therefore, CSS exploit technologies that offer challenges for their construction since they question the traditional form of developing software. CSS exhibit several characteristics (like service discoverability and interoperation). Among the characteristics that these systems exhibit is context awareness [2] which is the focus of this work because it challenges current engineering practices [3], particularly when related to testing.

Previous investigations highlighted relevant characteristics regarding the engineering of context-aware software systems (CASS) [2,4,5]. The hype of the press went from CEOs arguing that *"driverless trucks will never happen"* [6] to headlining the first successful coast-to-coast driverless truck journey in the USA [7]. Our previous research regarding ubiquitous [8,9], Internet of Things [1,10], and general context-aware software systems [11–13] revealed some gaps and the need for software technologies to support the engineering of CSS [10], mainly when their non-functional requirements include

* Corresponding author.
E-mail address: santiago.matalonga@uws.ac.uk (S. Matalonga).

https://doi.org/10.1016/j.infsof.2022.106937
Received 27 November 2021; Received in revised form 2 May 2022; Accepted 3 May 2022
Available online 6 May 2022

context-awareness.

The expectation of CASS is well deserved, as such software systems are bound to enter several aspects of our daily lives. There are clear indications that CASS present challenges to current engineering technologies [1–3]. Their testing exposes the limitations of available software testing practices and technologies. Context-awareness amplifies the software testing conundrum between coverage and effort [14], as context variation greatly increases the test input space. CASS are currently well accepted and deployed in safety-critical underregulated domains (like Autonomous vehicles), presenting new risk sources for their users when not adequately tested.

Not considering the nature of context during testing and quality assurance costs lives. Examples are already in the news, like the Boeing 737 Max [15] and the Chevrolet Onix's recall in Brazil [16]. In the Boeing 737 Max case, there was an omission to consider all design changes to the airplane aerodynamics when updating a software component designed to assist pilots in potentially dangerous situations. This omission led to a scenario where the software component enhanced the danger due to a faulty context interpretation. Such a scenario was (apparently) never tested before deployment. In the Chevrolet Onix case, the car sensors were not prepared to detect the exceptional heatwave in Rio the Janeiro, leading to situations where the car batteries ignited.

This work is motivated by the assumptions that CASS are becoming mainstream, they are being introduced to the general public, and - considering previous works (including ours [11–14] and from other groups [2] or [15]) - there is little evidence on the testing of CASS in non-academic settings.

This paper presents a Rapid Review (RR) [17] to understand how the researchers deal with the variation of context when testing CASS developed in non-academic settings. Given our work assumptions, knowledge of the literature in the field, and practical experience in testing CASS, we claim that RR is a suitable method to timely achieve our research objective. Indeed, this RR aims to find evidence from the literature in a timely and resource-efficient manner by using methods to accelerate or streamline traditional systematic review processes in rapidly changing fields [18].

Our results show that current research initiatives involving non-academic applications aim to generate or improve test suites that can deal with the variation of context and the sheer volume of test input possibilities. To achieve this, they mostly rely on modeling the dynamic behavior of the systems. They also need to rely on increasing computing resources to generate test inputs. Furthermore, we found no evidence of research results to manage context variation throughout the testing lifecycle process. In addition to this, all identified solutions are domain-specific and are not ready for widespread transfer. It means that software testers looking to test CASS need to take on these approaches and re-engineer -or tailor - the solutions to their working domains, which comes at a considerable cost.

The remainder of this paper is organized to convey the research method and results. First, we ground the terminology used through this work in Section 2. Section 3 aims to relate state of the art in testing CASS. Details of the research method design and its execution are provided in Sections 4 and 5, respectively. Data analysis is presented in Section 6. In Section 7, we present the results of this RR and frame them in the context of the state-of-the-art and the threats to the validity of this research, and propose some lessons learned. Future research directions are presented in Section 8, and empirical suggestions for testing CASS are reported in Section 9. Finally, Section 10 presents our conclusions.

2. Background concepts

This section conveys the background concepts that make the construct of this Rapid Review.

2.1. Context and Context-awareness

This work draws the definition of context by Abowd et al. [19], where context is any information that may be used to characterize an entity's situation (logical and physical objects present in the system*'s environment) and the relations relevant to the actor-computer interaction between actors and computers.* As a result, *context-awareness is a dynamic property of a software system that can evolutionarily affect its overall behavior in the interaction between actors and computers*, as defined in de Sousa et al. [11]. Therefore, context-aware software systems can identify changes in the logical or physical environment (i.e., context) and adapt their behavior to serve the actor better. In the relationship between the CASS and the context itself lies the problem with their testing. When the context variations are considered, the input space for stimulating the CASS grows beyond the current technology's handling capacity. The most used strategy to deal with the context and its variation is to model the system and exploit computing resources in simulation. In Matalonga et al. [12], we argued, and so have others [20–22], that the actual approaches for testing CASS are not prepared yet to deal with all the possible variations of the software system's context.

2.2. Software Testing

Software Testing is a systematic process for revealing failures and, therefore, indicating the presence of faults in software. This activity is important for developing high-quality software systems by assessing their functional and non-functional requirements [23].

The ISO/IEC/IEEE 29119-1-2013 [24] strictly relates the non-functional requirements to the quality characteristics of a software product. These software quality characteristics are defined in the ISO/IEC 25010:2011 [25] (a.k.a. SQuaRE), which proposes a model categorizing software product quality properties into eight characteristics: functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability. The ISO/-IEC/IEEE 29119-1 classifies the requirements into two main categories, i.e., FRs and NFRs. FRs align with the functional suitability quality characteristic outlined in ISO/IEC 25010, whereas the NFRs are linked to the remaining seven quality characteristics (a.k.a. quality attributes) outlined in SQuaRE. Henceforth, we will use the terms non-functional requirement, quality attributes, and quality characteristic without distinctions.

In this paper, to harmonize the terminology, we draw from the terms used in ISO/IEC/IEEE 29119-1-2013 [24]. In particular, we will use the test processes and test techniques for analyzing the data in this research work. According to this standard, the testing processes can be grouped into *Organizational Test, Test Management, and Dynamic Test Processes*. The former deals with organizational policies for testing and is out of this research's scope. The second refers to *"processes covering the management of testing for a whole test project or any test phase (e.g., system testing) or test type (e.g., performance testing) within a test project (e.g., project test management, system test management, performance test management)."* They rely on the execution of subprocesses for (1) *planning* the resources needed to execute the test activities, (2) *monitoring* and *controlling* the execution of the planned activities, and (3) *completing* the test process that is reached when the agreement that the testing activities are complete has been obtained. The latter *"are used to carry out dynamic testing within a particular phase of testing (e.g., unit, integration, system, and acceptance) or type of testing (e.g., performance testing, security testing, usability testing)."* They consist of four subprocesses.

- *Test Design & Implementation Process*: it is performed to derive test cases and test procedures by exploiting combinations of testing techniques.
- *Test Environment Set-Up & Maintenance Process*: it is used to build and maintain the running environment for executing the tests.

- *Test Execution Process*: it executes, in the test environment, the test procedures previously implemented.
- *Test Incident Reporting Process* reports test incidents such as anomalies, bugs, defects, errors, and issues found during the test execution process.

In this work, we will focus on dynamic testing techniques. In this paper, we will use the terms testing technique and dynamic testing technique without distinctions. According to the ISO/IEC/IEEE 29119-2-2013 [26], the main goal of dynamic testing is to derive test cases to be dynamically executed on a test item that runs in a testing environment. In practice, software testers usually apply one or more test design techniques for deriving **test cases** and procedures with the main goal of achieving a given test completion criteria, typically described in terms of test coverage measures [26], and so, to reveal as many failures as possible [27].

A test case is defined in [24] as the lowest *testing scope* of test documentation. Test cases define the "set of preconditions, inputs and expected results, developed to drive the execution of a test item" [24]. Test suites are defined as a "set of test cases" [24], which are grouped to be executed to assess that the test item behaves as expected in specific scenarios.

2.3. Non-academic setting

The term non-academic setting is used throughout the text to convey the scope and interest of this RR. By using "non-academic setting," we refer to software systems developed for industrial or commercial purposes. We look for evidence about testing processes adopted for these systems that have been presented in peer-reviewed white literature.

3. Related works

Secondary studies on CASS testing are new in the software engineering community. To the best of our knowledge, the first publication dates to 2016. In the following, we discuss (1) works focusing on specific CASS application domains, (2) studies addressing the problem from a general point of view, and (3) the novelties of this RR concerning published secondary studies.

3.1. Secondary studies on CASS testing for specific application domains

We analyzed six secondary studies focusing on specific application domains. Costa et al. [28] surveyed 13 primary studies to understand the strategies adopted for testing smart cities applications and to discuss the difficulties identified by the developers for testing this type of context-aware software system. Schmidtke [29] presented a classification of different approaches to testing intelligent sensor actuator systems (ISAS), with a special focus on autonomous vehicles. The author discussed the verification of algorithms generated by machine learning and reasoning. Schmidtke also showed that while the general verification of autonomous vehicles is a highly complex problem, it could become solvable piecewise if it can be broken down spatiotemporally. In addition, Schmidtke argued that verification of ISAS is a solvable problem if it is not seen purely as a computer engineering problem but also as a civil engineering problem so that it can be solved with a novel type of trustworthy human-like AI systems. Almeida et al. [30] and Luo et al. [22] presented systematic mapping and a survey in context-aware mobile applications testing. Almeida et al. [30] analyzed 68 studies to summarize the current state of the art concerning test automation tools for Android context-aware applications. The authors identified five tools for testing context-aware Android apps and five tools supporting context-aware testing. These tools were compared to understand which context data are supported and how and if these tools can test context variations.

Moreover, the study pointed out that the main challenges in mobile context-aware testing are mainly related to (1) the wide diversity and heterogeneity of context data types and context variations and (2) the nature of the context that constantly varies asynchronously. Luo et al. [22] gave an overview of the state-of-the-art context simulation methods for testing mobile context-aware applications. The work also discussed how each identified method could be implemented and deployed by testing tool developers and mobile application testers. Moreover, the authors summarized the main five challenges in context simulation. These are: to support the simulation in the early stages of the testing, to have high-fidelity emulators, to provide sufficient coverage of heterogeneous contextual data, to achieve the context simulation for multi-device scenarios, and to improve the support of automation. Siqueira et al. [31] extended a previous work of Ferrari et al. [32] with the main goal of characterizing the challenges that are faced during the testing activity of adaptive software systems (AdSs). Authors selected and analyzed 25 papers and classified 34 specific testing challenges for adaptive systems and inferred relationships among them through the definition of 12 generic challenges. Moreover, the authors also associated these challenges with the characteristics of adaptive software systems, such as the elements receiving environment data and elements dealing with system adaptations.

