Self-Adaptive Testing in the Field

SAMIRA SILVA, Gran Sasso Science Institute (GSSI), Italy
PATRIZIO PELLICCIONE, Gran Sasso Science Institute (GSSI), Italy
ANTONIA BERTOLINO, ISTI-CNR, Italy

We are increasingly surrounded by systems connecting us with the digital world and facilitating our life by supporting our work, leisure, activities at home, health, etc. These systems are pressed by two forces. On the one side, they operate in environments that are increasingly challenging due to uncertainty and uncontrollability. On the other side, they need to evolve, often in a continuous fashion, to meet changing needs, to offer new functionalities, or also to fix emerging failures. To make the picture even more complex, these systems rarely work in isolation and often need to collaborate with other systems, as well as humans. All such facets call for moving their validation during operation, as offered by approaches called testing in the field.

In this paper, we observe that even the field-based testing approaches should change over time to follow and adapt to the changes and evolution of collaborating systems or environments or users' behaviors. We provide a taxonomy of this new category of testing that we call self-adaptive testing in the field (SATF), together with a reference architecture for SATF approaches. To achieve this objective, we surveyed the literature and collected feedback and contributions from experts in the domain via a questionnaire and interviews.

Additional Key Words and Phrases: Software testing in the field, Self-adaptive testing, Knowledge gaps

1 INTRODUCTION

Traditionally, the software testing task has always been thought of as a fault-finding process performed during the development cycle [2]. Since the activities in this process are commonly performed in the development environment, the work in [8] refers to such activities as *in-house* or *in-vitro* software testing. However, in recent years academic researchers and industry professionals are becoming more and more aware of the need to continue testing even after software release, when the system is in use, e.g., [4, 7, 22, 44]. In fact, many of the failures reported in production correspond to problems that would be difficult, if not impossible, to uncover by in-house testing [27] due to the tremendous complexity, evolvability, and interconnection of current software-intensive systems. The work in [8] defines this tendency of moving testing activities from the development to the production environment with different nuances. *Testing in the field* [8] can be subdivided into *online testing*, *offline testing*, and *ex-vivo testing*. Test cases are executed directly in the field, on the same instance of the system used in production, which is called *online testing*, or on a separated instance that still runs in production, called *offline testing*. Finally, *ex-vivo testing* refers to the case in which test cases are executed in-house but using data collected from the field.

The above cited work [8] reviews the literature on field-based testing techniques. It selected 80 primary studies published from 1989 to 2017 and classified them according to different dimensions, including, among

Authors' addresses: Samira Silva, samira.silva@gssi.it, Gran Sasso Science Institute (GSSI), Viale Francesco Crispi, 7, L'Aquila, AQ, Italy, 67100; Patrizio Pelliccione, patrizio.pelliccione@gssi.it, Gran Sasso Science Institute (GSSI) , Viale Francesco Crispi, 7, L'Aquila, AQ, Italy, 67100; Antonia Bertolino, ISTI-CNR, Via Giuseppe Moruzzi, 1, Pisa, Italy, antonia.bertolino@isti.cnr.it.

1556-4665/2023/10-ART $15.00
https://doi.org/10.1145/3627163

others, how, when, and where the generation and activation of field tests take place. The authors of this review noticed that many of the collected works adopt some adaptation strategy in order to deal with uncertainty or confront emergent behaviors of the SUT (System Under Test). Furthermore, they conclude their summary of open challenges in field-based testing by stating that, concerning test cases generation, test cases "*shall adapt to the production environment*" [8], and that, concerning the oracles for field testing, these "*need to adapt to the unknown execution conditions that can emerge in the field*" [8].

The definition of adaptive testing is also investigated in the recent area of "software cybernetics" [13], which looks at how software and control processes interact. In this scenario, adaptive testing means that a software testing strategy is improved by taking advantage of the information collected during the testing process, which deepens our comprehension of the tested system [12]. However, this definition goes in a different direction since it does not refer to testing in the production environment, as we are concerned here.

As a matter of fact, we believe that the same reasons for moving testing activities from in-house to the field, such as tackling dynamism, context-dependence, and uncertainty, also support the need for self-adaptive testing approaches. Field-based testing is in general important to reveal those faults that escape in-house testing [8], but it becomes even more important for systems that change over time. In other words, **self-adaptability represents an important feature when carrying out field testing activities**.

Self-adaptive systems have been actively researched in the recent years [16, 36, 50]. They can be described as software-intensive systems that modify themselves while in operation in response to internal activities or changes in the environment (the SUT environment, that is, the system under which the SUT is running, which may include software and hardware) or context (a broader concept that involves other variables, such as changes in the rules of the company to which the SUT belongs) in which they operate, and that may also be based on predetermined properties or policies [50]. This description, in our opinion, is equally appropriate for a software framework whose objective is to carry out testing in the field: the test approach may need to be modified in response to events in the field or due to shifting needs or expectations. In this work, we refer to such category of testing approaches as "self-adaptive testing in the field" (SATF for short). Generally speaking, SATF might be a useful option to supplement traditional testing rather than replace it.

In fact, several of the strategies proposed to test self-adaptive systems (SAS) are inherently designed to be used at runtime, generally when an adaptation takes place, and are themselves conceived as adaptive (we analyse many of such works in Section 4). Nonetheless, we do not believe, and this is confirmed by some of the works we collected (e.g., [15, 19, 29, 41]), that the SATF scope is restricted to SAS. It is unquestionably a wise technique for testing new service choreographies, as was first anticipated in the Choreos European project [9] where policies were established to adapt the runtime testing approach. More generally, in a recent opinion paper that explores challenges and promising directions in testing "changing software in a changing world" [10], the authors state that the testing strategy should support the adaptive generation of test plans. From a more pragmatic perspective, based on the primary studies covered in this work we can notice that there are three types of systems for which the use of self-adaptive testing in the field approaches would play an important role: (i) self-adaptive systems (e.g., [23]), (ii) systems that could be updated over time, due to the origination of new needs, among other reasons (e.g., [41]), and (iii) systems for which the configuration may be changed by the user at runtime (e.g., [15]). Of course, we could even have systems that present more than one of the above-mentioned dynamic characteristics.

In the aforementioned review on field-based testing [8], even though self-adaptation aspects do appear at various points, they are not examined as a stand-alone dimension. With respect to that work, this study puts the spotlight on field testing approaches that can, in a way or another, self-adapt themselves. We do not restrict a priori what aspect of the approach is adaptive or when the adaptation occurs, or what is the trigger that launches the adaptation. In fact, our research aims at properly defining such approaches, explaining what are their characteristics, and proposing a reference model.

This work extends a previous work published at SEAMS in 2022 [53], where we surveyed the literature of SATF, highlighted the main characteristics of the topic, and discussed some research challenges. In this work, we extend the previous one by (i) expanding literature coverage to papers published recently and to papers identified via snowballing (not performed in the SEAMS paper), (ii) providing a taxonomy for SATF according to the consolidated and refined list of characteristics of SATF and their relationships (also specified via a feature model), (iii) providing a reference architecture for self-adaptive approaches to field testing, and (iv) providing an updated list of research gaps and challenges in self-adaptive field-based testing.

More precisely, we aim to answer the following research questions:

RQ1 What are the main dimensions of a taxonomy for self-adaptive field-based testing?
RQ2 What are the main components of a reference architecture for self-adaptive field-based testing approaches?
RQ3 What are the known gaps and/or challenges in self-adaptive field-based testing?

RQ1 looks at existing definitions of SATF and at the characteristics of existing approaches and proposes a taxonomy for SATF. RQ2 builds on the data collected to answer RQ1, as well as on the taxonomy built in RQ1, and proposes a reference architecture for SATF systems. RQ3 discusses the knowledge gaps and challenges encountered when performing SATF.

On the methodological side, we constructed the taxonomy and the reference architecture by following a design science research methodology organized in three iterations. The first iteration consists of the work described in our previous paper [53]. The other two iterations, are the new content described in this work. More details about the research methodology we followed are provided in Section 3.

Paper outline. Section 2 discusses related work. Section 3 describes the methodology followed for the study. Section 4 reports the results of the study in outlining a taxonomy for SATF. Section 5 presents the reference architecture for SATF approaches. Section 6 discusses the identified open challenges. Section 7 describes the activities we performed to validate the taxonomy and the reference architecture, and presents the outcome of the validation. Section 8 characterizes limitations and threats to validity of this study. Finally, in Section 9, we discuss the findings and suggest promising future research avenues.

2 RELATED WORK

As detailed in Section 3, to build awareness of the problem we surveyed the literature on approaches for self-adaptive testing in the field. In this section, we prepare to this view of the state of the art on self-adaptive testing in the field with an analysis of existing secondary studies that are related to our work and with some additional works that we identified by performing specific search. For example we searched for works discussing challenges of testing self-adaptive systems since this is relevant for RQ3. Indeed, these works are not part of the set of primary studies we analysed in the survey we performed, but they are worth attention in this section.

Although there are a few secondary studies that partially relate to our subject, we are not aware of any previous literature reviews focusing on the same objective. A systematic literature review (SLR) has been carried out in 2016 aiming at recognizing and categorizing the challenges encountered when testing SAS [54]. This paper gathered 25 primary studies (published from 2003 to 2015) and identified 12 types of challenges to be faced. The challenges mentioned by their work are described as follows:

(1) How to handle the exponential growth of SAS configurations that need to be tested;
(2) How to ensure the correctness of SAS configurations that have never been subject to the test beforehand;
(3) How to spot and prevent erroneous SAS configurations defined at runtime;
(4) How to test SAS running in a distributed and heterogeneous environment;
(5) How to handle user interference in the configuration of SAS;
(6) How to foresee all relevant context changes and when they might have an impact on SAS behavior;
(7) How to handle data flow and context-dependent control in SAS;

(8) How to maintain updated test cases in a changing environment, and how to keep track of them in relation to system requirements and components;
(9) How to decide which sensors and how frequently the execution environment of SAS must be monitored for data;
(10) How to realistically simulate SAS execution environment and workload;
(11) How to automatically produce test cases for a dynamic environment; and
(12) How to conduct formal verification of adaptive behavior.

We believe that many of these challenges could be mitigated when taking into account adaptive testing in the field, for example, the high number of configurations to be tested (Challenge 1), or context-dependency (Challenges 6 and 7), or test cases maintenance (Challenge 8), among others. In Section 6 we will come back to these challenges to check whether there is an overlap with the challenges we identified for SATF or whether SATF helps in mitigating some of these challenges.

The work in [48] focuses on a specific class of self-adaptive systems, i.e., those utilizing health states for computing repair actions. It identifies the following challenges:

(1) *Oracle problem.* The authors highlight that the oracle problem is difficult. In this work, we distinguish between oracles that are embedded in the test cases, and oracles that are implemented outside the test cases and that we refer to as independent oracle (see Sections 4.2.3 and 4.2.2). From what is described in the paper, it seems that they are mostly referring to independent oracles. It is worth to mention that the oracle problem in general is investigated in [5].
(2) *Masking faults.* Simultaneously occurring faults may hide each other.
(3) *Nondeterminism.* It may be caused by the order in which events are sensed, monitored, and managed.
(4) *Dividing and conquering.* The strategy of "divide and conquer" is in general used to simplify complexity. However, the authors highlight that it is not always simplifying testing. In fact, even though the implementation of the SUT features is simplified, testing needs to consider all the possible cases and configurations.
(5) *Hardware.* Dealing with systems that include also hardware and not only software, might make testing more complex.
(6) *Safety and security.* These important aspects should be preserved even during adaptation and reconfiguration. Hence testing should check these aspects.

We believe that SATF approaches could help mitigating some of these challenges, and will discuss this more in depth in Section 6.

In 2021, two SLRs that concentrate on methods for testing a software system at runtime in its actual execution environment have been published. The already cited work in [8] distinguishes between testing methods that are undertaken in production (in-vivo) and others that use data from production. Their search query is more extensive than ours since it includes non-adaptive testing techniques in addition to adaptive ones, rather than only focusing on adaptive ones as we do. Therefore, in principle, our set of primary studies would be a subset of theirs. On the other hand, they cover the literature by looking at papers published until 2017, while we considered the literature until 2022, and 8 out of the 19 (i.e., 42%) primary studies we selected are, indeed, published after 2017. Furthermore, despite they include some of the existing works in SATF, in their work the notion of test adaptability is not a central concern as for us, and there is no discussion on the specifics of test adaptation, such as what is adapted, how, and when.

Another SLR on runtime testing [37] limits the scope to approaches to test systems that are dynamically adaptive and distributed, and in contrast to the SLR in [8], which did not limit the domain of the tested application, the authors actually chose fewer papers (precisely, 43 from 2006 to 2020 in [37], against 80 from 1989 to 2017 in [8]). In [37], the authors pose eight research questions about the properties of runtime test approaches, and

Table 1. Overview of existing secondary studies either on Testing in the Field or Self-adaptive Testing.

Paper	Year	Aim of Review	Focus on Testing in the Field	Focus on Self-adaptive Testing	Period
[54]	2016	Challenges when testing SAS	-	✓	2003-2015
[55]	2020	Faults when testing adaptive systems and context-aware systems	-	✓	-2019
[37]	2021	Testing in the field of dynamically adaptive and distributed systems	✓	✓	2006-2020
[8]	2021	Testing in the field	✓	-	1989-2017
[48]	2022	Challenges when testing a specific class of SAS	-	✓	-
This work	2023	Self-adaptive testing in the field	✓	✓	2012-2022

among them, RQ5 - which is phrased as: "What kind of dynamic adaptations can these approaches support?" - seems to be the one most closely related to our study. Nonetheless, considering the response they provide to the question, we comprehend that the adaptation they focus on concerns again the SUT, whereas their study does not address adaptation of the testing approach itself as we do here.

Lastly, a recent SLR [55] surveys faults that are experienced when testing adaptive systems and context-aware systems. As stated by the authors, such a study can be helpful for comprehending first the nature of flaws specific to those kinds of systems and then coming up with suitable testing approaches to find those types of faults. Thus, they concentrate more on fault-based testing methods. While still pertinent, their SLR entirely diverges from the focus of our current research because, unlike us, they do not characterize self-adaptive testing in the field as we aim to do.

Table 1 shows an overview of the above-mentioned secondary studies. It is important to notice that the work in [48] does not establish a specific period for its research. Also, we consider that a work focuses on self-adaptive testing even if this is not explicitly said, as in [37], a work that focuses only on testing dynamically adaptive and distributed systems, without considering the self-adaptive testing of other types of systems, as we do.

3 RESEARCH METHODOLOGY

In this work, we aim at defining a taxonomy for SATF, at contributing a reference architecture for SATF approaches, and at identifying and discussing the main challenges. To achieve these objectives we followed the design research method [17], which is mainly used for creating new knowledge and artifacts that are required to address a particular phenomenon or problem.

As shown in Figure 1, we performed three iterations (first, second, and third), and each iteration consisted of three steps (awareness of the problem, solution development, and evaluation); overall, this leads to nine phases. Each phase in the research methodology is identified by a code <x>.<y>, where <x> represents the step and <y> represents the iteration.

The first iteration has been reported in our previous paper [53] (phases 1.1, 2.1, and 3.1, i.e., the three steps of the first iteration). The other two iterations represent the extension of [53] that we present in this work. The three steps permit us to work incrementally to answer the three research questions. Precisely, the answer to the three RQs is provided at the end of the third step, by providing a taxonomy (RQ1), a reference architecture (RQ2), and challenges (RQ3), which have been also benchmarked with challenges available in the related works. In the following, we describe details of the six phases, organized and presented by the three steps:

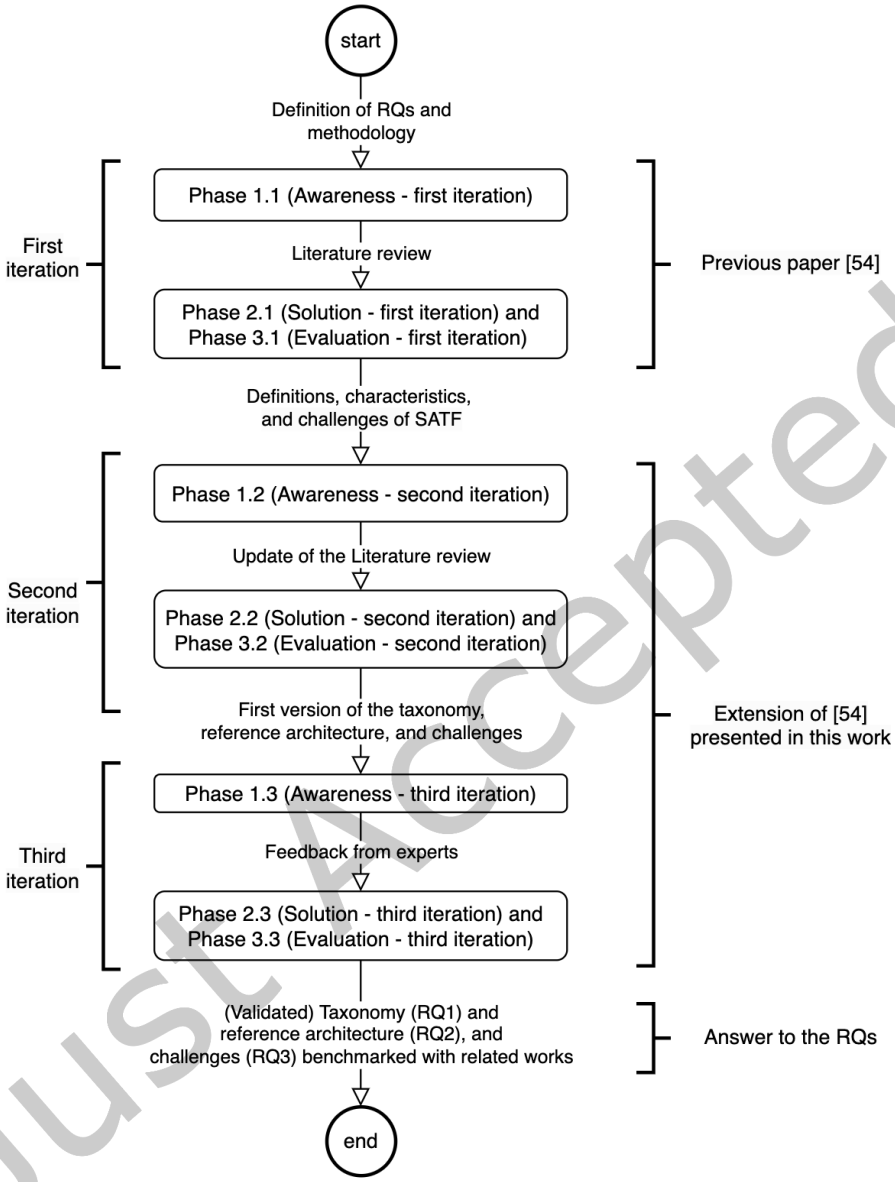

Fig. 1. Overview of the research methodology mapped to the results.

1. Awareness of the problem:

Phase 1.1 - Awareness, First Iteration: In the first iteration, the awareness of the problem was mainly developed through data collection from the literature. The collected data has been then analysed to extract information useful to define SATF and identify its characteristics and challenges.

Phase 1.2 - Awareness, Second Iteration: As described above, the second iteration has been performed to extend our previous work, so there was the need to update the previously performed review of the literature to papers published after [53]. In this iteration, we also complemented the review with snowballing to achieve a higher level of completeness. The identified primary studies have been then analysed to extract information useful to provide a first version of the taxonomy, reference architecture, and challenges.
Phase 1.3 - Awareness, Third Iteration: In this iteration, we exploited the feedback coming from experts of the domain to contribute a final and validated version of the solutions (see Phase 2.3) and we analysed the challenges highlighted in related secondary studies (see Phase 3.2).

2. Solution development:
Phase 2.1 - Solution, First Iteration: we contributed definitions, characteristics, and challenges of SATF. This step has been reported in our previous paper [53].
Phase 2.2 - Solution, Second Iteration: we contributed a first version of the taxonomy of SATF, reference architecture, and challenges.
Phase 2.3 - Solution, Third Iteration: we contributed a final and validated version of the taxonomy of SATF, reference architecture, and challenges.

3. Evaluation:
Phase 3.1 - Evaluation, First Iteration: We validated the definition and characteristics of SATF thanks to the participants of SEAMS 2022 that attended the presentation of the paper at the conference. Moreover, we presented the findings in other events, like an event about robotic software engineering[1]. Specifically, we gave a presentation (disseminated the work), collected lively feedback in the form of questions/answers after the presentation, and had a follow-up with 3 attendees (overall) that showed interest in the work.
Phase 3.2 - Evaluation, Second Iteration: We validated the first version of the taxonomy and reference architecture with experts in the domain. Moreover, we compared the identified challenges with those present in related secondary studies.
Phase 3.3 - Evaluation, Third Iteration: We validated the final version of the taxonomy and reference architecture with a subset of experts involved in phase 3.2. Specifically, in this phase, we involved only those experts that suggested or triggered changes in the previous iteration of the validation (phase 3.2). The validation, in fact, focused on the performed changes, to check whether we correctly understood and implemented the recommended changes. We did not involve those experts that were already satisfied during phase 3.2.

In the remainder of this section, and specifically in Section 3.1, we describe the scoping review made in step 1.1 and its subsequent update made in step 1.2. Then, in Section 3.2 we describe the validation we made with experts of the domain, in steps 3.2 and 3.3. We believe that the other steps summarized above do not need further explanation. We provide a replication package of the activities done in this research at https://tinyurl.com/2kdb5jef.

3.1 Scoping review

In this work, we employed the scoping review research methodology [18, 43, 46] to address the research questions mentioned in the introduction. The scoping review research methodology is popular in the medical field [18, 43, 46], but it is also gaining popularity in the field of software engineering [3]. Scoping reviews, according to [43], are analogous to systematic literature reviews and they adhere as well to a rigid structured process. On the other hand, they highlight some distinct essential methodological principles and have different objectives. In cases where systematic reviews are unable to achieve the essential goals or satisfy the demands of knowledge users, scoping reviews are increasingly recognized as a viable strategy [43].

[1] https://rsemeeting.github.io/rse2022

Since SATF is not yet developed enough to effectively conduct a systematic literature review, we think that scoping reviews are the most appropriate study methodology. Given that it identifies and analyzes knowledge gaps in the field of self-adaptive testing, this work may potentially serve as a useful precursor to upcoming systematic reviews. According to the traditional goals for performing a scoping review [43], this study will aid in (i) identifying knowledge gaps and important traits or elements associated with a concept, (ii) scoping a body of literature, and (iii) clarifying terminology and definitions used in the literature. The review protocol is presented in detail below to facilitate validation and replication studies. However, readers not interested in the methodology can skip to the brief summary in Sub-section 3.1.7.

3.1.1 Search Strategy. In light of the research questions mentioned in the introduction, it was decided that the search string to be used for searching the pertinent publications should comprise the term "software", co-joint with the terms identifying testing in operation, such as: "online testing", "runtime testing", or "field-based testing", and one other set of terms referring to the adaptation aspect, such as: "self-adaptive", "self-organizing", "autonomous", "self-managing", or "adaptive". These keywords were then connected appropriately using the Boolean operations AND and OR, obtaining the following search string:

```
software AND ("online test*" OR "run-time test*" OR "runtime test*" OR "field-based test*")
AND (self-adaptive OR "self adaptive" OR self-organizing OR "self organizing" OR autonomous OR
self-managing OR "self managing" OR "self organising" OR self-organising OR adaptive)
```

In comparison with the often cited previous review in [8], we have added here one more co-joint term relative to adaptiveness, which in the previous string was included by means of one only adjective "adaptive" included with an OR operator; i.e., here we focus specifically on self-adaptive approaches.

In order to be as comprehensive as possible, we applied our search across the "full text" of the articles under consideration. The databases that the aforementioned query was applied to are shown in Table 2. Three of the most popular databases, IEEExplore, ACM Digital Library, and Scopus, were employed in our scoping review.

Table 2. Search engines for scientific papers.

#No	Database	URL
1	IEEExplore	https://ieeexplore.ieee.org/
2	ACM Digital Library	https://dl.acm.org/
3	Scopus	https://www.scopus.com/

The number of records extracted for Scopus, ACM Digital Library, and IEEExplore, utilizing the aforementioned databases and the suggested search query, and only taking papers published from 2012 to 2022 into consideration, is equivalent to 637, 486 and 21, respectively[2].

3.1.2 Screening and Duplicate Removal. We screened the papers by simply reading their title and abstract as there were a variety of papers that matched the query but were not pertinent. Most of them were related to other scientific fields or were clearly related to other meanings for the terms searched. For instance, the term "online test*" is also used in papers tackling e-learning. All of these papers considered as unrelated are then eliminated. Next, after removing the duplicates and combining the results from screening for several databases into a single set, we obtained 121 publications.

[2]The literature search was conducted in more subsequent dates, precisely the searches for the previous version [53] were launched on 30/11/2021 (ACM and IEEE) and 06/12/2021 (Scopus). A search for more recent papers was then launched on 14/06/2022 (ACM, IEEE, and Scopus) but did not found any primary study to be added.

3.1.3 Selection Criteria. The selection criteria are summarized in Table 3. The inclusion criteria are based on the research questions, while the exclusion ones are standard quality criteria (similar to those from [8]). Aiming at comprehensiveness, we considered publications written in English within the last 10 years that could be downloaded. It is important to highlight that our search returned no publications written in languages other than English or that could not be downloaded.

Ultimately, 39 papers were obtained as a result of the application of inclusion and exclusion criteria.

Table 3. Inclusion and exclusion selection criteria.

Inclusion criteria	**Exclusion criteria**
I1 - Papers that provide a definition on self-adaptive testing in the field. I2 - Papers that describe characteristics of approaches to perform self-adaptive testing in the field. I3 - Papers that describe gaps/challenges in self-adaptive testing in the field. I4 - Papers that describe components of a model or an architecture to self-adaptive testing in the field. I5 - Related papers published from 2012 up to 2022.	E1 - Papers that cannot be downloaded. E2 - Studies in languages other than English. E3 - Papers published before 2012. E4 - Unpublished papers. E5 - Secondary studies papers. E6 - Overlapping, duplicate and out-of-scope papers (i.e. not fulfilling I1, I2, I3 or I4).

3.1.4 Full-text Checking. The selected papers were merged and given to two of the authors for review. Each reviewer separately read the full text of every paper before choosing whether or not it should be included. In plenary meetings, all papers that generated conflicting perspectives were discussed. Finally, after a final agreement, 16 papers made it to the final list.

3.1.5 Snowballing. The term "snowballing" describes the process of finding more publications by leveraging a paper's reference list, backward snowballing, or its citations, forward snowballing [57]. We performed both backward and forward snowballing, using as the initial list the one obtained in the previous stage that contains 16 papers. In the snowballing phase, the screening of papers and application of inclusion & exclusion criteria are also performed. A first application of snowballing resulted in 3 papers accepted out of 753 obtained. Then, with the 3 papers accepted in the previous iteration, we performed a second snowballing search that resulted in 1 accepted papers among 141 obtained. Finally, by using the paper accepted in the preceding search, we finally executed the snowballing a third time, obtaining 78 papers but did not accept any of them. Then, the total the number of papers collected by the snowballing phases[3] is equal to 4, resulting in 20 papers in total. Among these, we noticed an extended work of a paper already collected during the regular process. We then removed the older overlapping paper and kept the most recent one. Thus, **the final list of papers collected in this work contains 19 papers**.