3.2. Secondary studies on CASS testing in general

We selected four works addressing the problem from a general point of view. Siqueira et al. presented two secondary studies addressing at the same time the problem of CASS testing and AdSs testing [21,33]. Siqueira et al. [33] showed the results of a Systematic Literature Review (SLR) whose main goal was to characterize fault types for AdSs and CASS. To reach this goal, the authors analyzed (1) eleven primary studies addressing fault types, (2) seven additional works investigating fault-based testing for AdSs and CASS, and (3) existing code snippets of real projects. Moreover, the authors developed new code snippets to illustrate how the fault types may occur. As a result, 26 specific fault types and a summary of 6 fault type categories for AdSs and CASS were presented. In addition, the authors discussed relationships between the fault types with current fault-based testing approaches. Siqueira et al. [21] presented an SLR and a thematic analysis of studies to update existing reviews of the same authors. The SLR selected 102 studies, and their analysis allowed us to characterize the testing approaches by grouping techniques for AdSs and CASS. The authors also presented 13 types of challenges grouped by generic and specific challenges. Matalonga et al. [13] selected and analyzed 12 primary studies in a quasi-systematic literature review (qSLR) to characterize the methods adopted for CASS testing and understand their effectiveness. As for the results, the authors observed (1) the lack of consensus between the terms and naming conventions defined in the ISO/IEC 25010:2011 international standard w.r.t. the descriptions of the techniques used by the primary sources and (2) that only two of the twelve identified sources dealt with the effectiveness evaluation of the proposed testing approach. Moreover, the authors pointed out that (1) there is a lack of consensus on what the primary sources described as context, and (2) none of the selected primary sources considered the variation of context during the test process execution since the proposed testing approaches adapted traditional non-context-aware methods to design the test cases to be executed for testing CASS. De Sousa [11] published a qSLR that surveyed 17 primary studies to characterize the CASS testing approaches from different points of view, i.e., the methods used to design test cases, the quality characteristics addressed by the test cases, the coverage criteria used to evaluate the test cases adequacy, and the influence of the context in test cases design.

3.3. Novelties of the RR concerning the analyzed literature

All the analyzed secondary studies shared the common goal to understand the challenges and the difficulties when testing CASS. The most

common of these challenges are: (1) how to deal with the uncertainty introduced by the context, (2) how to accommodate the variation of the context during the execution of the tests, (3) how to deal with the huge combination of possible inputs and outputs of CASS, and (4) how to simulate realistic system execution environment and workload during the testing.

This RR complements the aforementioned secondary studies since none focused on how researchers deal with the context variations when they perform testing processes for CASS developed in non-academic settings. Moreover, the RR also extends the entire set of the papers selected by the secondary studies since it considers a different research goal.

4. Research method – rapid review

This section summarizes the Rapid Review research protocol we have employed as the research method to guide our work. A Rapid Review (RR) aims to provide evidence on a problem more quickly and resource-efficiently than a Systematic Literature Review [17]. This research effort compares the advances in testing CASS since 2017 when de Sousa et al. [11] and Matalonga et al. [13] were published. These systematic reviews uncovered evidence of a lack of testing techniques regarding the full validation of context-awareness. Therefore, we have developed the RR protocol with the following three assumptions:

Assumption 1. CASS have spread and are more pervasive than they were three to five years ago.

Assumption 2. The software engineering and software testing communities have had time to adopt (or develop new) techniques to deal with the context and effects of testing software systems.

Assumption 3. Academics have been able to work with non-academic environments to transfer or study the approaches used to test CASS.

4.1. Definition of goal and research questions

This RR aims to understand how non-academic software projects deal with context variation when testing Context-Aware Software Systems. In addition, this RR aims to characterize the software technologies that enable us to consider the context variation in the CASS testing processes adopted outside academia. More precisely, we are interested in distinguishing whether (and how) the non-academic environments consider the context in testing, understanding whether the context is instantiated and varied during the test cases and whether the software system under test interacts at runtime with the running context.

To reach this goal, we defined the following four research questions:

RQ1 Which software technologies support the **test management processes** dealing with the context for CASS?

Rationale: To understand how the context and its variations are being considered when managing testing activities. Under our work assumptions, software teams dealing with the context-aware CASS should have many test suites and cases whose execution they must properly manage.

RQ1 Which are the software technologies supporting the **dynamic testing processes** dealing with the context for CASS?

Rationale: To understand the support for executing CASS testing. We want to understand the given support to software practitioners regarding the context and its variation during the execution of CASS testing.

RQ1 What are the issues or limitations of the observed solutions for testing CASS?

Rationale. To understand the extent to which CASS are being tested and if the experiences and lessons learned can be captured for other testers looking to test CASS. When looking at solutions proposed in the literature (see Section 3), it is clear that CASS testing has many challenges for current engineering practices and solutions come with compromises.

RQ1 How mature are the identified solutions for widespread adoption?

Rationale: To understand the effort that a software tester looking for testing CASS would be required to invest in adopting one of the proposed solutions. To the best of our knowledge, all solutions observed in the literature (see Section 3) are domain-specific and would require extensive re-engineering to be adopted for other use cases or application domains.

4.2. The primary studies selection process

This section provides an overview of the process we followed in selecting primary studies in this study. As Figure 1 shows, the process relies on executing three steps, i.e., Automated Search Strategy, Primary studies selection, and Snowballing search strategy. Executions of this process are presented in Section 5.

The selection process (see Figure 1) was decomposed into the activities described in this section. The JabRef Tool[1] has been used to manage and support the selection procedure. In each activity, we voted for the source's inclusion and discussed the motivations for conflicting votes (see Section 5.1).

4.2.1. Automated search strategy and removal of duplicates

Scopus was the main search engine, and the ACM Digital Library was the secondary. We applied our search string and selection procedures in SCOPUS and repeated the task in ACM Digital Library. Based on this, the search string was adapted to fulfill the requirements of both search engines. The variations are built on a canonical search string developed following the PICOC method with five levels of filtering (see Table 1). Duplicated papers returned by both search engines were removed.

4.2.2. Application of the inclusion/exclusion criteria

Eight inclusion criteria (IC) and three Exclusion Criteria (EC) were applied to include or discard the primary studies returned by Automated and Snowballing Search Strategies.

To be included in our study, the study must accomplish all the following IC.

IC1. the paper must be in the context of **software engineering**;
IC2. the paper must be in the context of **context-aware software systems**;
IC3. the paper must be peer-reviewed;
IC4. the paper must report a **primary study**;
IC5. the paper should be conducted within a **non-academic setting**;
IC6. the paper must report an **evidence-based study** grounded in empirical methods (e.g., interviews, surveys, case studies, formal experiments, and others);
IC7. the paper must provide data to **answer** at least one of the RR research questions by showing that dealing with context during testing was a concern;
IC8. the paper must be written in the **English language**.

To be excluded from our RR, a work must satisfy at least one of the EC summarized in the following:

[1] http://www.jabref.org/

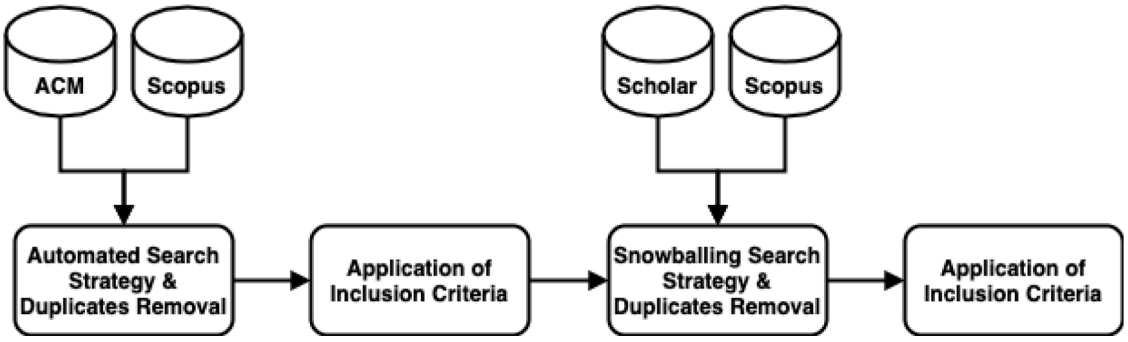


Fig. 1. Primary studies selection process

Table 1
PICOC-based search string

PICOC	Main Term	Synonyms
Population	Contemporary Software Systems	"ambient intelligence" OR "assisted living" OR "multiagent systems" OR "systems of systems" OR "internet of things" OR "cyber physical systems" OR "autonomous systems" OR "autonomic computing" OR "multi-agent systems" OR "pervasive computing" OR "mobile computing" OR "distributed systems" OR "cooperative robotics" OR "adaptive systems" OR "industry 4.0" OR "fourth industrial revolution" OR "web of things" OR "internet of everything" OR "contemporary software systems" OR "smart manufacturing" OR digitalization OR digitization OR "digital transformation" OR "smart cit*" OR "smart building" OR "smart health" OR "smart environment"
Intervention	Software Testing	"test* management" OR "test* planning" OR "test* monitoring" OR "test* control" OR "test* completion" OR "test* design" OR "test* type" OR "test* implementation" OR "test* environment" OR "test* execution" OR "test* reporting" OR "software test*" OR "software validation" OR "software verification"
Comparison	Not Applied	
Outcome	Software Testing Technologies	"test* management" OR "test* planning" OR "test* monitoring" OR "test* control" OR "test* completion" OR "test* design" OR "test* type" OR "test* implementation" OR "test* environment" OR "test* execution" OR "test* reporting" OR "software test*" OR "software validation" OR "software verification"
Context	Context Variation	"variation" OR "context" OR "context awareness"

EC1. The paper has similar contents w.r.t. other studies from the same authors and was published later in different venues;
EC2. the paper is an earlier version of a more recent or complete version also identified;
EC3. the paper has not been published in Computer Science and Computer Engineering.

We developed a shared understanding among the team regarding the concepts included in these criteria during the protocol execution. We exemplify this further in Section 5.1.

4.2.3. Snowballing search strategy

Forward and backward snowballing [34] was used as a complementary search strategy to mitigate the threat of missing literature. We seeded the snowballing process with the selected sources after applying the selection procedure (see Section 4.2.2) to the sources identified from the automated search process. Regarding the initial seeds, we verified that these were suitable to conduct a snowballing process by reviewing and evaluating the five-point soft test proposed by the snowballing guidelines by Wohlin [34]. Therefore, we decided to include all sources, as no research group or author seemed to gravitate toward the initial seeds. Furthermore, in 2020 (see Section 5), the seed sources were complemented with sources suggested by colleagues with expertise in software engineering and CASS software testing, who reviewed the protocol and the first round of results.