3.1.6 Data Extraction. Each author read the entire work and then individually extracted the information needed to respond to the three research questions (RQs) outlined in the Introduction (Section 1). Thus, we had two independent readings and classifications for each paper (in particular, for answering RQ2, we used a spreadsheet currently available from the replication package). These classifications were then discussed in a series of meetings that the authors attended in order to clarify any unclear categories and resolve any potential disagreements. It is important to notice that we performed the data extraction process incrementally in order to take into account the incremental building of the feature model and the reference architecture.

[3]The phases of snowballing were conducted on 14/07/2022, 30/08/2022 and 07/09/2022, respectively

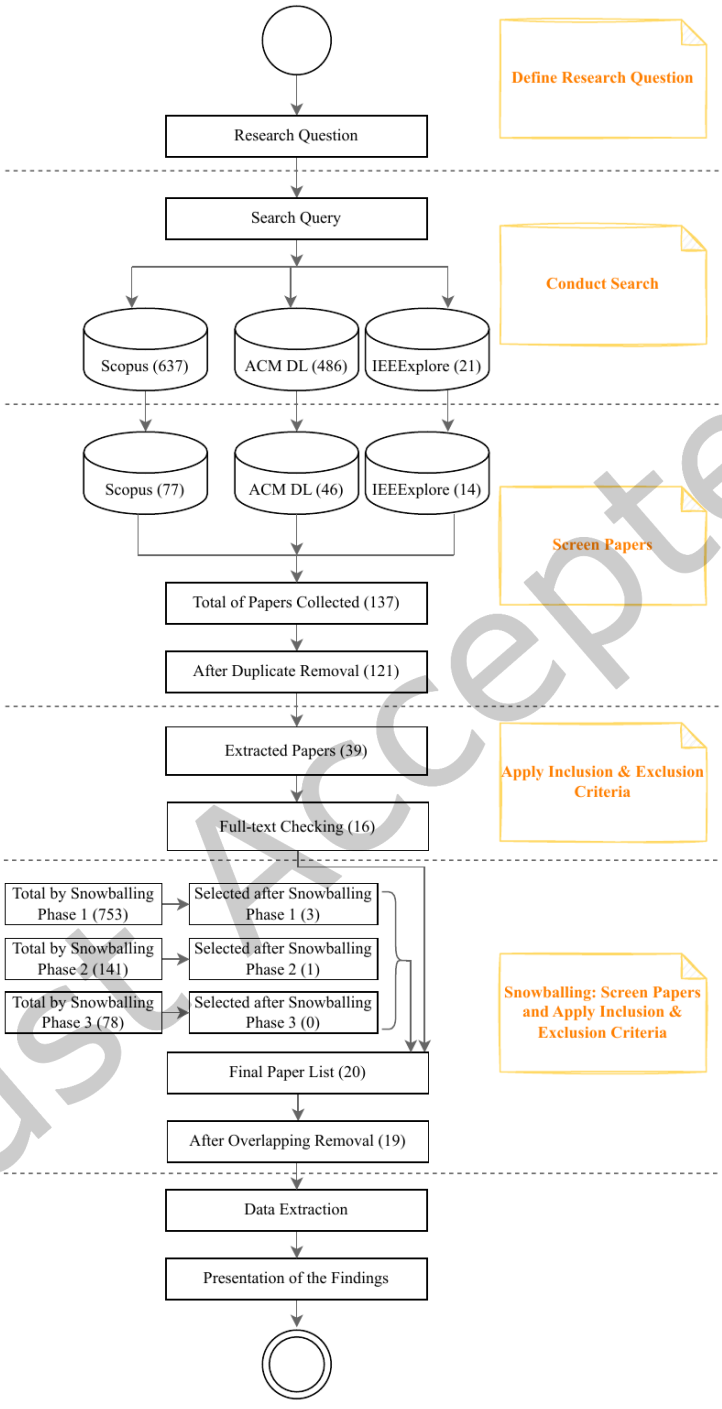

Fig. 2. Summary of the selection protocol for this scoping review.

Table 4. SATF approaches.

Final List of Papers			
Ref.	Year	Venue	Title
[42]	2012	International Workshop on European Software Services and Systems Research - Results and Challenges (S-Cube)	Verification and testing at run-time for online quality prediction
[25]	2013	International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)	Towards run-time testing of dynamic adaptive systems
[19]	2014	IEEE Transactions on Services Computing (TSC)	Dynamic test reconfiguration for composite web services
[24]	2014	International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)	Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty
[23]	2015	International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)	Automated generation of adaptive test plans for self-adaptive systems
[41]	2016	International Workshop on Automating Test Case Design, Selection, and Evaluation (A-TEST)	Automated workflow regression testing for multi-tenant saas: integrated support in self-service configuration dashboard
[51]	2016	International Middleware Conference (Middleware)	Bifrost – Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies
[38]	2016	Elsevier Science of Computer Programming (SCP)	Safe and efficient runtime testing framework applied in dynamic and distributed systems
[32]	2016	International Workshops on Foundations and Applications of Self* Systems (FAS* W)	Towards autonomous self-tests at runtime
[49]	2017	International Conference of Electronics, Communication and Aerospace Technology (ICECA)	Self-test framework for self-adaptive software architecture
[31]	2017	International Workshop on Variability and Complexity in Software Design (VACE)	Towards collective online and offline testing for dynamic software product line systems
[29]	2019	International Symposium on Software Reliability Engineering Workshops (ISSREW)	A hybrid framework for web services reliability and performance assessment
[39]	2019	Computer Software and Applications Conference (COMPSAC)	The SAMBA approach for Self-Adaptive Model-Based online testing of services orchestrations
[15]	2020	International Conference on Software Testing, Validation and Verification (ICST)	A Framework for In-Vivo Testing of Mobile Applications
[47]	2020	Wiley Software Testing, Verification and Reliability (STVR)	Testing microservice architectures for operational reliability
[30]	2021	International Conference on Web Research (ICWR)	On-demand Test as a Web Service Process (OTaaWS Process)
[21]	2021	Elsevier Information and Software Technology (IST)	Runtime testing of context-aware variability in adaptive systems
[14]	2022	International Conference on Automation of Software Test (AST)	Microservices Integrated Performance and Reliability Testing
[26]	2022	International Conference on Software Testing, Verification and Validation (ICST)	Testing Software in Production Environments with Data from the Field

3.1.7 Review Protocol Summary. This study has mainly focused on papers outlining strategies for self-adaptive testing in the field. The entire workflow employed in this scoping review is shown in Figure 2. 137 records in total were found across 3 separate databases. 121 records were later determined to be unrelated to our topic

and eliminated. As a result, just 16 records were determined to be pertinent at the end of the process. Then, 4 papers were added through snowballing, and one overlapping paper of the previous 16-papers list was removed, resulting in a total of 19 papers. The final list of primary studies included in this paper is shown in Table 4. Comparing this final list against the list of selected primary studies in [8], we notice that: *(i)* only four studies are common to the two surveys, namely [19, 23, 24, 38], while the remaining 13 papers of this review were not included in [8]; and *(ii)* 8 papers of this review appeared after the search conducted in the previous survey.

3.2 Validation with experts

In this section, we describe how we conducted the validation with experts of the artifacts we built to answer RQ1 and RQ2, a feature model and a reference architecture, respectively. The feature model and the reference architecture have been built by leveraging and organizing the data collected from the papers. During the validation with experts, we mainly focused on a few aspects that are not strongly grounded on the collected data but that included our subjective interpretation and beliefs. During the interviews, we had also the opportunity to discuss more generally about the overall work. These interviews have been driven by the questionnaires they filled out before the interview. In this sense, we went through the questions asking them to comment/justify their answers and we were able to make sure that they properly understood the questions. Then, we showed the reference architecture and used it to discuss more broadly. The interviewees were able to make their considerations on the architecture, without having to answer specific and well-formulated questions in this regard.

To select the experts, we collected the e-mails that are available in each of the 19 papers that belong to our final list. The rationale is that these authors made research and they published at least one paper in the field. The total number of authors was equal to 56 and some of the experts authored more than one paper. Then, we sent them a questionnaire. We did not show directly the figures since they are difficult to grasp without an explanation. To keep reasonable the time to answer the questionnaire, we formulated questions devoted to validate specific aspects, as discussed above. Overall, the questionnaire is composed of six questions, with the last one asking about the experts' availability for a subsequent interview. A copy of the questions contained in the questionnaire is available in the replication package.

2 authors were not reachable since their email addresses were not active anymore and we did not manage to recover them from searching on the Internet. Thus, we actually sent the questionnaire to 54 authors. Among these authors, 19 answered it (throughout the paper we make reference to the experts with the identifiers $E1,\ldots,E19$), covering a total of 15 out of 19 papers. 2 of them ($E4$ and $E8$) even replied to the e-mail providing some advice on our research. Considering their public profiles in Google Scholars, the experts span a broad range of research experience, with one expert having publications dating back to 5 years (E10), two back to between 6 and 10 years (E5 and E19), four back to between 11 and 15 years (E1, E2, E13, E17), three back to between 16 and 20 years (E6, E12, E18), four back to between 21 and 25 years (E4, E7, E8, E15), and finally five of them (E3, E9, E11, E14 and E16) had publications dating back to more than 25 years. 5 people ($E2$, $E4$, $E7$, $E10$ and $E18$) expressed interest in an interview to discuss the topic (in the last question of the questionnaire they typed their e-mails), covering a total of 7 out of 19 papers. We then arranged meetings with them.

4 TAXONOMY FOR SATF

In this section we aim at providing an answer to RQ1, i.e., **What are the main dimensions of a taxonomy for self-adaptive field-based testing?**. We start by giving a definition to SATF through the analysis of the current literature on this subject (Section 4.1). Then, we present the dimensions of the taxonomy obtained to SATF (Section 4.2) and briefly discuss their relations (Section 4.3).

4.1 Definition of SATF

In this section, we present the definitions of SATF provided by the literature (Section 4.1.1) and summarize them in order to create a single and coherent definition for this term (Section 4.1.2).

4.1.1 Definitions from the literature. Each of the works we analysed offers its own interpretation of SATF, e.g., in terms of what to monitor, what to adapt, when to adapt, how to adapt, etc. Although they do not offer a commonly accepted definition of SATF, their similarities can help to define one. The approaches in [24] and [23] modify test cases that match the specification at runtime based on system and environmental conditions. The work in [19] responds to changes in operations, operation arguments, and service composition. Likewise, the work in [29] examines modifications made to how the service is utilized or provisioned to trigger testing sessions. On the other hand, the work in [15] monitors the system to find untested configurations and activate the testing. Similarly, the work in [26] proposes an approach that reveals scenarios containing error states before they result in system failures. To test novel scenarios that appear in production but have not yet been tested, their field-ready testing uses data from the field. The observation of various instances of systems obtained from a dynamic software product line, along with their applied configurations, is taken into consideration in [31] with the purpose of predicting an up-to-date operational profile. Test cases are thereafter run incrementally based on this predicted profile. The findings in [32] support the notion that an "autonomous self-organising system must be capable of self-analysis to detect system components that are faulty". In this way, it suggests a strategy that allows the various parts of a self-organizing system to undergo testing. In passive tests, a component assesses the test subject's behaviour during typical system operation. In active tests, it creates test events and monitors the response.

The online failure prediction enables the system to anticipate adaptations and prevent actual failures from happening again. The authors of [42] use SATF to accomplish this by gathering usage statistics from constituent services and conducting online tests against these services to receive quality data only if the usage statistics are below a predetermined threshold. Then, using a composition of monitoring and testing data, failure prediction is carried out. The study in [25] suggests the MAPE-T feedback loop as an addition to runtime testing techniques. This consists of the standard Monitoring, Analyzing, Planning, and Executing stages of a SAS, which were created in this case to assist the testing activities. With their idea, they hope to argue that test cases should be viewed as "first-class entities that can evolve as requirements change and/or self-reconfigurations are applied" [25]. Additionally, they assert that test evolution is a multifaceted objective and that test cases must adapt and be safely run based on the system's present scenario. By maintaining their consistency, test cases can be reused at runtime to verify that they satisfy the specification as well as any potential conditions that would imply the need for adaptation. A strategy to assist the runtime testing of changes brought about by self-testing components in SAS is also suggested in [49]. Their research intends to make self-testing capabilities an implicit characteristic of the systems.

Activating tests on demand is another option. The research described in [47] enables the estimation of microservice architecture reliability at runtime in response to a reliability assessment request by a stakeholder. Also focusing on testing microservices, the work in [14] collects usage data during the Ops stages of a DevOps cycle and raw sessions that are automatically recorded in session logs and then analyzed to extract the workload intensity and the behaviour model of the testing session. Accordingly, to test different scenarios of a microservice-based case study application, the work in [51] defines and automatically enacts live testing based on data collected by metrics providers or external services. In [21] the authors assess the variability of an adaptive system at runtime by checking the necessity of runtime testing following the application of adaptation rules by the system. When required, their methodology provides test scenarios with sudden context changes to promote adaption responses and identify failures. The SAMBA approach, which is discussed in [39], focuses on functional and regression testing of service orchestrations during runtime with the goal of identifying errors caused by evolutionary

behaviours, such as the addition of new functionalities. The orchestration description is used to extract or change a model. Updates to the models are also triggered whenever changes in the orchestration are discovered. Their technique considers this model to automatically generate test cases.

Also, to perform regression testing at runtime, the approach proposed in [41] does not incorporate a monitor component for regression testing during runtime. Rather, it immediately derives test cases from selected successful executions that the tenant administrator has decided upon. Another approach in which the monitor component is not present is proposed in [30] to test a Service-Oriented Architecture (SOA) application. This work automatically generates test data based on the specification and input data coming from the consumer's application, which are then executed on demand. Lastly, the research in [38] suggests evaluating dynamic and distributed systems using the framework RTF4ADS that executes in the field "test cases covering only software components or compositions affected by the dynamic change" [38].

4.1.2 Definition of SATF. Based on our examination of the chosen studies, none of them clearly defines SATF, and as we have outlined, they each present a variety of approaches. In the previously referenced SLR regarding field testing [8], the definition of field testing is given as "any type of testing activities performed in the field", which is both very general and abstract. By taking into consideration the analysis of the recent literature in SATF aforementioned and the common main concepts provided by it, we adapt the above general definition as follows:

Definition 4.1 (Self-Adaptive Testing in the Field (SATF)). Self-Adaptive Testing in the Field (SATF) is any type of testing activities performed in the field, which have the capability to self-adapt to the different needs and contexts that may arise at runtime.

Such a broad description can encompass all the studies previously described: as we briefly summarised, each study instantiates the mentioned "capability to self-adapt" in various ways.

4.2 Dimensions of the taxonomy

In this section, we present the dimensions that compose the taxonomy for SATF approaches. They were extracted by looking at the papers we collected and observing what are the features that SATF approaches may have. They are described in the following subsections.

4.2.1 Monitor. Monitoring is the process of continuously acquiring, analyzing, and collecting information regarding the SUT execution [8]. In field testing, monitoring is crucial because it collects information about the activities and the states of both the SUT and environment, as well as the behaviour of the user or external actors. In SATF approaches, such data is necessary to trigger the testing process and identify and plan testing adaptation activities. However, by looking at the collected papers, we noticed that the monitor is not always present, and because of that, we have it as an optional component. This comes from the fact that the need for a monitor component depends on both the objective of the testing and the system itself. For instance, as mentioned by one of our interviewees, if we consider the reliability testing of web services, there is no need to monitor since only request-response couples are enough. Contrarily, the monitor component is essential in performance testing approaches since it observes the SUT performance details that cannot be derived by only request-response couples, such as CPU and memory consumption, for example. Also, in case the SUT is a system that evolves in operation thanks to a CI&CD process and toolchain, the adaptation of test cases might be directly triggered by a push in a repository and information can be directly retrieved without the need for a monitor.

The monitor component in our taxonomy is composed of three sub-dimensions: *Monitor Frequency*, *Monitor Scope*, and *Forseeability*.

A. *Monitor Frequency* defines whether the monitoring process is *Continuous*, that is, it is "continually collecting and processing data" [50], or *Discontinuous* (named Adaptive in [50]) so that only a few features are

observed, and if an anomaly is discovered, the monitor responds by gathering more information. The choice made in relation to this sub-dimension affects the monitoring cost and detection time. Most of the collected approaches use continuous monitors [14, 15, 19, 21, 24, 26, 29, 31, 32, 38, 39, 42, 47, 49, 51], and one of the papers in the survey use discontinuous monitor [23]. Some approaches, strictly speaking, do not use a monitor of any kind [30, 41], but they may rely on a human operator that acts as an external monitor. This is the case, for instance, in [41], where the authors count on a tenant administrator to trigger the testing process, that is, in their approach, the testing is conducted on demand.

B. The *Monitor Scope* defines the scope of the monitor component, that is, what the monitor is observing and pre-processing. The information gathered by the monitor may refer to the environment in which the SUT is executing, that is, related to *Environment Change or Evolution* [24, 29, 38, 51]. Also, the monitor scope may be defined as the result of the interaction between an external actor and the SUT, that is, *Configuration Change by User or Maintainer* [14, 15, 31, 47] or *External Component Change* [51]. Note that by the term "user" we aim to comprehend several possible stakeholders that can be involved in the engineering or the mere external usage of the system, as we discussed with one interviewee. Similarly, under the label "configuration" we also included in our classification the observation of the varying users' behaviour in using the system. Lastly, the *SUT Adaptation or Evolution* [19, 21, 26, 39, 42, 49], *Model Activities* and *Test Results* [23, 32] are clearly activities that should be observed by the monitor since we consider the execution of the testing in the field. The sub-dimensions *External Component Change*, *Model Activities*, *Test Results*, and *Environment Change or Evolution* were included in the taxonomy as a suggestion made by the experts E7, E10, E18, and both E4 and E2, respectively. However, we found no instance of *Model Activities* scope among the papers we collected. As said, the works in [41] and [30] do not present a monitor.

C. *Foreseeability* concerns whether "change can be predicted ahead of time" [16]. The approaches are categorized based on their degree of foreseeability: *Foreseen* (taken care of) and *Foreseeable* (planned for) [16]. Because we did not think *Unforeseen* (not planned for) was appropriate for testing approaches, we did not include it in the taxonomy. Most of the surveyed approaches are categorized as foreseeable [14, 15, 19, 21, 23, 24, 26, 29, 31, 32, 38, 39, 42, 47, 49]. One approach is classified as foreseen [51]. As aforementioned, the works described in [41] and [30] do not present a monitor.

4.2.2 Test Cases. Test cases are a mandatory dimension in a SATF approach. According to [56], they are traditionally defined as "a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement". In the case of SATF this definition, and in particular the part concerning the execution conditions, has also to embrace some notion of context, so that, where relevant, a test case can be adapted or validated against evolution. For instance, the approach in [25] associates test cases with utility functions in order to evaluate if the test cases need to be adapted; in the case of the approach proposed in [21], instead, field-based testing aims at validating an adaptation of the SUT, thus a test case contains "*context value inputs, which are responsible for stimulating system adaptations.*" The authors of [26] go further by defining *field-ready test cases* as composed of templates composed of three parts, namely *Precondition*, *Test Stimuli* and *Field Oracle*.

With regard to how the test cases are generated or selected (which is orthogonal to how test cases are defined and structured), we refer to the Testing Criteria sub-dimension (see Subsection 4.2.5C below).

One important component of a test case is the expected result, that, is, the Oracle. For instance, the work in [23] states that "a test case comprises an expected value and the conditions necessary for execution". However, we include the oracle in the taxonomy as a separate dimension (discussed here below), because we observed that not all papers actually consider the oracle as embedded within the test cases.

4.2.3 Oracle. We qualify Oracle as an optional dimension, in that it is now always present; when an oracle is not implemented, for example a human could play the role of the oracle. Oracles can be of two types, those that

are defined independently of test cases and that can work with various test cases, and those that are instead dependent on and associated with a specific test case. If we consider that the SUT is a web server, for example, the independent oracle can just look at the length of an answer received from the web server during testing, as well as to the similarity of the answer with a standard error message. In the work described in [26], in contrast to conventional oracles that only function with certain inputs and context, abstract test oracles that may be assessed in a variety of contexts that arise in production are defined. An example of test dependent oracle might be found in [21], where an oracle is also an object of the adaption as explained in the following (see Section 4.2.6). The idea of having the expected results, that, is, the oracle, embedded by test cases was also suggested by expert E4 during the interview to validate the proposed taxonomy.

It could be expected that, among the articles we collected, there would be works that address test oracles based on metamorphic relations, but this was not the case. The use of metamorphic testing is only mentioned as future work in [26].

4.2.4 Human Involvement. This optional dimension describes who is the agent of adaptation. There is *No Human Involvement* [23, 24, 29, 30, 32, 38, 39, 42, 49] in some approaches. However, we also found works that call for *Human Involvement* [14, 15, 19, 21, 26, 31, 41, 47, 51]. Testing Multi-Tenant SaaS is the main goal of the work in [41]. The tenant administrator (human) triggers the testing and gives instructions for the creation of test cases by selecting from a previous workflow execution. The work in [15] necessitates the assistance of developers when unknown configurations are discovered. The work in [19] demands human involvement for analysing test reports by the service provider. Lastly, the work in [31] needs humans to choose whether to trigger online tests or analyse reports. In self-adaptive approaches, it is expected the lowest degree of human intervention possible. However, in some cases, depending on the SUT, for instance, this is not reachable. For this reason, we have human involvement as optional, in case, for instance, there is no monitor and a human is responsible for activating the testing process, as the already provided examples. Another example is the case in which the testing strategy uses a human to play the oracle function, as presented in the work in [51] that uses a dashboard for the developer to visualize the outcome of executed checks and based on this, takes decisions.

4.2.5 Test Strategy. *Test Strategy* is a mandatory dimension that comprises all the elements that are related to decisions taken with respect to the testing process. It includes *Class of Field-Testing*, *Tested Requirements*, *Testing Criteria*, and *Testing Level*.

A. Class of Field-Testing - The work in [8] categorizes field-based testing approaches into *Ex-vivo* [14, 29], *Offline* [26, 49] and *Online* [15, 19, 21, 23, 24, 30–32, 38, 39, 41, 42, 47, 49, 51], depending on the timing of testing activities and whether these activities make use of the real system. *Ex-vivo* testing includes testing approaches "performed in the development environment using information extracted from the field" [8]. The approach in [29], which uses ex-vivo testing, involves carrying out online non-functional testing to gauge the web service's dependability and/or performance. *Offline Testing* consists of testing approaches "performed in the production environment on a SUT separated from the actual system" [8]. The work in [26] conducts offline testing, by launching the tests in a sandboxed environment, whereas the work in [49] performs both offline and online testing. Finally, *Online Testing* comprehends testing approaches "performed in the production environment on the actual system" [8]. Most of the collected works conduct online testing.
B. Tested Requirements - This sub-dimension classifies approaches according to the type of faults they are addressing. An approach might aim to discover *Functional* [15, 19, 21, 23, 24, 26, 30–32, 38, 39, 41, 42, 49, 51] or *Non-Functional* [14, 29, 47, 51] faults in order to fulfill the requirements. The work in [14] assesses two non-functional requirements of microservice systems, reliability and performance. On the other hand, the

work presented in [32] employs self-tests at runtime to identify a malfunction of a self-organised system under test.

C. Testing Criteria - This sub-dimension refers to which specific techniques and criteria are employed to derive and/or select test cases. Based on our overview of the literature, the Testing Criteria that have been used so far can be divided into the following categories: *Model-based* [21, 29, 38, 39, 42, 49, 51], in which the test adaptation relies on a model of the SUT and/or of the environment, and is the most commonly used technique; *Operational* [14, 26, 31, 42, 47], i.e., the testing is driven by the usage profile in operation; *Specification-based* [30, 32, 41], in which the tests are black-box functional ones and are derived from the system requirements or an informal specification; *Evolutionary* [23, 24], two papers from a common set of authors adopt a lightweight evolutionary strategy that facilitates test evolution by adapting genomes in a low-impact manner; finally, one paper simply performs *Random* testing [19], and another one applies *Combinatorial* criteria [15].

D. Testing Level - This sub-dimension aims to classify approaches according to the different levels at which the software testing is performed. In our review of the literature, we found papers that can be categorized into the following testing levels: *Unit Level* [15, 26, 32, 38, 42, 51], i.e., each module or component is tested in isolation; *Integration Level* [15, 30, 38, 39, 42], in which a group of related modules is tested in each iteration; and *System Level* [14, 15, 19, 21, 23–25, 29–31, 39, 41, 47, 49], that is the level where the whole integrated application is examined in its entirety. For instance, the work in [32] performs unit testing by using a selection strategy in which the remote test module of an Agent i decides which other agent j to monitor. On the other hand, the work presented in [30] employs integration testing to the complete testing of composing services. Finally, the approach in [24] utilizes system testing through Veritas, a framework based on the use of utility functions to guide test case adaptation when testing a smart vacuum system (SVS) application.

4.2.6 Adaptation. In an SATF approach, the *Adaptation* dimension is mandatory due to the self-adaptive nature of approaches, that is, its components may self-adapt to handle the data gathered from the field. We consider that this dimension could be subdivided into seven sub-dimensions: *Object to Adapt*, *Trigger*, *Decision Making*, *Technique*, *Adaptation Type*, *Openness*, and *Degree of Decentralisation*.

A. *Object to Adapt* - Taking into account the components of testing activity, we consider that *Test Cases*, *Oracle*, *Monitor*, or the *Test Strategy* (e.g. the test criterion, the test plan, etc.) may need to be adapted while conducting testing in the field. From the questionnaire, we can validate with experts that more than one of these components could be adapted at the same time. According to what is observed in the field, the test suite or the collection of *test cases*, can be adapted by either *changing existing test cases* (which includes also deleting existing test cases) [23, 24, 31, 49] or *creating new ones* [15, 19, 21, 26, 29, 30, 39, 41]. As mentioned above, test cases might have associated oracles, and therefore oracles should be created for new test cases or potentially updated for adapted test cases. The idea of creating new test cases involves more effort than creating them at design time, but it resolves the issue of maintaining alignment with a previously created test suite [29]. In other scenarios, it is unavoidable because test cases may become invalid as a result of system adaptation [24]. Existing test cases may also be adapted, as mentioned, e.g., in [23, 24, 31]. Some approaches also include multiple objects that need to be adapted. The work in [23], for example, adapts test cases for fine-grained adaptation and adapts the test suites and/or plans for the coarse-grained adaptation. The *Oracle*, both test-cases-independent and dependent ones, could also be *adapted* according to what is observed in the field (e.g., [21]). Different types of oracles are used by field-based techniques to determine the results of tests [8]. Oracles may be based on user-defined or QoS-defined specifications, or they may take advantage of the detection of crashes or unchecked exceptions. Only one work in our study uses the oracle as the target for adaptation [21]. In their work, oracles are built at runtime using the extended

Context Feature Model (eCFM) [20], which is used to model systems that adapt their features in response to the environment. They represent the oracles as propositional formulations that declare the correct state of the features. This is an example of oracles that are connected to their specific test cases, that is, a dependent oracle.