4.3. Data extraction process

We designed a data extraction form to extract the information needed to answer the research questions. This form's pertinence became clear since it fine-combed the sources for their suitability to answer the research questions. The following fields were included in the data extraction form:

Field 1. Abstract: it contains the abstract of the analyzed primary study.
Field 2. Description: it reports a high-level summary of the proposed testing technique in the author's words.
Field 3. Study type: it is a description of the type of experimental study that was described in the source.
Field 4. Application domain: it describes the non-academic setting in which the study took place.
Field 5. Type of software system: it reports the possible type of software systems reported in the source.
Field 6. RQ1: it reports sentences for answering research question 1.
Field 7. RQ2: it reports sentences for answering research question 2.
Field 8. RQ3: it reports sentences for answering research question 3.
Field 9. RQ4: it reports sentences for answering research question 4.

Excerpts that directly answered the research questions were extracted in these fields. Table 2 presents the data extraction form for Qin et al. [35] (S10).

5. Execution of the protocol

The fast feedback nature of the RR research process allowed us to execute three iterations of the research protocol defined in Section 4. Figure 2 shows in detail the execution of the primary studies selection process we performed in the RR.

The swimlanes in Figure 2 represent the three executions of the process we performed in 2019, 2020 and 2022 respectively. Within each swimlane, the same steps of the RR process presented in Figure 1 are depicted, thus indicating, by means of full arrows, the flow of execution of such steps during the process. The document icons, connected through dotted arrows to the steps, represent the set of primary studies produced as output or needed as input by each step. The numerical values reported in the document icons represent the total number of

Table 2
Example of Data Extraction Form regarding the work of Qin et al. [35]

Field	Example Extraction excerpt
Abstract	<A full text of the abstract was extracted. We mainly used this as a reference for discussions. For brevity, we do not include it here>
Description	"in this article, we present an approach, named context-based multiinvariant detection (CoMID), to automatically generating invariants for specifying developers' implicit assumptions and checking these invariants for detecting when a cyber-physical program has entered an abnormal state at its runtime. CoMID addresses the preceding challenges with its two techniques: context-based trace grouping and multiinvariant detection." [35]
Study type	"We present the evaluation of our CoMID approach, including comparing it with two existing approaches. We select three real-world cyber-physical programs, namely, NAO robot.., and six-rotor UAV …. as the evaluation subjects." [35]
Application Domain	Cyber-physical program
Type of software system	<Authors make no explicit statement as to this field. For our purposes, we can abstract from the 'Study type' field and leave this field empty>
Answer to research questions	
RQ1	<No data extracted for this source>
RQ2	"CoMID works in four steps, which are as follows: 1) it first executes program P in the environment E to collect safe execution traces, i.e., no failure condition triggered (Step 1: trace collection); … We implement CoMID as a prototype tool in Java 8" [35]
RQ3	"CoMID still has room for improvement. For example, it currently records the values of program variables at entry and exit points of all executed methods, and uses these variable values to generate invariants. Monitoring all executed methods greatly increases the time overhead of CoMID, and makes it less effective when applied to a time-critical cyber-physical program (e.g., a program whose iteration length is less than 100 ms, as discussed in Section IV-C)." [35]
RQ4	<No data extracted from this source regarding RQ4>

primary studies in each set.

The first execution of the protocol was executed in 2019. In this iteration 8 papers were filtered in over the 611 distinct studies returned by the search engines, and a final set of 8 primary studies were selected from the 43 distinct papers returned by snowballing. The results of the analysis of these papers were published in a technical report [36].

Regarding the 2020 iteration, the actor icon represents three researchers from academia with more than ten years of experience in CASS software testing who suggested six primary studies after reviewing the technical report we published at the end of the previous iteration. In this iteration no one of the 16 distinct papers, gathered from the search engines, passed our IC and EC. We considered the six papers suggested by the researchers as candidate sources, but, after applying the IC and EC, none of them were selected. Furthermore, we included these studies as additional input for the snowballing process to further mitigate the risk of missing literature. At the end of this iteration a final set of 11 primary studies were selected over the 72 papers returned by snowballing.

In January 2022 we revisited again the protocol execution, and no new sources were included after we applied the IC and EC to the 41 distinct primary studies returned by the search engines. Finally, from the 110 works collected by snowballing, the same 11 works already selected in 2020 have been included in the final set of primary studies.

It is worth observing that in the snowballing search strategies performed in 2020 and 2022, we also added, as input, the primary studies already selected in the previous stages.

5.1. Achieving consistency with the inclusion criteria

We invested much effort in establishing the criteria' consistency among the different researchers when applying the inclusion criteria at the different RR inclusion activities. In all such activities, each primary study was analyzed by two researchers that voted the paper as "*yes,*" "*no,*" or "*doubt.*" Studies that obtained two "*yes*" were included. On the contrary, studies with two "*no*" were discarded, and researchers discussed the remaining ones to reach a common consensus.

We present a few examples of the research team's discussions to convey our rationale regarding the inclusion/exclusion criteria.

Regarding IC2, we extensively discussed what qualifies a context-aware software system. Upon reflection, most of these discussions revolved around the complexity of the software system under test. This decision draws from Lewis's conceptual pragmatism [37]. Thereby, we probed the test item described in the candidate sources to determine whether the added complexities needed for considering the context and its variation were worth making to test such software. This exercise led to the rejection of several sources. Each proposal was reviewed to verify that the software under test was a CSS that exhibits context-aware characteristics (even within a non-academic setting) to merit the concepts' application. For instance, in Xu et al. [38], the proposal considers context-awareness. It conveys how the software under test is expected to be deployed in an industrial setting. However, the complexities of the defined setting for the evaluation are too simple for this RR (column Test Item in Table 4 synthesizes this process for the selected studies). Therefore, we make explicit that we are not making a judgment call on the pertinence of that system to the research goals expressed in [38]. Instead, we are making a judgment call on the pertinence of that software under test to our study's goals. Furthermore, we note to the reader that this judgment was made after going through the data extraction process for this source (conveying how thorough we were of the inclusion/exclusion decision taken).

We want to clarify how we have interpreted that the primary study was conducted within a *non-academic setting* regarding IC5. This RR aims to identify evidence of testing CASS outside academia. Therefore, the source's criterion clearly states that the test item was intended to be deployed in the production environment. We wanted to weed out test items developed for the research, even if these were based on real-life applications and/or with an industrial sponsor's support. For instance, Rosenthal and Lewis [39] were evaluated in this RR's process, passing the inclusion criteria until the Data Extraction process. At this point, we noticed that the authors commented that they "implemented the SVS (Smart Vacuum System) application as a completely autonomous robot that receives input from sensors and responds accordingly." [39]. As a result, we agreed that these implementation types could not be judged to belong to a non-academic setting. There is an explicit statement that the researchers implemented the test item. Interestingly, several other sources suffered the same analysis as [39]. For instance, [40,41] contain similar statements that can only be identified in a thorough reading (or during the Data Extraction process). Qin et al. [35] present an interesting case on the other end. The proposal is relevant to this research, but it became evident that only the second use case described in the paper presents evidence of an industrial setting during the data extraction. Therefore, only evidence from the second use case was included in the analysis of this research.

Regarding IC7, we settled into three conditions to appraise this criterion: the paper must address the phenomenon of interest, the test item must be context-aware, and the proposal must show evidence of having a CASS software under test. For instance, for [38,42], we judged that there was no evidence of the test cases' execution upon finalizing the data extraction of those references.

6. Data analysis

In this section, we report a detailed analysis of the extracted data.

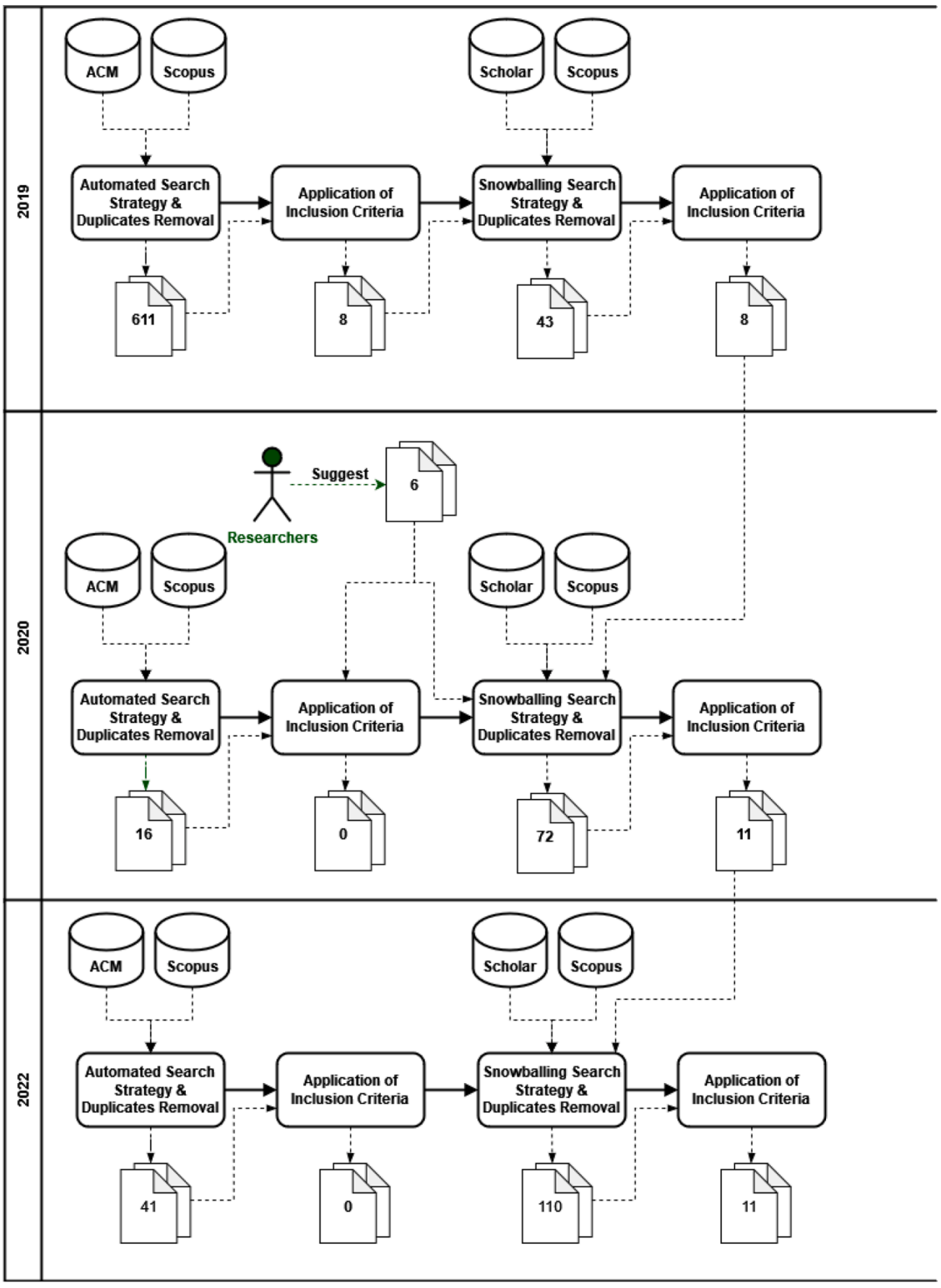


Fig. 2. Primary studies selection process execution

First, we briefly describe the research reported in the selected papers. Then, we characterize these testing techniques and answer the research questions we proposed based on our evidence.