Another item to *adapt* could be the *Monitor*. The process of monitoring, which consists in collecting and interpreting data about the SUT execution, is extremely important to field-based testing, as this task is mainly based on the information collected by this component. We did not identify any examples of monitor adaptation in the works we collected. Despite that, we choose to keep it as an option for the object to adapt since it can be useful to adapt the monitor at runtime in accordance with the prospective evolution of the SUT. An example of monitor adaptation is the change of the monitoring frequency or the monitored parameters according to the SUT adaptation. Besides, all the experts we interviewed (E2, E4, E7, E10 and E18) stated that they agree that the monitor is a component that may be adapted. The expert E10 even says that "test cases, oracle, monitor, and/or the testing approach must be adapted to be able to spot new negative and positive behaviors of the SUT".

The *Test Strategy* may also be adapted [14, 15, 23, 32, 38, 41, 42, 47, 49, 51]. Possible adaptations include, for example, the adaptation of the test plan or timing when test cases are periodically scheduled to execution. The work in [38] adapts test selection and test placement, that is, the assignment of test components to the execution nodes, with the goal of testing dynamic and distributed systems. Another illustration is the work in [41], which adapts the test case selection and the workflow regression testing for multi-tenant Software as a Service (SaaS) to avoid expensive, time-consuming, or workflow-halting steps.

B. *Trigger* - The *Trigger* is also an important sub-dimension for the adaptation stage of SATF approaches. It aims at capturing how the adaptation is initiated and can be classified according to the strategy for *Activation* and the *Trigger Type*. A trigger indicates the events that result in the activation of field test cases. More specifically, "a trigger is any kind of event, scenario, or configuration whose occurrence leads to the execution of some field test cases" [8]. The adaptation may be triggered or activated (i) *periodically* [30], (ii) by internal events of the *SUT* or changes in its environment or technical resources [14, 15, 21, 23, 24, 29, 31, 38, 39, 42, 49], (iii) by some *policy* [19, 23, 24, 26, 51], or (iv) on a *request* of a testing session from testers, runtime infrastructure/container, etc. [24, 30, 31, 41, 47]. One of the experts we interviewed (the expert E10) suggested considering also (v) "*changes in the users' behaviour*" [14, 29] (change in a user profile). In fact, changes in the way users or external systems interact with the system under test can impact its performance and reliability. For instance, a self-testing approach can get an unintended use of the system by a user that might create problems and consequently can learn a new way of testing or a new strategy with respect to how the SUT has been tested so far. The same expert also highlighted that systems can have associated a model of the SUT (e.g. microservices and energy consumption). In this context, the model itself can trigger an adaptation in the case the model is changing a lot, for instance. This can be useful to keep the model aligned with the system, which is of key importance also to have the testing process adapting and evolving in the right direction. We also confirmed by the questionnaire sent to experts that more than one of these strategies could be used to activate adaptation.

Referring to the surveyed papers, most strategies use the SUT as a trigger, which might be an environmental or internal SUT event [15, 24], a change in a technological resource [23], or changes or adaptations made to the SUT [21, 38], among others. Aside from periodic triggers or triggers based on specific policies, adaptation can also be triggered on demand by, for instance, the testing infrastructure [24] or the tenant administrator [41]. Lastly, the adaptation trigger for the work in [32] is not mentioned.

The *Trigger Type* is related to the question "When should we adapt?". Thus, approaches are classified according to the time they adapt. This temporal aspect of the adaptation, according to [36], can be separated into two sub-dimensions: *Reactive* [14, 15, 19, 21, 26, 29, 30, 38, 39, 41, 42, 49, 51] and *Proactive* [23, 24, 31, 32,

42, 47]. By the general description in [50] and adapting it to testing, in the case of reactive approaches, the testing system responds when a change has already occurred, whereas in the case of proactive approaches, the testing system forecasts when the change is likely to occur and can anticipate its self-adaptation. Most of the works surveyed are reactive, although some are proactive. A clear illustration of a proactive approach is shown in [42], where external and local system faults are foreseen and the testing process self-adapts to address them. The work in [31], on the other hand, performs adaptations based on an estimated operational profile.

C. *Decision Making* - The *Decision Making* of the adaptation can be classified into *Static* [14, 15, 19, 21, 23, 24, 26, 29–32, 38, 39, 41, 42, 47, 49, 51] and *Dynamic*. This categorization was proposed in the taxonomy for self-adaptive software described in [50]. The *static decision-making* hardcodes the decision-making process, thus changing it necessitates recompiling and redeploying the system (or its components). The *dynamic decision-making*, on the other hand, facilitates runtime management of policies, rules, or QoS to cover a new behaviour connected to both functional and non-functional software requirements [50]. All the approaches we examined present static decision-making.

D. *Technique* - The *Technique* dimension aims at answering the question "What kind of change is needed?" [36], i.e., what adaptations are necessary based on what is observed. The study in [36] divides adaptation techniques into three categories: *Parameter* [14, 23, 47], *Structure* [15, 19, 23, 24, 26, 29–32, 38, 39, 41, 49], and *Context* [21, 42, 51]. Parameters techniques conduct the adaptation by changing parameters. Structure techniques adapt by changing the structure of the testing system. They include techniques that remove/add test cases or update test reports [19, 31], test placement [38] or test plan and identification of where to conduct the test [49]. Changes in how technological resources, environment/users, or other components relate to one another are also comprehended [36]. Lastly, context techniques deal with changes in the context, such as adapting the testing rate in accordance with the usage rate [42]. There is also the option of combining these three adaption strategies, as in the work in [23] that makes use of parameters for fine-grained adaptation and structure for coarse-grained.

   Through the questionnaire and interviews, we validated with experts the possibility of using more than one of these techniques in SATF approaches, and they were positive concerning this. One of the experts we interviewed (expert E10) mentioned that it would be important to add "test execution", since, in many cases, the way tests are executed can impact the results. For instance, running tests from the "user"'s perspective (e.g. sending the requests to the Gateway) can show different results than running tests from the "microservices" perspective (e.g. sending the requests directly to the microservice under test). The way you test the system in microservices can change the results. In fact, when it is tested from the outside, many faults can be managed and hidden by the resilience pattern. Testing directly the microservice can instead spot the errors.

E. *Adaptation Type* - The *Adaptation Type* sub-dimension describes how the application logic and adaptation mechanism are separated [50]. In this way, approaches can be divided into *Internal* [14, 15, 19, 21, 23, 24, 29–32, 38, 39, 41, 42, 49, 51] or *External* [26, 47]. Internal approaches merge the application logic and adaptation mechanism. It can be useful for handling local changes, but it may present problems with scalability and maintainability. External approaches employ an external adaptation engine that comprises the adaptation processes. The key benefit of external approaches is the reusability of the adaptation engine. Most of the approaches we surveyed are internal. An example of an external approach is provided in [47], which employs an external monitor to provide information useful for revising the test profile.

F. *Openness* - *Openness* indicates the degree of openness of the set of adaptive actions [50]. A *close-adaptive* system [14, 15, 19, 23, 24, 26, 29–32, 38, 39, 41, 42, 47, 49, 51] has a predetermined number of adaptive actions; hence, no additional action may be added at runtime. In contrast, an *open-adaptive* system [21] can be expanded to allow for the addition of new adaptive actions, which opens the door for the inclusion of

additional entities in the adaptation mechanism. With the exception of the work in [21], which is open, all of the surveyed approaches are close. The adaptation in their work is based on adaption rules, which may alter over time.

G. *Degree of Decentralisation* - This sub-dimension regards the level of decentralisation held by the adaptation logic [36], without considering the application logic. A *centralised adaptation* [14, 19, 21, 23, 24, 26, 29–31, 38, 39, 41, 42, 47, 49] logic is advised for small systems with minimal resources to handle. In contrast, a *decentralised adaptation* [32, 51] approach would be useful for dividing tasks and enhancing the ability to adapt complex systems with a lot of components to manage. Finally, *hybrid adaptation* [15] approaches divide the functionality of the adaptation logic into separate sub-systems or add central components to decentralised approaches. Most of the approaches in this work are centralised since the testing system frequently consists of a small number of components. The work in [32] is an example of decentralised testing and it proposes to test a self-adaptive system made up of a number of independent agents that can test one another. Instead, even though the approach in [15] performs decentralised in-vivo tests, it is considered as hybrid since the overall testing operations are coordinated by a central server.

4.3 Relations among the dimensions

In this section, we provide an overall view of the various dimensions described in Section 4.2. Figure 3 puts together the various dimensions in a feature model that summarizes the components of a SATF approach and their relationships. It is important to notice that, in the figure, the mandatory components are marked with • and the optional ones with ○. In the feature model, the *Alternative* symbol indicates that only one of the children can be defined. On the other hand, the *Or* symbol determines that more than of the children can be defined. When there is no symbol in the connection between components, it would be equivalent to a Boolean *And* symbol, that is, if the father is defined, so all the children must be. It is important to highlight that, both *Alternative* and *Or* symbols also mean that once the father has been defined, at least one child also must be. As described above, and visible in Figure 3, some of the features are alternatives, some mandatory, and some optional. Indeed, not all the possible configurations are valid. In the following, we describe the constraints that should be valid[4]:

- `SATF.Adaptation."Object to adapt"."Oracle Adapt."."Independent Oracle Adapt."` $\implies$ `SATF.Oracle."Independent Oracle"`: since an oracle is optional, Independent Oracle can be an object to be adapted only when an Independent Oracle is present.
- `SATF.Adaptation."Object to adapt"."Oracle Adapt."."Dependent Oracle Adapt."` $\implies$ `SATF.Oracle."Dependent Oracle"`: since an oracle is optional, Dependent Oracle can be an object to be adapted only when a Dependent Oracle is present. As highlighted by one of the interviewees, in practice, the dependent oracle is adapted when the associated test case is adapted. Since in principle it is admissible to adapt a dependent oracle without updating the associated test case, we add no further constraints for this concern.
- `SATF.Adaptation."Object to adapt".Monitor` $\implies$ `SATF.Monitor`: since a monitor is optional, Monitor can be an object to be adapted only when a Monitor is present.
- `SATF.Adaptation.Trigger.Activation.On-demand` $\implies$ `SATF."Human Involvement"`: since the human involvement component is optional, the activation of adaptations could only be performed on-demand in the presence of a human.
- `SATF.Monitor."Monitor Scope"."Configuration Change by User or Maintainer"` $\implies$ `SATF."Human Involvement"`: since the human involvement component is optional, the monitor scope could be defined by configuration change by user or maintainer with the presence of a human.

[4]The character "." is used to navigate the feature model in Figure 3. Also, we assume that a feature is TRUE when it is active. For instance, `SATF.Monitor=TRUE` when a monitor is present, `FALSE` otherwise.

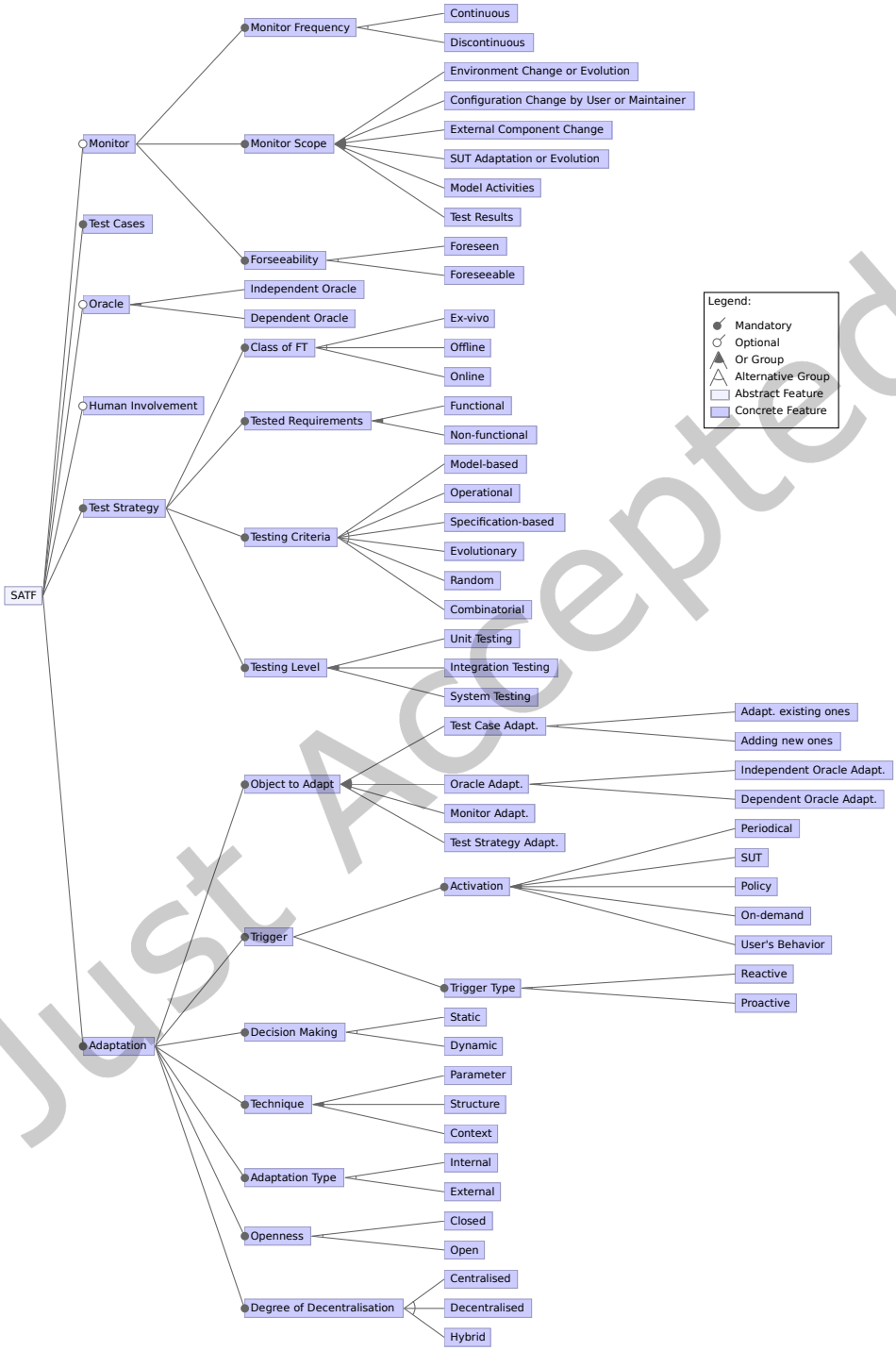

Fig. 3. Feature model of the dimensions.

- Each test case should embed a (dependent) oracle or should be covered by an independent oracle or by a human acting as an oracle. Consequently, in this case, `SATF.Oracle."Independent Oracle" OR SATF."Human Involvement" = TRUE`, i.e., one of the two should be present.

5 REFERENCE ARCHITECTURE FOR SATF APPROACHES

In this section we aim at providing an answer to RQ2, i.e., **What are the main components of a reference architecture for self-adaptive field-based testing approaches?** A reference architecture is a general architecture that is used as a foundation for the design of concrete architectures within a given context or application domain [11]; then, a concrete architecture can be considered as a specific instance of a software reference architecture. For instance, the work in [45] presents a reference architecture for connected vehicles. In this paper, we contribute a reference architecture for SATF approaches. It has been built by taking into account the primary studies that have been identified and analysed in the literature review. Since we made an effort to generalize and interpret the data collected by analysing the primary studies, we also validated the reference architecture with our interviewees. Then, we improved the architecture according to the received feedback. Further details about the followed research methodology can be found in Section 3 and the results of the validation are described in Section 7.

An architecture can be described in various ways and it can be organized in various views and viewpoints [1]. For instance, software architectures can be described as "boxes and lines" when focusing mostly on the component and connector view, or they can be represented by referring to architectural patterns and tactics. We will first describe the main architectural decisions and then provide a component and connector view, which permits us to identify the main components and connections among them. The component and connector view is described in Figure 4. When describing the main architectural decisions, we will also describe how they have been defined, relate to the feature model described in the previous section, and have been influenced by the validation with experts. The main architectural decisions are described in the following:

AD1: *Adopt the MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) feedback loop as style.*
Rationale: The MAPE-K loop is the most influential reference control model for autonomic and self-adaptive systems [34]. SATF approaches might have or not have an explicit MAPE-K loop in their architecture, however, conceptually they will have the phases of the MAPE-K loop. It is important to highlight that, as often happens in self-adaptive systems, SAFT approaches can have humans in the loop. This means that humans might be involved in various ways in each of the MAPE-K phases. We do not show this aspect in the figure to avoid making it unnecessarily complex. However, we have examples of human-in-the-loop involvement in the surveyed papers [15, 19, 31, 41] and it has been suggested by one of the interviewed experts (expert E18).

AD2: *The monitor component should be able to gather and pre-process the System Under Test (SUT), from the SUT model, from entities in the execution environment that can affect the SUT, or from external actors, which can be users or maintainers of the system as well as other systems or devices, e.g. in an Internet-of-Things (IoT) setting.*
Rationale: From the feature model in Figure 3 we can see that the monitor scope includes: (i) environment change or evolution, (ii) configuration change by user or maintainer, (iii) external component change, (iv) SUT adaptation or evolution, (v) Model activities, and (vi) Test results. As described also in the previous section, points (i), (ii), (iii), (v), and (vi) have been recommended by the experts we interviewed. One of the experts (E10) also recommended considering that the SUT can have a model associated with it. In this case, to avoid misalignments of the SUT with its model, which can be dangerous and can also lead the self-adaptive test to diverge from the SUT, there can be strategies to trigger a test of the SUT in specific situations, e.g., when the model of the SUT is changing too often. In the case of the SUT being a self-adaptive system, the monitor of SATF can be profitably connected with the monitor of the self-adaptive system

under test. In the case of the SUT is evolving, e.g. through a release in a Continuous Integration and Continuous Deployment (CI/CD) setting, in the case of the user has defined a new configuration or in the case of a maintainer operated changes, the monitor can also become an optional component [30] (or simple component) since the trigger of adaptation might come directly from a CI/CD release [51] or a user action [14].

AD3: *The analysis component should decide which adaptation should be planned and, then, it should analyse (i) environmental changes, (ii) external actors' interaction changes, and (iii) SUT changes.* External actors' interaction changes can concern changes in user behaviour that should trigger a new testing session, e.g. concerning a user profile, or of an external system, e.g. caused by a new version of an external system, or even an attack from an external system (suddenly behaving unusually).
Rationale: This component is aligned with the monitoring component since the gathered information needs to be analysed. The connection between the monitor and the analyse phases is bidirectional since the analysis might trigger the need for further data to be monitored. The analyse phase might require information from the database of test cases and oracles. The Analysis phase should also be able to understand when there is a need for testing discrepancies between SUT and its model. This type of testing has been recommended by one of the experts we interviewed (E10). Then, the request for adaptation is analysed and different types of adaptation are triggered, according to the specific adaptation that is requested. Referring to Figure 4, this aspect is hidden within the *c* label connecting the Analyse and the Plan phases, however, the analysis phase supports decision-making on the necessity of self-adaptation of the SATF approaches and identifying the components within the Plan phase that should be involved.

AD4: *The adaptation of the SATF concerns the adaptation of test cases or generation of new ones, oracle, monitor, and test strategy.*
Rationale: This is aligned with the feature model in Figure 3, where we describe the object of adaptation in equal terms. This decision impacts the *Plan* phase since there is a need for planning the adaptation of each of these parts. As shown in Figure 4, the *Plan* phase generates suitable actions to affect the SUT according to the supported adaptation mechanisms and the results of the Analysis of adaptation. The Plan phase might require information from the database of test cases and oracles, and from the model of the SUT. This phase contains five components, each specialised in specific adaptation actions: (i) Plan for test cases generation, (ii) Plan for test cases adaptation, (iii) Plan for monitor adaptation, (iv) Plan for strategy adaptation, and (v) Plan for oracle Adaptation. It is worth mentioning that we have no paper showing the need for adaptation of the monitor, but we found it useful to give the possibility of adapting the monitor and the experts confirmed that, indeed, the monitor can be one of the objects of adaptation. As already discussed in the item of oracle adaptation, it can also happen that a triggered change might cause more than one adaptation, e.g. adaptation of the test strategy and test cases, as in [15], in which tests are only executed when an untested configuration is observed and new test cases are generated. As shown in Figure 4 we have a dedicated component in the Execute phase to manage the SAFT adaptation and it has connections with each potential object of adaptation.

AD5: *The Execute phase should perform two different types of activity: (i) the adaptation activities, as in AD4, and (ii) testing activities.*
Rationale: Concerning (i), the execute phase implements actions with the goal of adapting the specific components of the SATF approach according to the plans identified by the plan components. As shown in Figure 4 and as discussed in AD4, the SATF adaptation component is responsible for the adaptation of the Monitor, Test strategy, Oracle, as well as updating existing test cases or creating new ones, according to the plans received by the Plan components. Concerning (ii), it is in charge of testing the SUT via the Test execution component, which exploits the test strategy, uses the test cases in the database and exploits the oracle.

AD6: *Knowledge that is needed by the various phases is included in the K part of the MAPE-K loop, according to the MAPE-K loop style.* The Knowledge component enables data sharing, data persistence, decision-making, and communication among the components of the feedback loop.
Rationale: Besides the computational states of the various components and other information that might be needed, it consists of a database of test cases and oracles, when they are associated with specific test cases. Test cases and potential oracles are used to test the SUT; they can be updated or new test cases (with associated oracles) can be added as an effect of the adaptation. Moreover, the knowledge contains also a Model of SUT since it might be needed by all phases of MAPE-K loop. This was suggested by an expert (E10).

It is important to highlight that the figure shows a reference architecture including the components that are conceptually required by SATF approaches. In specific cases, some of the components in the reference architecture might be integrated into components of the SUT. For instance, when the SUT is a self-adaptive system, it will probably implement a MAPE-K loop or a similar loop. Then, in the case in which the tester has available the code of the SUT or has available APIs to interact with the various components responsible for the adaptions of the SUT, e.g. monitor, analyser, planner, and executor, the tester might implement the self-adaptive testing approach by profitably exploiting or reusing parts of these components. In these cases, it might be interesting to refine the architecture in Figure 4 by identifying a better integration among the two MAPE-K loops, i.e., the loop of the SUT and the loop of the testing approaches. For instance, the monitoring component of the testing approach might reuse the monitoring APIs of the SUT to sense the SUT environment, exploit the knowledge of the SUT, or even profitably interact with its analyse component. However, since there can be too much variability and uncertainty in the architecture (e.g. a self-adaptive SUT might even implement an architecture not compliant with the MAPE-K loop style), the development aspects, as well as the accessibility of the SUT or its APIs, we decided to provide a general architecture assuming that the SUT is a black box component. Then, the reference architecture might be considered as a conceptual model and we delegate to the developers specific testing approaches improvements and optimizations that might become possible when targeting a specific class of SUTs.

6 CHALLENGES

In this section, we attempt to respond to RQ3, i.e., **What are the known gaps/challenges in self-adaptive field-based testing?**, and highlight the major research challenges and gaps in SATF. We here provide a list of these challenges/gaps and then compare them with the ones we found in the related literature (previously discussed in Section 2).

6.1 Uncertainty

Uncertainty is one of the most challenging issues in SATF. This challenge may result from the fact that the system being tested might encounter various operational circumstances that are challenging to foresee [24], and of course, the testing system itself reflects such uncertainty. For example, there may be an exponential number of setups when testing mobile applications [15]. The study in [23] also highlights how difficult it is to adapt testing to a self-adaptive system as it reconfigures. In addition, interactions that were not anticipated at design time might be discovered at runtime [38], and the testing methodology should be equipped to manage them. In [25], a specific illustration of the repercussions of uncertainty is given. They demonstrate how changes to the environment or to the requirements can affect how well test cases and/or oracles behave. They also bring up the issue of when to test and which system properties should be monitored by the testing methodology.

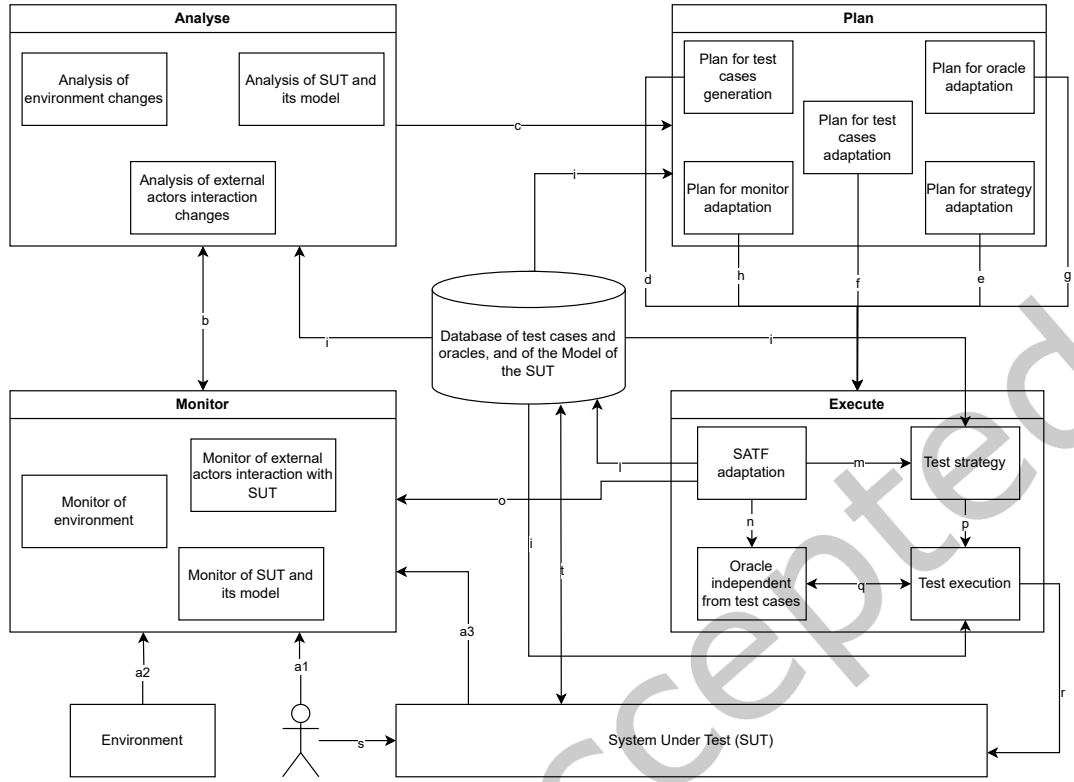

Fig. 4. Reference architecture for SATF approaches.