6.1. Overview of the included sources

This section provides an overview of the sources selected as this RR. Table 3 presents the bibliographic information of the final sources, reporting the number of publications by venue type and year. Next, we present a summary of the selected sources. We added this summary to provide the readers with an overall understanding of the selected sources.

S1. Ma et al. [43] present a model-based approach for self-healing software systems. Their approach includes a modeling framework and an execution environment. Their approach focuses on dealing

Table 3
The final set of selected papers in the Rapid Review

Code	Full Reference	Venue Type	Year
S1	Ma et al. *Modeling foundations for executable model-based testing of self-healing cyber-physical systems* [43]	Journal	2019
S2	Arrieta et al. *Automatic generation of test system instances for configurable cyber-physical systems* [44]	Journal	2017
S3	Shin et al. *Test case prioritization for acceptance testing of cyber-physical systems: a multi-objective search-based approach* [45]	Conference	2018
S4	Shin et al. *Uncertainty-aware specification and analysis for hardware-in-the-loop testing of cyber-physical systems* [46]	Journal	2021
S5	Lahami et al. *Safe and efficient runtime testing framework applied in dynamic and distributed systems* [41]	Journal	2016
S6	Fröhlich et al. *Testing Safety Properties of Cyber-Physical Systems with Non-Intrusive Fault Injection – An Industrial Case Study* [47]	Conference	2016
S7	Abdessalem et al. *Testing autonomous cars for feature interaction failures using many-objective search* [48]	Conference	2018
S8	Abdessalem et al. *Testing advanced driver assistance systems using multi-objective search and neural networks* [49]	Conference	2016
S9	Abdessalem et al. *Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms* [50]	Conference	2018
S10	Qin et al. *CoMID: Context-Based Multiinvariant Detection for Monitoring Cyber-Physical Software* [35]	Journal	2020
S11	Qin et al. *SIT: Sampling-based interactive testing for self-adaptive apps* [51]	Journal	2016

with the uncertainties bright to the software systems by perceiving the context. The authors evaluate their approach on a computer-based simulation, in which their approach is applied to nine self-healing software systems. Test cases are designed in the modeling framework and executed simultaneously in the model and the system.

S2. Arieta et al. [44] propose testing individual configurations from software product lines of cyber-physical systems. The individual configurations are selected by employing software-based simulation. Their approach models the Software product line of the CPS, and simulation is used to evaluate the CPS's interactions and the context dynamically. The authors evaluate their approach in a computer-simulated environment. Their case study subject is a model of an Unmanned Aerial Autonomous Vehicle. Test cases are generated based on an initial set of test cases evaluated against the SPL-CPS modeled variability points.

S3. Shin et al. [45] propose a search-based approach for prioritizing acceptance test cases for CPS. Interestingly, while context variation is not explicitly addressed in this proposal, the authors consider the problems brought by uncertainties in the environment's perception. That consideration is reflected in the case study. The presented case study involves a simulation environment where software for a satellite system is evaluated. Test cases are defined in a domain-specific language, and the author's search-based proposal optimizes the selection and execution of the test cases.

S4. Shin et al. [46] present an approach for specifying and analyzing hardware-in-the-loop test cases for cyber-physical systems. A textual language named Hardware-In-the-loop TEst Case Specification (HITECS) is defined using the UML profile mechanism to apply these methods. Their approach deals with uncertainties that context can realize in the software under the test's environment. They present a case study where the test item is a satellite system to evaluate their approach. Their main aim with the case study is to evaluate their test cases' quality concerning the approach capacity to estimate the test cases' execution time. Their case study environment simulates an in-orbit test setup that includes all hardware and communication protocols.

S5. Lahami et al. [41] present a runtime approach to detect and execute test cases affected by context changes. They evaluate their approach in a case study involving a healthcare application deployed within a controlled environment in their lab. The authors introduce changes to the system's components under test to observe how the approach senses the context and selects test cases from a set of initially defined test cases.

S6. Fröhlich et al. [47] present a method for evaluating the safety of critical context-awareness systems' failures. Their underlying assumption is the combination of input values from context sensing components of the software system. Their approach consists of injecting faults and observing how safely the CASS fails. They evaluate their approach in a steering system of an autonomous car within a simulated environment custom-built to their proposal.

S7. Abdessalem et al. [48] deal with conflicting feature interactions from feature composition of self-adaptive behavior driven by context. Context is considered in this proposal as it is the driver for triggering the adaptations. In this proposal, the authors reframe the problem of finding undesired feature interaction as a search problem. They evaluate their approach using three systems from a self-driving car (automated emergency braking, adaptive cruise control (ACC), and traffic sign recognition). Executable models of these systems are simulated as the search-based approach identifies conflicting feature interactions.

S8. Abdessalem et al. [49] propose an automated technique to test complex Advanced Driver Assistance Systems (ADAS) using physics-based executable models of the system and its environment. As an evaluation, they presented an exploratory case study using an ADAS called Pedestrian Detection Vision-based (PeVi) system. The PeVi was created to help drivers detect pedestrians' proximity (either human or animal) during low visibility situations. The case study focused on identifying high-risk test scenarios (the ones that are more likely to reveal critical failures) using a multi-objective search. The context was captured as a domain model, specifying a restricted simulation environment and serving as input data (which can be either static or dynamic properties) to test scenarios. Since testing PeVi with real hardware and environment would be dangerous, time-consuming, and costly, they used physics-based simulation platforms as a testing environment.

S9. Abdessalem et al. 2018 [50] focus on simulation-based testing of vision-based control systems from the automotive domain (ADAS). A domain model is used to capture the test input space and output. The input variables were classified into two categories: Static input variables (the values still fixed during the entire ADAS simulation) and Dynamic objects (indicating objects that change their position during the simulation). As an evaluation, they presented an exploratory case study of an Automated Emergency Braking (AEB) system from the automotive domain. The objective was to evaluate the ability of a search-based testing algorithm called NSGAIIDT to investigate critical regions in ADAS input spaces to identify critical test scenarios. They used a commercial ADAS simulator called PreScan simulator as a test environment.

S10. Qin et al. [35] present an approach for dealing with context input states that can lead to failures. In their approach, context varies as no restrictions on context variables' values are imposed. However, they demonstrate this approach offline, as execution captures of the system under test are performed in the field for later post-processing in the lab to evaluate their approach. As mentioned in Section 5.1, we are interested in evaluating their approach to the Unmanned Autonomous Vehicle in this paper. Their main idea is to define rules for context values that, if broken, would lead to failure (they call these invariants), then observe the software system's behavior to

abstract these invariants states and execute them in a model of the system to observe the behavior.

S11. Qin et al. [51] present an approach for selecting test cases for self-adaptive context-aware software systems. Context variation is considered in terms of uncertainty in the adaptation rules and uncertainty in the sensor's capacity to measure the environment accurately. They evaluate their approach with three case studies to implement it in a simulated environment. Their approach selects test cases by sampling input space parameters after sorting them into different categories according to the abovementioned abstraction.

The following section characterizes these technologies and describes how they are applied in non-academic settings to test context-aware CSS.

6.2. Analysis of the identified papers

In the selected primary studies, we did not find any evidence regarding software technologies supporting the management of the test activities dealing with the context for CASS, RQ1.

In the following, we describe the analysis of the extracted data that allowed (1) to characterize the technologies for supporting the testing execution (RQ2), (2) to understand the limitations of the approaches as described by the authors (RQ3), and (3) to understand whether the identified solutions are mature enough for their widespread adoption (RQ4). The discussion is supported by tables summarizing the analysis and focusing on the evidence. In addition, in Annex 1, we reported additional tables having deeper details about the evidence we extracted from the analyzed papers to answer the research questions.

6.2.1. Analysis of the software technologies supporting the execution of test activities dealing with the context.

To describe the software technologies supporting the execution of CASS testing processes, we analyzed the extracted data to understand, for each proposed approach, the following characteristics:

1 the test items, i.e., the types of CASS under test (see Table 4),
2 the testing scope of the proposal (either test case or test suite level), and the addressed quality attributes (see Table 5),
3 the context variation, i.e., how the context variation has been considered (see Table 6),
4 characteristics of the testing environment adopted for executing the proposed testing process (see Table 7),
5 the adopted dynamic testing technique (see Table 8).

An interesting variety of test items were used to evaluate the proposed testing techniques regarding the observed test items. Table 4 summarizes the evidence on the test items under investigation in the selected sources.

Table 5 presents that the test techniques for testing CASS are focused on either defining test cases or improving test suites. When defining test cases, the intent is to assure functionality, evaluate the reliability and verify that when the systems fail, it does so in a controlled manner. When looking at test suites, the sources look at improving a specific quality attribute in the set of existing test cases in the test suite (reliability or performance). When looking at the quality attributes, it can be observed that many of them are still to be explored. It can be argued that those present in Table 5 are related to the challenge of dealing with the test input space brought about by context variation.

Table 6 presents the three strategies we have observed in the sources to deal with context variation. One strategy is to model the contextual variables and their influence on the proposal. Another strategy is to consider that the sources of contextual variation are related to the intrinsic errors in the environment measurements through the sensors (referred to as environmental uncertainties). While in S10, execution of the test item in its operational environment was used to capture the contextual variables, and then these measurements were used to validate the proposal.

Table 7 looks at the technology used to develop the proposed solutions and the test items' environments. Table 6 allows the reader to observe the relationship between the technologies and the environment used during the empirical evaluations. We divided the environments into Simulation, where empirical evaluations were carried out within computational environments, and Physical-world, where empirical evaluations were carried out involving Physical-world elements (such as sensors field trials). For instance, UML-based models were the most frequent technology, and it has been used in both simulation and real-like environments. Nevertheless, Matlab/Simulink has been used in three sources, but we did not uncover evidence of Matlab/Simulink models being used in real-like environments to test CASS. Another evidence we collected regards the adoption of Physical-world environments for implementing Hardware in the Loop (HIL) techniques, as we observed in S3 and S4. Indeed, such an environment is needed to provide an effective platform to emulate in real-time the context variation. A HIL

Table 4
Test item types in the selected sources

Test item	Sources
Self-driving automotive system	S7, S8, S9
Satellite Cyber-physical system	S3, S4
Unmanned Aerial Vehicles	S2, S10
Self-healing cyber-physical systems	S1
Adaptable and distributed healthcare system	S5
Electric car steering cyber-physical system	S6
Self-adaptive mobile application	S11

Table 5
Scope and quality attributes under investigation in the selected sources

Testing scope of the proposal	Quality attribute	Sources
Test case	Functional suitability	S1, S2, S6, S9
	Reliability	S7, S9, S10, S11
	Safety	S6
Test suite	Reliability	S3, S5
	Performance	S4

Table 6
How sources consider context variation

Consideration of context variation	Sources
Modeled contextual variables	S2, S5, S6, S7, S8, S9
Environmental uncertainties perceived as context variation	S1, S3, S4, S11
Captured from executions in the real world	S10

Table 7
Technology and environments where testing took place in the selected sources

Technology	Environment	
	Simulation	Physical-world
Matlab/Simulink	S7, S8, S9	
UML Diagram or UML metamodels	S1, S2, S11	S4, S5
Ad-hoc Domain-specific language	S3	S3, S6
Though source code invariants		S10

Table 8
Test techniques involved in the selected sources

Test Technique	Sources
Model-based testing	S1, S2, S5, S11
Search-based testing	S3, S7, S8, S9
Formal Method	S4
Fault injection	S6
Multi-invariant-based	S10

simulation environment usually includes the electrical emulation of sensors and actuators. These electrical emulations act as the interface between the context and the system under test. Finally, ad-hoc domain-specific languages are also frequently observed, both with simulations and physical-world environments. While the technology of the Test Item probably drives the technology section, it is interesting to observe in Table 7 that those technologies with application in simulation and the real world require effort in customized development and are unlikely to be available off-the-shelf.

Finally, we looked at the test techniques used to define the test cases or suites (Table 8). Model-based testing and search-based testing are the most frequently used technique. These techniques make a good fit to deal with the volume of test cases (or test suites) needed to warrantee CASS quality.

6.2.2. Analysis of the limitations of the proposals

As shown in the Data Extraction, we collected the authors' self-reflection sections and passages regarding the potential imitations of their proposals. The common limitations emerged when analyzing the extracted sentences, and the studies where they have been presented are reported in Table 9. First, Resource-consuming indicates that the proposed testing approach consumes too many resources. Extension to other application domains' limitations abstracts the common theme that the solutions are domain-specific and require re-engineering to be adapted to other domains (this is also discussed in Section 6.2.3).

It is also interesting to observe that three sources declared that while considering the huge variability of the input domain space, their approaches presented limitations that did not allow them to fully accommodate the variation in the context.

Finally, two other sources highlighted different limitations that we thought were interesting to highlight while not common to other sources. The effort-driven approach for manually building the models indicates that the activity for building the models of the context requires the intervention of a human tester. As for Model technical limitations that simplify reality limitations, the source referred to the technical inability of the adopted framework to model contexts that are very close to reality.

These results showed evidence of the complexities of testing CASS and the inherent limitations of current engineering approaches.

6.2.3. Analysis of how the identified solutions are ready for widespread adoption

To understand how and if the proposed solutions are ready for being widely adopted, we evaluated their technology readiness levels (TRLs). TRLs provide a useful model for conveying the maturity of technologies. The classification was initially defined for the Apollo missions and has been extensively adopted in Europe [52]. The TRL levels are a discrete scale of nine levels, from Basic research (TRL1) to Operational Readiness (TRL9). We used an estimator tool developed by the University of San Diego [53]. Table 9 shows how we classified the sources into three orthogonal dimensions (Engineering cycle, Study type, and Environment control) to support an evidence-based input to the TRL estimator.

The "Engineering cycle" column is based on Wieringa et al. [54] and presents a classification for technology development projects that classifies the aim of experimental work. We used this classification to provide a uniform language to address the study's primary sources' experimental evaluations. We were able to classify the sources into two of the six phases of the engineering cycle (Problem investigation, Solution design, Solution validation, Solution selection, Solution implementation, and Implementation evaluation). These were:

- *Solution design:* the research project aims to propose an improvement to a current situation.
- *Solution validation:* the research project aim at characterizing some properties/quality attributes of a proposed solution.

The "Study type" column is based on the classification provided by Easterbrook et al. [55]. This classification indicates the investigation strategy described in the primary source. We used this classification to provide uniform language and convey the capacity to generalize the sample's research results. While the classification provided by Easterbrook et al. [55] includes five main study types (controlled experiments, case studies, survey research, ethnography, and action research), all of the selected studies in our sample were classified within the case study main category, with the possible subcategories meaning:

- *Exploratory case studies* to develop new theories or observations.
- *Confirmatory case studies* to confirm existing theories.

The column "Environment control" is based on the classification proposed in Travassos and Barros [56], indicating the degree of control that the experimenters introduce in the environment to observe the phenomenon under study. We used this classification to convey the degree of control the experimenter has in the environment where the empirical evaluation is described in the source tool place. Potential categories in this classification are:

- *In-vivo*: the empirical step was executed in the target environment (with real organizations and professional developers).
- *In-vitro*: the empirical step was executed in a controlled environment (such as a laboratory, controlled sett-up, or community of practice).
- *In-virtuo*: the empirical step was executed in semi-simulated and controlled environments with interactions between the participants and a computerized model of reality.
- *In-silico*: the empirical step was executed in a fully simulated and controlled environment (the model and the subjects are described as computer models).

We followed a Delphi-type cycle [57] to achieve consensus on our estimates for each category. First, the two lead researchers would classify the paper into the available categories and explain their positions. Then, the other authors reviewed the results.

Table 10 conveys that the proposals were classified within the design half of the engineering cycle. It means authors are still mainly concerned about the purpose fitness of the proposed solution. Furthermore, selected sources apply case study research. While this can be suitable to evaluate the proposals, it also limits the transferability to other application domains. Finally, we also note that the researchers had some degree of control over the environment (i.e., we could not classify any proposals within the "in-vivo" category). That last observation is consistent that all estimations of the TRL levels are within levels 3 to 6, showing that the identified sources are not ready for widespread adoption.

Table 9
Limitations of the proposal

Limitation	Sources
Resource-consuming	S4, S5, S8, S10
Extension to other application domains	S1, S2, S4, S5
Variability of the input domain	S1, S10, S11
Effort-driven approach for building the models manually	S2
Model technical limitations that simplify reality	S8

7. Results

This section presents the result of this RR. Frist, we present a straightforward evidence-based answer to the research questions of the RR. Next, we discuss the experimental limits of our research method and reflect on the threats to the validity of the answers. The following subsection gradually frames the answers in terms of the literature and then looks forward by summarizing the lessons learned results.

Table 10
Level estimators of testing technology readiness

ID	Engineering Cycle	Study Type	Environment Control	Estimated TRL
S1	Solution validation	Confirmatory case study	*In-vitro*	L4
S2	Solution validation	Exploratory case study	*In-vitro*	L3
S3	Solution design	Confirmatory case study	*In-virtuo*	L4
S4	Solution validation	Confirmatory case study	*In-virtuo*	L4
S5	Solution validation	Confirmatory case study	*In-vivo*	L5
S6	Solution design	Exploratory case study	*In-vitro*	L3
S7	Solution validation	Confirmatory case study	*In-virtuo*	L5-L6
S8	Solution validation	Exploratory case study	*In-silico*	L3
S9	Solution validation	Exploratory case study	*In-silico*	L3
S10	Solution validation	Exploratory case study	*In-vitro*	L4
S11	Solution design	Exploratory case study	*In-virtuo*	L5-L6

7.1. Answering the RQs

The RR uncovered evidence on test proposals for dealing with the complexity considering the context for CASS. Despite the breadth of the system types for which these proposals were identified, they share the common trait that they deal with the complexities brought by considering the context in the testing process.

RQ 1: Which software technologies support the test management processes dealing with the context for CASS?

We did not identify any solution dealing with test activities management.

Test management activities for CASS must become a research direction to address the huge volume of test cases that must be managed during the test process, as discussed in Section 8.

RQ 2: Which software technologies support the **dynamic testing processes** dealing with the context for CASS?

The reviewed solutions focus on enabling the execution (dynamic testing) of context-aware software systems. As shown in Section 6.2, technologies can be characterized from different perspectives:

Perspective 1. Interesting results were obtained regarding the testing goals. We observed that the scope of tech-niques is divided into those that look at test cases and those that look at test suites. When developing test cases, the sources intend to verify the functional suitability (S1, S2, S6, S9), the Reliability (S7, S9, S10, S11), or the Safety (S6). Another approach is that technique assumes that a test suite is already available. The proposal looks to improve Reliability by identifying the test cases in the test suite that can damage the software or the hardware (S3, S5). Alternatively, they look to improve the performance of the test suite (S4).

Perspective 2. The variation of context is mainly considered by utilizing models that can also be executed. Usually, these models are implemented in Matlab/Simulink (S7, S8, and S9), UML diagrams, or UML metamodels (S1, S2, S4, S5, S6, and S11). Ad-hoc solutions relying on a DSL (S3) and source code invariants (S10) were also proposed.

Perspective 3. As for the testing environment, we observed that it is mainly a simulation environment where the executable models can be executed with the software system under test (S1, S2, S3, S4, S7, S8, S9, and S11). Therefore, another possible solution is to build a testing environment like the real environment (S5, S6, and S10).

Perspective 4. Regarding the adopted dynamic testing techniques, we observed that model-based (S1, S2, S5, and S11) and search-based or genetic (S3, S7, S8, and S9) approaches are the most exploited. Model-based is mainly adopted for testing the software system's quality under test. In contrast, search-based or genetic improves a test suite's quality attribute or generates test scenarios. Other types of approaches have also been proposed, such as the use of formal methods (S4), fault injection (S6), and multi-invariant based (S10).

The results abstracted in *Perspective 1* open two future research directions (*Future direction 3* and *Future direction 4*) as described in Section 8. The findings expressed from *Perspectives 2 to 4* are not new in the literature. We discussed them in Section 7.3, confirming that these technologies are ready to be applied in non-academic settings. Furthermore, this was discussed in Section 7.4. In Lesson learned 1 and Lesson learned 2, we describe our understanding of the requirements that dynamic models and simulation environments must attend to support CASS testing.

RQ3. What are the issues or limitations of the observed solutions for testing CASS?

Limitations for testing CASS were presented in Section 6.2.2. We grouped the authors' self-assessments of the limitations into five groups (see Table 9). We interpret these limitations as a signal that testing CASS is still a problem that has not been solved. It is interesting to note that the issues and limitations are linked to testing CASS's challenges on current engineering solutions. While we note that these sources address these challenges, the authors' self-reflection on the issues and limitations reveals the complexities of Testing CASS. For instance, one of the main challenges for testing CASS stems from the myriad of possible input brought about by context variation (see [13,14]), yet three sources (S1, S10, and S11) present that an issue or limitation is to *consider the variability of the input domain*.

The most common issue or limitation was that the approaches consumed many computing resources (S4, S5, S8, and S10). A limitation also stems from the attempt to reproduce the contextual inputs from computerized models. However, all models represent reality, and as such is fitting that the authors are concerned with the (technical) simplifications that their model brings *into reality* (S8). This dependency on a model is also a constraint in implementing or extending *an approach into other application domains* (S1, S2, S4, and S5).