6.2 Overhead

Several testing strategies are concerned with the overhead in terms of memory, network, and execution time [6]. However, since the testing procedure may add some overhead to the SUT in SATF, solving this issue is even more significant. It results from the possibility that self-adaptive testing may use resources to make adaptations and cause an overhead that could impair the system's performance. The overhead incurred should be as minimal as

possible to make this form of testing acceptable to the end-user [15]. The method in [15] examines the overhead imposed by the monitoring stage and determines that it is imperceptible to negligible in their case. The research in [38] provides a technique for testing dynamic and distributed systems and determines that the overhead, in terms of execution time and memory usage, is comparatively modest and bearable, especially if dynamic adaptations are not frequently requested. The length of the test sequences might be a crucial element in determining the overhead. Small test sequences might incur little overhead, whereas lengthy test sequences may have a significant impact on the system execution [21]. The strategy outlined in [25] emphasizes the significance of striking a balance between increasing test coverage and reducing test overhead. Their strategy organizes the test execution schedule so that testing doesn't adversely effect system performance. For the strategy described in [29], the equilibrium between the profit generated and the cost incurred is left as future work. Based on the overhead in terms of memory usage and execution time, the research in [23] comes to the conclusion that their SATF approach had a considerable influence on system execution time but had a minimal impact on memory. Ultimately, the research in [30] states that the user's response time has increased as a result of an overhead that they are unable to handle.

6.3 Human Intervention

In general, self-adaptive systems might benefit from human involvement, since human operators can complement the capabilities of systems [40]. In the context of SATF, having the human in the loop can be crucial, e.g. when testing critical systems. Instead, for some approaches, it could be essential to minimize human involvement because it can incur expenses that can be avoided. In fact, having SATF approaches totally self-adaptive can lower costs associated with human intervention, both financially and in terms of computer resources. Several of the surveyed approaches require manual intervention, e.g., [15, 19, 31, 41].

6.4 Test Isolation

Test isolation is a well-known problem for any field testing strategy, as noted in [8]. The idea behind this term is that testing should not be obtrusive, i.e., "should not interfere with the processes running in production and their data" [8]. Since testing process adaptations could give unintended access to critical system components, this challenge might become even more significant in SATF. Field testing approaches must offer a plan to reduce the system's sensitivity to its execution at runtime, which lowers the risk of negative effects on the system and its environment [21]. The system's sensitivity indicates which testing operations, when carried out, interfere unfavorably with the environment or the system that is now running [28]. The work in [25] isolates specific test cases to stop failures from impairing the functionality of the real system. Additionally, runtime adaptations of component-based systems may cause new defects to appear, which could cause malfunctions and put the execution of the system in an unsafe state [38]. Even though the work proposed in [29] does not address this issue, it does mention that online testing sessions may affect how the services function or, in some cases, risk the proper running of the service. The work in [41] supports the idea that when testing in a production environment, care should be taken to ensure that the operational database is not compromised. Modifying mission-critical data or sharing it with unauthorized parties are two examples of this unwanted impact. Lastly, the work in [15] isolates the runtime testing session from the regular user session by using managed profiles[5].

6.5 Summary of the Main Challenges and Other Challenges

The preceding subsections significantly compile the main challenges when performing SATF. A summary of these challenges is presented in Table 5. In this table, we tackle the main causes/origin of the challenge, its impact on the testing process, and the solutions proposed by the papers we collected in order to address them.

[5]https://source.android.com/devices/tech/admin/managed-profiles

Table 5. Summary of the main challenges in SATF

Challenge	Cause	Impact	Possible solutions
Uncertainty	SUT might encounter various operational circumstances that are challenging to foresee [24]. Ex: An exponential number of setups when testing mobile applications [15].	Difficulty to adapt testing to a self-adaptive system as it reconfigures [23]. System fails due to configurations never tested [38]. Environment or requirements changes impact test cases and/or oracle behavior [25]. Difficult to know when to test and which SUT properties should be monitored [25],	-
Overhead	Self-adaptive testing may use resources to make adaptations.	Cause an overhead that could impair the system's performance.	The overhead incurred should be as minimal as possible to make this form of testing acceptable to the end-user [15]. Examine the overhead imposed by the monitoring stage in order to guarantee that this is imperceptible [15]. Make dynamic adaptations not very frequent [38]. Small test sequences might incur little overhead [21]. Organize the test execution schedule so that testing doesn't adversely affect system performance [25].
Human Intervention	Human operators can complement the capabilities of systems [40], when testing, for example, crucial systems.	It can incur costs that can be avoided, both financially and in terms of computer resources	Minimize human involvement, that is, make the approach as self-adaptive as possible.
Test Isolation	Both testing process and the SUT are running under the same environment	The testing may be obtrusive, that is, could interfere with the processes running in production and their data [8]. Testing process adaptations could give unintended access to critical system components. Runtime adaptations of component-based systems may cause new defects to appear, which could cause malfunctions and put the execution of the system in an unsafe state [38]. Online testing sessions may affect how the services function or, in some cases, risk the proper running of the service [29]. Operational database may be compromised [41]. Modifying mission-critical data or sharing it with unauthorized parties are two examples of this unwanted impact [41].	A plan to reduce the system's sensitivity to its execution at runtime, which lowers the risk of negative effects on the system and its environment [21]. The work in [25] isolates specific test cases to stop failures from impairing the functionality of the real system. [15] isolates the runtime testing session from the regular user session by using managed profiles.

Nonetheless, it is important to point out that further more specific and minor issues related to SATF were also discovered, such as:

- proper reaction to identified malfunctions [32];
- provisions for reliable traceability between test cases and requirements [25];
- constraints arising from the specific techniques employed [19, 31] (e.g. with respect to the complexity of data type handled);
- the use of test cases results to improve reliability and handling different types of applications [47];
- building strengthening oracles and increasing the level of automation of the approach [26];
- modeling dependencies between services and versions, assumption that all changes are forward and backward compatible, and that provisioning and load balancing service instances is handled by an external component [51];
- specification of adaptation constraints [25]; and
- strategies to make test events indistinguishable from normal events [32].

6.6 Comparison with challenges highlighted in related surveys

In Section 2 we discussed some existing reviews addressing topics related to our study, and anticipated some relevant challenges they highlighted. After our review of primary studies, we can now briefly discuss the challenges that we identified in this study against some of the challenges listed in Section 2.

The uncertainty challenge (Section 6.1) includes many of the challenges identified in [54] and presented in Section 2, but from a different perspective. In fact, the challenges in that work surveying approaches testing adaptive systems are basically related to uncertainty and they do not identify the following challenges that get probably much more importance when the test is performed in the field.

When looking to the challenges reported in [8], it is interesting to observe that the first challenge the authors report is about *Generating and implementing field test cases*. The authors highlight that there is a lot of uncertainty when generating test case before going in the field and they should adapt to the production environment. Self-adaptive testing in the field will solve many of these issues, as also envisioned by the authors, which envision sophisticated techniques for an opportunistic generation of test cases based on the characteristics of the underlying production environment. Another challenge highlighted in [8] is *Isolation Strategies*, which is related to Test Isolation described above. The remaining challenges, i.e. *Oracle definition*, *Security and Privacy*, *Orchestrating and Governing Test Cases*, and *Challenging Domains* are not explicitly mentioned in the papers we analysed, however, these are important challenges that look relevant also for SATF approaches.

As discussed in Section 2, the work in [48] presents main challenges of a specific class of self-adaptive systems. Our uncertainty challenge (Section 6.1) includes many of the highlighted challenges since it may affect the SUT in various ways. For instance, due to uncertainty the oracle problem becomes more challenging. On the other side, having testing self-adaptive itself permits to adapt towards the changes of the SUT over time. This permits also to tune overtime the testing approaches towards dealing with concurrent faults that can mask each other and nondeterminism. Moreover, having the testing approach self-adaptive, enables to define specific test cases that should be executed during the adaptation and reconfiguration of the SUT to check safety and security constraints. As future work we aim to investigate more in depth the *dividing and conquer* and the *hardware* challenges proposed in [48], since we could not identify them as important ones in the data we collected.

7 VALIDATION

In this section, we discuss the validation of the proposed taxonomy and reference architecture conducted with experts through a questionnaire and further discussions by online meetings. We first sent an e-mail to all of the 56 authors of the primary studies we collected. It is important to notice that one same person may be the author of

more than one work. For 2 of them, the e-mail bounced and we were not able to find an alternative valid address. Among the 54 authors that in fact received the e-mail, 19 expressed interest in participating in the questionnaire survey. These are the people we refer to here as the experts. The questionnaire contained five main questions and a last question asking whether the respondent was available for an interview during an online session. 5 experts (E2, E4, E7, E10 and E18) answered positively to the latter point. The idea behind the questionnaire was validating the structures in the feature model that were not solidly grounded on the data, but that we considered reasonable, whereas the interviews aimed at getting further feedback and additional comments.

In the *first question*, we asked the experts whether they think that a monitor should be optional. In other words, we wanted to validate with experts if they believe that a SATF approach may contain or not the monitor component. Also, for all the questions, we asked them to comment their opinion. 9 people agreed (E1, E3, E7, E10, E11, E13, E14, E16, E18) with this statement. Among the comments, the most interesting ones reported that they agreed with this statement since "the need for a monitor component depends on both the objective of the testing and the system itself"(E10) and that "the outcomes of one test set execution can be used to gradually improve/tune the testing strategy for subsequent executions"(E18). We believe that this is enough to give the possibility to have the monitor component as optional, since the respondents that agreed had in mind scenarios in which the monitor can be optional. 7 people disagreed with the statement (E2, E4, E5, E8, E12, E15, E17) . Among these respondents, the main comments are that in a SATF approach "there should be a monitor (even if hidden or called differently)"(E2), and that in a self-adaptive approach, adaptations should be based on something that is observed, so there is a need for a monitor (E4). Also it was suggested that if the monitor is not present, the approach should contain any other mechanism for detecting changes that it has to adapt to. In fact, we have these other *mechanisms* in our taxonomy and reference architecture. Finally, a last comment said that if an approach does not have a monitor, it is not fully self-adaptive. However, we have a literature of human involvement (e.g. human-in-the-loop or even human-on-the-loop) in self-adaptive systems that accepts the human acting as monitor, e.g. [40]. 3 people neither agreed nor disagreed (E6, E9, E19). We noticed that, for several respondents, this question was not completely clear. So, through the interview, we had the opportunity to better explain the question and guarantee that they properly understood it. Then, after further discussions during online sessions, all the interviewed respondents agreed with the statement that a monitor should be optional, even those that did not agree before.

In *question two*, we provided a list of possible events a monitor could expect in order to activate the testing (i.e., SUT adaptation, SUT evolution and/or configuration change made by user) and asked them if they agree that more than one of these events could be monitored by the same SATF approach. 15 people agreed (E1, E2, E3, E5, E7, E9, E10, E11, E12, E13, E15, E16, E17, E18, E19) with this statement. Among the agreeing respondents, the most interesting comments made suggestions on what else could be monitored in a SATF approach. Their suggestions were valuable and very appreciated, resulting in the incorporation of new features into the feature model (e.g., Configuration change by maintainer (E4), External Component Change (E7), Environment Change or Evolution (E4 and E2), Test Results (E18) and Model Activities (E10)) and also into the reference architecture, as we explain later in this section. 2 people disagreed (E6, E8) with the statement. One of them (E6) seemed not to understand the question properly since his/her comment said that a single component adaptation should also be considered, what we already do here. The other respondent (E8) made a suggestion for including "requirements changes" to the list of possible events in the statement. We did not add any new item to the feature model based on this comment since the monitor is already capturing when these changes in the requirements result in SUT changes. Similarly, a person that neither agreed nor disagreed (E4) with the statement also wanted to make a suggestion regarding the inclusion of "changes operated by maintainers and/or evolution of the environment". The other person (E14) stated that "any of them can in principle active testing as they can provide complementary information". The discussion regarding this question during the interviews was extremely fruitful, since it gave us the opportunity to include their feedback and then refine both the taxonomy and the reference architecture.

In *question three*, we listed the possible objects to be adapted (i.e., test cases, oracle, monitor, and testing strategy) in a SATF approach and asked them whether they agree with the assertion that more than one of these objects could be adapted by the same SATF approach. 18 people (E1, E2, E3, E4, E5, E7, E8, E9, E10, E11, E12, E13, E14, E15, E16, E17, E18, E19) responded it positively. We had a comment (E5) saying that these adaptations have different natures and are all relevant. Another (E14) said that even though their combination is possible, in real cases it may not be feasible. We also received the suggestion (E16) of including "test plans" as a possible object to be adapted, but this object was already included in the "testing strategy" object. Another interesting comment (E17) said that "the more objects changed at once, the riskier the adaptation itself". One person (E6) disagreed with the statement, but we did not have the opportunity to discuss the reasons for that. A suggestion coming from the interviews (expert E4) that we incorporated into the taxonomy and reference architecture was the idea of having two different oracles: the dependent and the independent one.

In *question four*, we mentioned the possible strategies to trigger adaptation (i.e., periodically, by changes in the SUT, through a policy, and on-demand) and questioned if they agree that more than one of these strategies could be employed by the same SATF approach. 15 people agreed (E1, E2, E3, E4, E5, E6, E7, E9, E11, E12, E15, E16, E17, E18, E19) with the statement. Among these respondents, an interesting comment (E18) brought the idea that these strategies "are not fully orthogonal", and that they could be combined, for instance, by using "a policy to periodically perform adaptation of the test suite". 4 people neither disagreed nor agreed (E8, E10, E13, E14). One person (E10) among those suggested the inclusion of "changes in the users' behavior", meaning that "the way users interact with the system under test can impact its performance and reliability" and for this reason, could activate adaptation. The interviews were very useful to make clearer this question for the experts, since it may not be completely self explanatory. Once we could clarify it, all the interviewees agreed with the statement.

Finally, in *question five*, we listed the possible techniques (i.e., changing test parameters, the test structure and the test context) that could be employed in order to perform adaptation in SATF approaches. Then, we asked the experts if they agreed that more than one of these techniques could be applied by the same SATF approach. 13 people agreed (E1, E3, E5, E6, E7, E11, E12, E13, E15, E16, E17, E18, E19) with the statement, 5 people neither agreed (E2, E4, E9, E10, E14) nor disagreed, and 1 person disagreed (E8). The comments coming from this question were more related to the fact that they either did not understand the question or did not understand the difference between this question and question three. An interesting comment was that the respondent was "not sure if test structure is fundamentally different from test parameters". All of these points could be better discussed during the interviews. With respect to the difference between this question and question three, we explained that they present different perspectives regarding the adaptation. In question three we simply look at what is the object adapted by the approach. On the other hand, question five focuses on the technique that is used to perform adaptation, that is, according to what is observed, what kinds of adaptation are necessary.