Concerning the literature, we observed that some challenges and issues were already known, but new ones have been raised for CASS testing in non-academic settings, as we discus in Section 7.3).

RQ4: How mature are the identified solutions for widespread adoption?

Observing the analysis in Section 6.2.3, we claim that none of these proposals is mature enough for widespread adoption. Only one paper was classified at TRL6, which means that the proposals can only be expected to have been demonstrated in a relevant, not completely stressful environment, and transfer to another environment can come at a high cost. We have observed that all proposals require significant investment in technological development. Furthermore, though the path set by these tools can potentially be implemented in other working domains, they are still shy of higher readiness levels where the proposed approaches can be transferred to other domains without significant re-engineering or tailoring.

The second outstanding observation is that the selected papers did not present technologies to support either Organizational Test Process or

Test Management Processes (**RQ1**). Regarding Organizational Test Process, this can be understood as the ISO/IEC/IEEE 29119-2-2013 [26] guidelines for Organizational Test Process are mainly related to organizational policies for testing. In contrast, as mentioned in our answer to **RQ1**, the lack of technologies to support the test management process suggests an unexplored research direction. Moreover, it has implications for the widespread adoption of the proposals since organizations looking to incorporate these approaches will have to consider issues like costs and deadlines (project-based elements which have been not considered in the observed sources). Furthermore, establish policies for accepting CASS deployment to the production environments.

We observe that progress has been made since the secondary studies reviewed in Section 3. In Section 7.3, we compare these results with previous secondary studies.

7.2. Threats to validity

This section discusses the threats to this research work's validity using the categories described by Wholin et al. [58].

Regarding internal validity, the threat of missing literature is common in all secondary studies. Nonetheless, we want to bring two points to the reader's attention. First, throughout Section 4, we intended to convey the thoroughness of our approach to searching the available literature. Second, the aim of Section 6.1 is to convey the criteria with which the literature was analyzed and ultimately selected. Other research works to address the challenge of testing CASS, yet they are not included in our sample as they do not meet our inclusion criteria; therefore, they are not within the scope of this study.

Another point to make is that we decided to only look at white literature. Given that the working assumption is that CASS are mainstream, it is likely that big players[2] have found ways to test these types of systems. However, it is just as revealing that it has not been contributed to the academic literature if the knowledge is available. Therefore, it should be the object of a multi-vocal study.

Regarding construct validity, the reader can question the objectivity of the RR since it is biased by our previous research. Rapid Reviews are designed to be executed by experts in the field. Therefore, though our approach toward the new evidence is objective, our previous knowledge informed our judgment. We would argue that this strengthens the results. Nonetheless, we have taken care to expose these assumptions so the reader can judge.

We stress that we have taken a specific understanding of the constructs mentioned in the inclusion criteria regarding external validity. We have fine-combed the literature and have had extensive discussions (see Section 6.1) before deciding if a source complies with our understanding of *Context Awareness, Testing, Contemporary software systems*, and *non-academic setting*. This process has strengthened the construct validity at the expense of our external validity. Therefore, further evaluation with practitioners is necessary to strengthen our findings.

We have tried to provide evidence for our research questions' answers (Section 6.2) regarding conclusion validity. The analysis section fairly portrays the uncovered evidence, which is, in turn, based on a thorough and verbatim data extraction process. Thus, we believe we have provided end-to-end traceability from the sources to the answers to the research questions. In addition, we have made a clear distinction in this paper to differentiate evidence-based answers to the research questions (Section 6.2) from our interpretation (following sections).

7.3. Comparing this Rapid Review against other secondary studies

This work is the first secondary study to understand how CASS are tested in non-academic settings**.** We found primary studies that were not already considered in previous studies. Their analysis from similar and different points of view, concerning the ones already addressed in the past, allowed us to increase the body of knowledge on CASS testing in different ways.

On the one hand, this RR showed similar results obtained in previous secondary studies, such as the limitations of the testing strategies, how the context variations are taken into account, and the types of adopted testing techniques, which are still valid when the focus moves to non-academic settings. On the other hand, we observed that new challenges arise in those more realistic settings differently from these previous works. Indeed, much more interest is given to reducing the costs of the testing processes. In some cases, test cases must not damage the expensive hardware of the testing environment. Also, for the way to characterize the context variation, we observed that not only models or formal specifications could be used. Indeed, we understood that domain languages or the preliminary data collection from the real field to infer the context variations are also adopted in real scenarios. Finally, we pointed out that, along with model-based testing and simulation-based techniques already reported in other secondary studies, search-based, genetic algorithms, and code invariant-based testing techniques are also adopted.

Another interesting result is that, in non-academic settings, the testing processes should also be executed in a testing environment that is identical, or resemble as much as possible, the execution environment. It leads the test engineers to put much effort into building a proper testing environment where test cases can be executed.

An additional improvement to the body of knowledge provided by this RR has been to understand which are the most interesting application fields for the community. In our observations, hot industrial topics (like automotive, autonomous drones, and satellite systems) have been the most addressed types of systems.

We also pointed out how mature are the proposed solutions for being adopted in production scenarios thanks to the analysis of the experimental evaluations and the estimation of the reached TRLs. Another interesting result regards the absence of evidence on how the testing processes are managed and planned.

7.4. Lessons learned

Considering the RR results, our previous results, and our experience in CASS testing, we put forward the following three main lessons we learned from this research.

7.4.1. Lesson learned 1: needs for proper models to describe the system's dynamic behavior

Testing context-awareness features of contemporary software systems require a model capable of modeling the system's dynamic behavior. A model is a reasonable representation of the system that can be used as a surrogate of the actual system to design or improve test suites. In our answer to RQ2 (Section 7.1), all sources use models to explore the system.

Our point of view is that systems models that cannot evolve with the software system are not good for modeling the dynamic nature. These models successfully design stable context states but fail to provide insight into the system's behavior during a state transition. For instance, these models cannot perceive the variation of context that has not been defined into a set of valid states.

As a result of this observation, we favor proposals modeling the systems' dynamic behavior as these are better suited to capture the dynamic nature of the context. Nonetheless, we have also observed proposals that, while developing dynamic models, these are used to fix values for context variables, thereby limiting the models' capacity to reproduce the varying nature of the context.

7.4.2. Lesson learned 2: needs for executable models

Another issue regarding models relates to the technologies used in their development. We have observed various technologies, from formal

[2] A few come quickly to mind: Tesla; Google; NASA, DJI, among others.

mathematical-based models to Matlab Simulink to UML-based models (RQ2). They all convey a key requirement that these models must be executable. Models are mainly used to simulate the software system execution to derive or improve a test suite.

Testing CASS is cost-intensive and requires the exploitation of computational resources. We would conjecture that the effort required to test CASS is still an order of magnitude greater than the effort required to develop them. Much like had been observed in Matalonga et al. [13], the identified solutions in this RR all require significant design and development effort, and, when deployed, they tend to consume significant computational resources. Furthermore, all solutions identified in the RR cannot be transferred to other contemporary software systems (see the answer to RQ4 in Section 7.1). Therefore, while the alternatives presented in this paper set a path to test CASS, software testers are bound to reproduce the steps and technologies. Testing CASS is more cost-intensive than testing traditional software and requires the exploitation of cost-effective techniques that reduce the (1) testing time and (2) risks of damaging expensive hardware. We learned that multi-objective, search-based, and model checking-based testing techniques could be successfully applied to reduce CASS dynamic testing processes' costs.

7.4.3. Lesson learned 3: needs of not trivial testing environments in SDLC

The Software Development Life Cycle (SDLC) impacts the requirements for driving context variation in test environments. Indeed, to properly reproduce the variation of context during the dynamic execution of testing processes, it is necessary to design, implement and deploy a not trivial (and perhaps not available yet) test environment. The way the dynamic context variation is provided may depend on the specific stage of the SDLC. We observed that two main solutions might be applied. In the first stages of the development life cycle, the testing environment could provide a context-aware simulated environment where the context models can drive the simulation (see the answer to RQ2/perspective 4 in Section 7.1). Second, towards the SDLC final stages, the testing environment should resemble as much as possible the real execution environment where the CASS will run (see the answer to RQ2/perspective 3 in Section 7.1). Possible solutions to guarantee the testing environment's context-awareness are using emulators for mocking up real devices or using the same hardware/software components composing the final execution environment.

8. Future research directions on CASS testing

Drawing from the previous observations, the lessons learned in this RR, and our experience, we propose research directions for testing the context-awareness of contemporary software systems.

Future direction 1. Evaluating the efficacy of models to represent the real world. George box's quote is often repeated: "All models are wrong, but some are useful." We are concerned with the model's capacity to represent their production environments for testing CASS. If the model does not represent the production environment, then the results of the techniques can be challenged.

Future direction 2. Measuring the coverage of the test suites. Coverage measurements of the test suite were not identified in this RR. However, Matalonga et al. [13] identified a research interest concerning coverage measurement (see [59,60]). Nevertheless, coverage measurement was not observed in this sample of papers. Therefore, we would call for more research on the coverage measurement of the test suites regarding CASS.

Future direction 3. Functional suitability. Related to coverage is the question of generating test suites for evaluating the correct behavior of CASS in the face of the variation of context. This problem is related to the sheer volume of the test cases and the capacity to solve the test oracle problem [61] for each possible context adaptation. Applications of metamorphic testing [62] might be a possible way towards it. Nonetheless, we were not surprised that the primary sources did not capture this approach's applications, given its inherent complexity in non-academic settings. In any case, we claim this is a research line that needs further effort and investigation. Our preferred approach is to keep a degree of human factors in the loop [63,64] and then exploit it with simulations.

Future direction 4. Safe failure of safety-critical CASS. When looking at the reliability quality attribute, existing alternatives to testing CASS tend to use models and simulations to identify the combinations of inputs that might reveal a failure. This information can be captured in a test case and feedback into the development process for safe failure. However, as complete coverage of the test input space cannot be warrantied, eventually, the software will fail. Therefore, we would suggest that a research line should be to warranty the correct safe-failing behavior of CASS. It is a reduced problem from the test oracle, as it is only looking at ensuring a safe behavior when the set of inputs makes it fail.

Future direction 5. Management of test activities for CASS. The RR results raise the attention to managing test activities (RQ1 and RQ4). The importance of this future direction lies in the volume of test cases needed to cover the variation of context, which introduces challenges for the testing process management in non-academic settings. We would assume that practices and procedures should evolve to manage the context, and test suites in the CASS environments must be dynamically tested.

Future direction 6. Artificial Intelligence (AI) approaches for dealing with the complexity of the context. The RR indicated that novel testing techniques based on genetic algorithms [65] or reinforced learning [66] had been successfully adopted to reduce the costs of CASS's dynamic testing processes. We believe that this trend may lead to the use of AI in testing processes for CASS. Furthermore, AI could be exploited to implement solutions aiming to emulate the variation of context in a more realistic way.

Future direction 7. Tertiary, Multivocal, and Survey studies execution. As we observed in Section 3, many secondary studies addressing the topic of CASS testing have been published in the literature. We believe that the field is ripe to conduct a tertiary study that can uncover trends. Moreover, to deepen the understanding of how CASS are tested in practice, we believe that a Multivocal secondary study will be able to uncover practices that have not made their way into peer-reviewed publications. Finally, Survey studies where practitioners are interviewed to understand how they perform CASS testing processes.

9. Empirical suggestions for testing CASS

Good practice conveys that testing must be carried out in an environment close to the intended production environment. Each deviation from this heuristic means the test process risks the test Environment's behavior differently from the Production Environment. Thus, testing is and has always been, from its definition, an exercise in risk-taking. Since its initial formalizations, testing and test case selection are driven by the test analyst, striking a balance between comprehensiveness and risk of defects being carried into production. In modern CASS, these forces are pushed to levels that current test design techniques have yet to adapt. The sheer possibilities of variables and corresponding values in that context can make the conundrum of selecting a suitable set of test cases even more challenging. Worsened still by the cost of reproducing the production environment for testing.

We put forward three empirical suggestions based on our reflections regarding testing CASS that must be taken into consideration for developing technologies to test such software systems:

9.1. Conceptual

Accept the nature of the context and differentiate that the test item is

subjected to different input types - The *test input and the input from the context*. First, the context in which the test item is being executed will vary through external forces and interaction with the test item. Test management and Test design techniques for testing CASS must accept this. Secondly, test input and inputs from the context are two different types of inputs. The test input can and should be planned by the test analyst— the test input results from applying a test design technique during a test case definition. Model and simulation can enhance the capacity of the test analysis. However, as we mentioned, we prefer solutions that keep the test analyst in the loop.

On the other hand, context input is not within the control of the test analyst. Instead, they come from the interaction of the test item with the environment. CASS's test design techniques must accept this lack of control and design the test cases around it. We recommend that software testers accept this behavioral nature of context [67] when testing CASS and gradually relinquish control of the context to the environment. Thus, the software system's development stabilizes through its journey through the software development life cycle.

9.2. Technical

Start with a dynamic system model. All successful observed experiences of testing CASS started by modeling the software system. This model must support dynamic simulation. The state-based simulation will not capture the nature of the context. Computing resources can then be leveraged to evaluate input value combinations to fulfill a quality attribute (performance, reliability). In short, the idea is to have a system model to aid the test analyst with the development of test cases and explore computing simulations to minimize the risk that a combination of test cases can provide contextual inputs that leads to a failure (reliability). Finally, it is fed to the system during its execution in a production environment.

9.3. Procedural

Manage the context - and the exposure of the test item to the context - throughout the SDLC. Testing CASS is expensive. Therefore, there is little use in investing time, effort, and money into these complex solutions for testing CASS if there is no reasonable assurance that the software systems behave as expected and have been developed following accepted quality guidelines. We claim that other elements of the testing lifecycle must be evolved to accept CASS. As far as we could investigate, we have already noticed that this research did not find any study that caters to the influence on the context of the "Organizational Test Process'' or the "Test Management Process."

10. Conclusions

Testing context-aware software systems are challenging, mainly because of the behavioral nature of the context. Moreover, the context variation in the environment during the contemporary software system's lifecycle generates an exponential increase in the test input space. This increase worsens the challenge of selecting suitable test suites to guarantee the test item's quality.

This paper presents the result of a *Rapid Review* aimed to identify alternatives to testing CASS observed and evaluated in non-academic settings. We commissioned this RR under the assumption that CASS are mainstream (from self-driving autonomous vehicles to systems of systems that source data from multiple IoT sources). Therefore, they are being tested outside academia.

The results of this work can be summarized as follow.

- Current research initiatives focus on generating or improving test suites that can deal with context variation and the sheer volume of test input possibilities. To achieve this, we have observed strategies that revolve around two concepts: creating a dynamic model of the software system and using computer-based simulations to identify or improve test suites. Furthermore, we have made the case that these dynamic models of the system must evolve with the CASS in order, for the testing result, to maintain their representativeness of the changing context and evolving CASS requirements.
- All identified solutions are at a relatively early development stage and domain-specific. It means that there is still no technology readily transferable for testing CASS. Furthermore, we observed that the environment used to support the dynamic testing of CASS is complex. Finally, none of the technologies manage the testing process throughout the SDLC.
- Two important claims to help software testers regarding the state of practice have been presented. First, testing CASS is effort-intensive and costly in terms of computational resources. Second, testing CASS starts with having a useful dynamic model of the contemporary software system. We conceptualized these claims into three propositions that software testers must abide by when developing their technologies for testing CASS. Conceptual - *Accept the nature of context and differentiate that the test item is subjected to different input types: the test input and the input from the context.* Technical - *start with a dynamic model of the software system.* Procedural - *Manage the context - and the test item's exposure to the context - throughout the SDLC.*
- We proposed seven future research directions for evolving the state-of-the-art testing CASS. Current research has mostly focused on dynamic test execution. The research community must address other aspects of the testing life cycle to evolve knowledge regarding testing contemporary software systems.

CRediT authorship contribution statement

Santiago Matalonga: Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Domenico Amalfitano:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Andrea Doreste:** Investigation, Writing – original draft. **Anna Rita Fasolino:** Writing – review & editing. **Guilherme Horta Travassos:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.infsof.2022.106937.

References

[1] R.C. Motta, K.M. de Oliveira, G.H. Travassos, A conceptual perspective on interoperability in context-aware software systems, Inf. Softw. Technol. 114 (2019) 231–257, https://doi.org/10.1016/j.infsof.2019.07.001.
[2] U. Alegre, J.C. Augusto, T. Clark, Engineering context-aware systems and applications: A survey, J. Syst. Softw. 117 (2016) 55–83, https://doi.org/10.1016/j.jss.2016.02.010.
[3] D. Amalfitano, S. Matalonga, G.H. Travassos, Introduction to the special issue on engineering context-aware software systems, Inf. Softw. Technol. 132 (2021), 106509, https://doi.org/10.1016/j.infsof.2020.106509.
[4] J.y. Hong, E.h. Suh, S.J. Kim, Context-aware systems: A literature review and classification, Expert Syst. Appl. 36 (2009) 8509–8522, https://doi.org/10.1016/j.eswa.2008.10.071.
[5] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems, Int. J. Ad Hoc Ubiquitous Comput. 2 (2007) 263, https://doi.org/10.1504/IJAHUC.2007.014070.
[6] S. Klicarr, Trucks likely will never be driverless, 2018. https://www.trucker.com/technology/executives-trucks-likely-will-never-be-driverless (accessed November 22, 2019).

[7] J. Titcomb, Ex Google Engineer Completes 3,000 Mile Coast-to-Coast Journey in Driverless Car, Telegr (2018). https://www.telegraph.co.uk/technology/2018/12/18/ex-google-engineer-completes-3000-mile-coast-to-coast-journey/ (accessed November 22, 2019).

[8] R.O. Spínola, F.C.R. Pinto, G.H. Travassos, Supporting Requirements Definition and Quality Assurance in Ubiquitous Software Project, 2008, pp. 587–603, https://doi.org/10.1007/978-3-540-88479-8_42.

[9] R.O. Spínola, G.H. Travassos, Towards a framework to characterize ubiquitous software projects, Inf. Softw. Technol. 54 (2012) 759–785, https://doi.org/10.1016/j.infsof.2012.01.009.

[10] R.C. Motta, G.H. Travassos, On challenges in engineering IoT software systems, in: Proc. XXXII Brazilian Symp. Softw. Eng., 2018.

[11] I. de S. Santos, R.M. de C. Andrade, L.S. Rocha, S. Matalonga, K.M. de Oliveira, G. H. Travassos, Test case design for context-aware applications: Are we there yet? Inf. Softw. Technol. 88 (2017) 1–16, https://doi.org/10.1016/j.infsof.2017.03.008.

[12] S. Matalonga, G.H. Travassos, Testing Context-aware Software Systems: Unchain the Context, Set It Free!, in: Proc. 31st Brazilian Symp. Softw. Eng., ACM, New York, NY, USA, 2017, pp. 250–254, https://doi.org/10.1145/3131151.3131190.

[13] S. Matalonga, F. Rodrigues, G.H. Travassos, Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review, J. Syst. Softw. 131 (2017) 1–21, https://doi.org/10.1016/j.jss.2017.05.048.

[14] S. Matalonga, F. Rodrigues, G.H. Travassos, Challenges in Testing Context-Aware Software Systems, in SBQS, Bello Horizonte, 2015. Syst. Autom. Softw. Test.

[15] G. Travis, How the Boeing 737 Max Disaster Looks to a Software Developer, IEEE Spectr (2019).

[16] A. Schaun, O QUE PODE TER CAUSADO OS INCÊNDIOS DO CHEVROLET ONIX PLUS?, Auto Esporte, 2019. https://revistaautoesporte.globo.com/Noticias/noticia/2019/11/o-que-pode-ter-causado-os-incendios-do-chevrolet-onix-plus.html (accessed December 18, 2019).

[17] B. Cartaxo, G. Pinto, S. Soares, The role of rapid reviews in supporting decision-making in software engineering practice, in: ACM Int. Conf. Proceeding Ser, 2018, https://doi.org/10.1145/3210459.3210462.

[18] P. Moons, E. Goossens, D.R. Thompson, Rapid reviews: the pros and cons of an accelerated review process, Eur. J. Cardiovasc. Nurs. 20 (2021) 515–519, https://doi.org/10.1093/eurjcn/zvab041.

[19] G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, P. Steggles, G.D. Abowd, A.K. Dey, P.J. Brown, N. Davies, M. Smith, P. Steggles, Towards a Better Understanding of Context and Context-Awareness, in Proc. CHI 2000 Work, What, Who, Where, When How Context-Aware. (1999) 304–307, https://doi.org/10.1007/3-540-48157-5_29.

[20] N. Leveson, Are you sure your software will not kill anyone? Commun. ACM. 63 (2020) 25–28, https://doi.org/10.1145/3376127.

[21] B.R. Siqueira, F.C. Ferrari, K.E. Souza, V. V. Camargo, R. Lemos, Testing of adaptive and context-aware systems: approaches and challenges, Softw. Testing, Verif. Reliab. (2021). https://doi.org/10.1002/stvr.1772.

[22] C. Luo, J. Goncalves, E. Velloso, V. Kostakos, A Survey of Context Simulation for Testing Mobile Context-Aware Applications, ACM Comput. Surv. 53 (2020) 1–39, https://doi.org/10.1145/3372788.

[23] R. Pressman, B. Maxim, Software engineering : a practitioner's approach, McGraw-Hill Education, New York, NY, 2015.

[24] ISO/IEC/IEEE 29119-1, 2013, Software and systems engineering Software testing Part 1:Concepts and definitions, ISO/IEC/IEEE 29119-1:2013, 2013, pp. 1–64, https://doi.org/10.1109/IEEESTD.2013.6588537.

[25] ISO Standard, ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011, p. 34.

[26] ISO/IEC/IEEE 29119-2:2013, Software and systems engineering Software testing Part 2:Test processes, ISO/IEC/IEEE 29119-2:2013(E). (2013) 1–138. https://doi.org/10.1109/IEEESTD.2013.6588540.

[27] SEBoK contributors, Guide to the Systems Engineering Body of Knowledge, Guid. to Syst. Eng. Body Knowl. (2020). https://www.sebokwiki.org/w/index.php?title=Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)&oldid=60050 (accessed November 24, 2020).

[28] A. Costa, L. Teixeira, Testing Strategies for Smart Cities applications, in: Proc. III Brazilian Symp. Syst. Autom. Softw. Test. - SAST '18, ACM Press, New York, New York, USA, 2018, pp. 20–28, https://doi.org/10.1145/3266003.3266005.

[29] H.R. Schmidtke, A survey on verification strategies for intelligent transportation systems, J. Reliab. Intell. Environ. 4 (2018) 211–224, https://doi.org/10.1007/s40860-018-0070-5.

[30] D.R. Almeida, P.D.L. Machado, W.L. Andrade, Testing tools for Android context-aware applications: a systematic mapping, J. Brazilian Comput. Soc. 25 (2019) 12, https://doi.org/10.1186/s13173-019-0093-7.

[31] B.R. Siqueira, F.C. Ferrari, M.A. Serikawa, R. Menotti, V.V. de Camargo, Characterisation of Challenges for Testing of Adaptive Systems, in: Proc. 1st Brazilian Symp. Syst. Autom. Softw. Test. - SAST, ACM Press, New York, New York, USA, 2016, pp. 1–10, https://doi.org/10.1145/2993288.2993294.

[32] F. Ferrari, B.B. de P. Cafeo, J. Noppen, Ruzanna Chitchyan, Investigating Testing Approaches for Dynamically Adaptive Systems, in: 2nd Int. Work. Var. Compos., Porto Gallinhas, 2011.

[33] B.R. Siqueira, F.C. Ferrari, K.E. Souza, D.S.M. Santibanez, V.V. Camargo, Fault sTypes of Adaptive and Context-Aware Systems and Their Relationship with Fault-based Testing Approaches, in: 2020 IEEE Int. Conf. Softw. Testing, Verif. Valid. Work, IEEE, 2020, pp. 284–293, https://doi.org/10.1109/ICSTW50294.2020.00054.

[34] C. Wohlin, Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering, in: Proc. 18th Int. Conf. Eval. Assess. Softw. Eng., New York, NY, USA 38, 2014, pp. 1–38, https://doi.org/10.1145/2601248.2601268, 10.

[35] Y. Qin, T. Xie, C. Xu, A. Astorga, J. Lu, CoMID: Context-Based Multiinvariant Detection for Monitoring Cyber-Physical Software, IEEE Trans. Reliab. 69 (2020) 106–123, https://doi.org/10.1109/TR.2019.2933324.

[36] D. Amalfitano, S. Matalonga, A. Doreste, A.R. Fasolino, G.H. Travassos, A Rapid Review on Testing of Context-Aware Contemporary Software Systems, 2019.

[37] S. Rosenthal, C.I. Lewis in focus : the pulse of pragmatism, Indiana University Press, Bloomington, 2007.

[38] C. Xu, S.C. Cheung, X. Ma, C. Cao, J. Lu, Dynamic fault detection in context-aware adaptation, in: Proc. Fourth Asia-Pacific Symp. Internetware - Internetware '12, ACM Press, New York, New York, USA, 2012, pp. 1–10, https://doi.org/10.1145/2430475.2430476.

[39] E.M. Fredericks, B. DeVries, B.H.C. Cheng, Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty, in: Proc. 9th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. - SEAMS 2014, 2014, pp. 17–26, https://doi.org/10.1145/2593929.2593937.

[40] M. Sama, S. Elbaum, F. Raimondi, D.S. Rosenblum, Z. Wang, Context-Aware Adaptive Applications: Fault Patterns and Their Automated Identification, IEEE Trans. Softw. Eng. 36 (2010) 644–661, https://doi.org/10.1109/TSE.2010.35.

[41] M. Lahami, M. Krichen, M. Jmaiel, Safe and efficient runtime testing framework applied in dynamic and distributed systems, Sci. Comput. Program. (2016), https://doi.org/10.1016/j.scico.2016.02.002.

[42] C. Xu, S.C. Cheung, X. Ma, C. Cao, J. Lu, Adam: Identifying defects in context-aware adaptation, J. Syst. Softw. 85 (2012) 2812–2828, https://doi.org/10.1016/j.jss.2012.04.078.

[43] T. Ma, S. Ali, T. Yue, Modeling foundations for executable model-based testing of self-healing cyber-physical systems, Softw. Syst. Model. 18 (2019) 2843–2873, https://doi.org/10.1007/s10270-018-00703-y.

[44] A. Arrieta, G. Sagardui, L. Etxeberria, J. Zander, Automatic generation of test system instances for configurable cyber-physical systems, Softw. Qual. J. 25 (2017) 1041–1083, https://doi.org/10.1007/s11219-016-9341-7.

[45] S.Y. Shin, S. Nejati, M. Sabetzadeh, L.C. Briand, F. Zimmer, Test case prioritization for acceptance testing of cyber-physical systems: a multi-objective search-based approach, in: Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal. - ISSTA 2018, ACM Press, New York, New York, USA, 2018, pp. 49–60, https://doi.org/10.1145/3213846.3213852.

[46] S.Y. Shin, K. Chaouch, S. Nejati, M. Sabetzadeh, L.C. Briand, F. Zimmer, Uncertainty-aware specification and analysis for hardware-in-the-loop testing of cyber-physical systems, J. Syst. Softw. 171 (2021), 110813, https://doi.org/10.1016/j.jss.2020.110813.

[47] J. Fröhlich, J. Frtunikj, S. Rothbauer, C. Stückjürgen, Testing safety properties of cyber-physical systems with non-intrusive fault injection – An industrial case study, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) (2016), https://doi.org/10.1007/978-3-319-45480-1_9.

[48] R. Ben Abdessalem, A. Panichella, S. Nejati, L.C. Briand, T. Stifter, Testing autonomous cars for feature interaction failures using many-objective search, in: Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng., ACM, New York, NY, USA, 2018, pp. 143–154, https://doi.org/10.1145/3238147.3238192.

[49] R. Ben Abdessalem, S. Nejati, L.C. Briand, T. Stifter, Testing advanced driver assistance systems using multi-objective search and neural networks, in: Proc. 31st IEEE/ACM Int. Conf. Autom. Softw. Eng., ACM, New York, NY, USA, 2016, pp. 63–74, https://doi.org/10.1145/2970276.2970311.

[50] R. Ben Abdessalem, S. Nejati, L.C. Briand, T. Stifter, Testing vision-based control systems using learnable evolutionary algorithms, in: Proc. 40th Int. Conf. Softw. Eng., ACM, New York, NY, USA, 2018, pp. 1016–1026, https://doi.org/10.1145/3180155.3180160.

[51] Y. Qin, C. Xu, P. Yu, J. Lu, SIT: Sampling-based interactive testing for self-adaptive apps, J. Syst. Softw. 120 (2016) 70–88, https://doi.org/10.1016/j.jss.2016.07.002.

[52] M. Héder, From NASA to EU: The evolution of the TRL scale in Public Sector Innovation, Innov. J. (2017).

[53] U. of S.D.-A. Program, TRL Estimator, (n.d.). http://aries.ucsd.edu/ARIES/MEETINGS/0712/Waganer/TRLCalcVer2_2.xls (accessed January 11, 2021).

[54] R. Wieringa, N. Maiden, N. Mead, C. Rolland, Requirements engineering paper classification and evaluation criteria: a proposal and a discussion, Requir. Eng. 11 (2006) 102–107, https://doi.org/10.1007/s00766-005-0021-6.

[55] S. Easterbrook, J. Singer, M.A. Storey, D. Damian, Selecting empirical methods for software engineering research, Guid. to Adv. Empir. Softw. Eng. (2008), https://doi.org/10.1007/978-1-84800-044-5_11.

[56] G.H. Travassos, M.D.O. Barros, Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering, in 2nd Work, Work. Ser. Empir. Softw. Eng. Futur. Empir. Stud. Softw. Eng. (2003).

[57] H.A. Linstone, M. Turoff, The Delphi method : techniques and applications, Addison-Wesley Pub. Co., Advanced Book Program, Reading, Mass, 1975.

[58] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, https://doi.org/10.1007/978-3-642-29044-2.

[59] H. Wang, W.K. Chan, Weaving Context Sensitivity into Test Suite Construction, in: 2009 IEEE/ACM Int. Conf. Autom. Softw. Eng., IEEE, Auckland, 2009, pp. 610–614, https://doi.org/10.1109/ASE.2009.79.

[60] H. Wang, W.K. Chan, T.H. Tse, Improving the Effectiveness of Testing Pervasive Software via Context Diversity, ACM Trans. Auton. Adapt. Syst. 9 (2014) 1–9, https://doi.org/10.1145/2620000, 928.

[61] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, The Oracle Problem in Software Testing: A Survey, Softw. Eng. IEEE Trans. 41 (2015) 507–525, https://doi.org/10.1109/TSE.2014.2372785.
[62] W.K. Chan, T.Y. Chen, H. Lu, T.H. Tse, S.S. Yau, Integration testing of context-sensitive middleware-based applications: a metamorphic approach, Int. J. Softw. Eng. Knowl. Eng. 16 (2006) 677–703, https://doi.org/10.1142/S0218194006002951.
[63] F. Rodrigues, CATS Design: A Context-Aware Testing approach, Universidade Federal do Rio de Janeiro, 2015.
[64] D. Andreia C, G.H. Travassos, Towards supporting the specification of Context-Aware software system test cases. CIBSE 2020, 2020.
[65] Y. Luo, X.-Y. Zhang, P. Arcaini, Z. Jin, H. Zhao, F. Ishikawa, R. Wu, T. Xie, Targeting Requirements Violations of Autonomous Driving Systems by Dynamic Evolutionary Search, in: 2021 36th IEEE/ACM Int. Conf. Autom. Softw. Eng., IEEE, 2021, pp. 279–291, https://doi.org/10.1109/ASE51524.2021.9678883.
[66] T. Ma, S. Ali, T. Yue, Testing self-healing cyber-physical systems under uncertainty with reinforcement learning: an empirical study, Empir. Softw. Eng. 26 (2021) 52, https://doi.org/10.1007/s10664-021-09941-z.
[67] P. Dourish, What we talk about when we talk about context, Pers. Ubiquitous Comput. 8 (2004) 19–30, https://doi.org/10.1007/s00779-003-0253-8.