Table 6. Number of responses for each question in the questionnaire.

Question/ Response	Agree	Disagree	Neither agree nor disagree
Q1	9 (E1, E3, E7, E10, E11, E13, E14, E16, E18)	7 (E2, E4, E5, E8, E12, E15, E17)	3 (E6, E9, E19)
Q2	15 (E1, E2, E3, E5, E7, E9, E10, E11, E12, E13, E15, E16, E17, E18, E19)	2 (E6, E8)	2 (E4, E14)
Q3	18 (E1, E2, E3, E4, E5, E7, E8, E9, E10, E11, E12, E13, E14, E15, E16, E17, E18, E19	1 (E6)	0
Q4	15 (E1, E2, E3, E4, E5, E6, E7, E9, E11, E12, E15, E16, E17, E18, E19)	0	4 (E8, E10, E13, E14)
Q5	13 (E1, E3, E5, E6, E7, E11, E12, E13, E15, E16, E17, E18, E19)	1 (E8)	5 (E2, E4, E9, E10, E14)

Concerning the comment that connects test structure and test parameters, we had also the opportunity to explain the differences between them since this seemed to be unclear by only reading the question. Table 6 presents the proportion of responses for each question in the questionnaire.

During the interviews with experts, we were also able to present the reference architecture that is based on the data we gathered in this research. We presented them with an initial version of the architecture, generated based on the components found in the collected works. Then, we provided a brief explanation of each of the components of this initial architecture. Finally, we allowed them to make comments, suggestions or clarify their doubts about what we presented. The interviewees provided several useful comments with respect to the architecture. Besides, many of the suggestions regarding the feature model were also incorporated into the architecture. The changes in the reference architecture suggested by the interviews are summarized as follows:

- Inclusion of the environment component;
- Inclusion of external actors (humans and/or other systems);
- Inclusion of the SUT model;
- Monitoring of the environment component;
- Monitoring of external actors interaction with SUT;
- Monitoring of the SUT model;
- Monitor divided into several sub-components according to what is being monitored;
- Analysis divided into several sub-components according to what is being analysed;
- Bidirectional communication between "Analyse" and "Monitor" components;
- Distinction between dependent and independent oracles; and
- Database being explicitly shared with all the main components.

Based on the conducted questionnaire and interviews, we concluded that the taxonomy (including the feature model) and reference architecture we built are good representatives for SATF approaches and that we are heading in the right direction in order to establish common definitions and terminologies to this still immature topic.

8 LIMITATIONS AND THREATS TO VALIDITY

This research has been performed by following well-established guidelines for empirical studies in software engineering including systematic studies [35], surveys [33], and interviews [52]. We identified the main threats to validity and mitigation strategies according to [58], as described in the following subsections.

8.1 Threats to internal validity

Threats to internal validity are primarily associated with the design of the study since they aim to understand to what extent claims are supported by the obtained data [58]. We defined and followed some strategies to mitigate these threads to validity. First, we defined a research protocol to conduct this study employing well-established guidelines for systematic studies, surveys, and interviews on software engineering. All the decisions required during the definition of the protocol have been made by reaching a consensus among the authors.

Moreover, we performed surveys and interviews to validate the obtained results. For the questionnaire we used a mix of close-ended and open-ended questions to keep the respondents focused and motivated. The interviews helped us collecting qualitative data that nicely complement the data collected via the questionnaires. We are confident that the subjects selected for filling the questionnaire and for the interviews are experts and qualified. In fact, we contacted all the authors of papers included in the literature review, i.e. those persons that demonstrated knowledge and experience in the field. One potential threat to validity here is that among the 19 experts who made themselves available for the validation, a few people have been co-authors in the past of the third author of this paper. However, either the collaboration covered a quite different topic than SATF (3 experts), or the only paper co-authored was the secondary study in [8] (3 more experts): in other words, it is not granted that such

past co-authors would bring an opinion about SATF that is biased towards the authors' opinions. This threat was anyhow in part mitigated by using a standard questionnaire survey and having the interviews led by the first author, who had no previous acquaintance with them.

8.2 Threats to construct validity

Threats to construct validity focus on the relation between the theory and the observation [58]. We are confident that the papers included in the list of primary studies are representative of the population defined by the research questions since we followed a well-defined protocol. We mitigated the risks associated with data extraction by having at least two authors' looking at each paper. Moreover, we discussed all papers whose inclusion was not so evident during plenary meetings. A potential threat to construct validity in this work is the fact that occasionally in the self-adaptive systems community, tests that are "in the field", may be referred to simply as "tests", that is, the "online/runtime/in the field" characteristic is implicit. The search string we use does not include "test*" without an online/runtime/in-the-field qualifier. However, we believe that we mitigate this problem by performing snowballing. In this way, this type of work would also be able to be included. Another possible threat, that is related to the interview, is hypothesis guessing or confirmation bias, happening in case respondents adjust their answers with the main goal of the study. In order to mitigate such a threat, we posed the questions objectively and used references to relevant sources.

8.3 Threats to external validity

External validity threats relate to the generalisability of the final observed results and outcomes of the study [58]. Specifically, since this study investigates a subject not yet well-established in the literature, we loosened some of the inclusion and exclusion criteria often used in similar studies. This enabled us to collect perspectives and definitions on the subject, even if not supported by a rigorous validation. Due to this, we also include a few opinion papers in our collection of primary studies, which may contain material that is not backed up by empirical research.

Moreover, we also performed both backward and forward snowballing to identify more works. Indeed, the taxonomy and the reference architecture should be considered as living documents that will be refined when the field will reach a more stable maturity.

8.4 Threats to conclusion validity

Threats to conclusion validity refer to the relationship between the extracted data and the obtained findings, and they affect the credibility of the conclusions drawn from the extracted data [58]. We are confident that the taxonomy and the reference architecture are representing all the works considered. In fact, we followed a well-defined process to construct them, we clearly identified our interpretations and hypothesis, and we validated them with the experts, both with questionnaires and interviews. Moreover, we have documented every step of our research and provided a public replication package to ensure transparency and replicability.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we presented a new category of testing called self-adaptive testing in the field. The need of this new category of testing comes from the observation that modern systems are more and more required to evolve after they are released and are in operation. This is the case of self-adaptive systems but also of systems that are continuously updated to new versions as required to fix bugs as well to offer new functionalities to their customers. Therefore, testing should be performed also when the systems are in operation in the field: this calls for field-based testing. Moreover, the test itself should adapt over time in an autonomous way to meet the changing and evolving systems over time. In this paper, we offered a taxonomy of self-adaptive testing in the

field together with a reference architecture of SAFT approaches. The taxonomy and reference architecture have been defined by surveying the literature and by collecting feedback and contributions from experts in the field through questionnaires and interviews. We also provided a wealth of challenges to be solved in SATF approaches, which, when compared to the amount of papers collected (i.e., 19 papers), endorse how this topic is immature and not yet recognized by software testing researchers as a self-standing emerging discipline.

We believe that our contributions of this paper can help shaping the research in the future of this important category of testing. This work might also help practitioners to anticipate a need they will have in the near future or to select among the available techniques those that are better matching their needs. Also, the reference architecture and the feature model may be employed as a guide to the development of novel SAFT approaches. When used in research, to the proposal of new approaches, the reference architecture leads the researcher to understand what are the main components and their sub-components in SATF approaches, and how they connect. On the other hand, the feature model lists the different possible characteristics a SATF approach may present, how they are connected and the existing restrictions associating them. In this way, the researcher can make wise decisions when the development is still in the initial phase. As future work, we plan to further explain how engineers and researchers can use the reference architecture and the feature model to build their own SAFT approach. We plan to build our SAFT approach and we will leverage this opportunity to dig also into this aspect.

ACKNOWLEDGMENTS

The authors express their gratitude to the experts who offered their time to answer the questionnaires and provide valuable feedback during interview sessions. The authors acknowledge the support of the PNRR MUR project VITALITY (ECS00000041), Spoke 2 ASTRA - “Advanced Space Technologies and Research Alliance", of the PNRR MUR project CHANGES (PE0000020), Spoke 5 “Science and Technologies for Sustainable Diagnostics of Cultural Heritage”, and of the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI.

REFERENCES

[1] ISO/IEC/IEEE 42010:2011. 2011. Systems and software engineering — Architecture description. , 37 pages.

[2] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing* 1, 1 (2004), 11–33.

[3] Morena Barboni, Antonia Bertolino, and Guglielmo De Angelis. 2021. What We Talk About When We Talk About Software Test Flakiness. In *Quality of Information and Communications Technology*, Ana C. R. Paiva, Ana Rosa Cavalli, Paula Ventura Martins, and Ricardo Pérez-Castillo (Eds.). Springer International Publishing, Cham, 29–39.

[4] Luciano Baresi and Carlo Ghezzi. 2010. The disappearing boundary between development-time and run-time. In *Proceedings of the Workshop on Future of Software Engineering Research (FoSER)*.

[5] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[6] Antonia Bertolino. 2007. Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering (FOSE'07)*. IEEE, 85–103.

[7] Antonia Bertolino, Guglielmo De Angelis, Sampo Kellomaki, and Andrea Polini. 2012. Enhancing Service Federation Trustworthiness through Online Testing. *Computer* 45, 1 (2012), 66–72. https://doi.org/10.1109/MC.2011.227

[8] Antonia Bertolino, Pietro Braione, Guglielmo De Angelis, Luca Gazzola, Fitsum Kifetew, Leonardo Mariani, Matteo Orrù, Mauro Pezzè, Roberto Pietrantuono, Stefano Russo, et al. 2021. A Survey of Field-based Testing Techniques. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–39.

[9] Antonia Bertolino, Guglielmo De Angelis, and Andrea Polini. 2012. Governance policies for verification and validation of service choreographies. In *International Conference on Web Information Systems and Technologies*. Springer, 86–102.

[10] Antonia Bertolino and Paola Inverardi. 2019. *Changing Software in a Changing World: How to Test in Presence of Variability, Adaptation and Evolution?* Springer International Publishing, Cham, 56–66. https://doi.org/10.1007/978-3-030-30985-5_5

[11] Alessio Bucaioni, Amleto Di Salle, Ludovico Iovino, Ivano Malavolta, and Patrizio Pelliccione. 2022. Reference architectures modelling and compliance checking. *Software and Systems Modeling* (2022). https://doi.org/10.1007/s10270-022-01022-z

[12] Kai-Yuan Cai. 2002. Optimal software testing and adaptive software testing in the context of software cybernetics. *Information and Software Technology* 44, 14 (2002), 841–855. https://doi.org/10.1016/S0950-5849(02)00108-8

[13] Kai-Yuan Cai, João W Cangussu, Raymond A DeCarlo, and Aditya P Mathur. 2003. An overview of software cybernetics. In *Eleventh Annual International Workshop on Software Technology and Engineering Practice*. IEEE, 77–86.
[14] Matteo Camilli, Antonio Guerriero, Andrea Janes, Barbara Russo, and Stefano Russo. 2022. Microservices integrated performance and reliability testing. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test*. 29–39.
[15] Mariano Ceccato, Davide Corradini, Luca Gazzola, Fitsum Meshesha Kifetew, Leonardo Mariani, Matteo Orrù, and Paolo Tonella. 2020. A Framework for In-Vivo Testing of Mobile Applications. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 286–296.
[16] Betty H Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al. 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*. 1–26.
[17] Allan Collins, Diana Joseph, and Katerine Bielaczyc. 2004. Design Research: Theoretical and Methodological Issues. *Journal of the Learning Sciences* 13, 1 (2004), 15–42. https://doi.org/10.1207/s15327809jls1301_2
[18] Heather L. Colquhoun, Danielle Levac, Kelly K. O'Brien, Sharon Straus, Andrea C. Tricco, Laure Perrier, Monika Kastner, and David Moher. 2014. Scoping reviews: time for clarity in definition, methods, and reporting. *Journal of Clinical Epidemiology* 67, 12 (2014), 1291–1294. https://doi.org/10.1016/j.jclinepi.2014.03.013
[19] Mark B Cooray, James H Hamlyn-Harris, and Robert G Merkel. 2014. Dynamic test reconfiguration for composite web services. *IEEE Transactions on Services Computing* 8, 4 (2014), 576–585.
[20] Ismayle de Sousa Santos, Magno Luã de Jesus Souza, Michelle Larissa Luciano Carvalho, Thalisson Alves Oliveira, Eduardo Santana de Almeida, and Rossana Maria de Castro Andrade. 2017. Dynamically Adaptable Software Is All about Modeling Contextual Variability and Avoiding Failures. *IEEE Software* 34, 6 (2017), 72–77. https://doi.org/10.1109/MS.2017.4121205
[21] Erick Barros dos Santos, Rossana MC Andrade, and Ismayle de Sousa Santos. 2021. Runtime testing of context-aware variability in adaptive systems. *Information and Software Technology* 131 (2021), 106482.
[22] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176–189.
[23] Erik M Fredericks and Betty HC Cheng. 2015. Automated generation of adaptive test plans for self-adaptive systems. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 157–167.
[24] Erik M Fredericks, Byron DeVries, and Betty HC Cheng. 2014. Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 17–26.
[25] Erik M Fredericks, Andres J Ramirez, and Betty HC Cheng. 2013. Towards run-time testing of dynamic adaptive systems. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 169–174.
[26] Luca Gazzola, Leonardo Mariani, Matteo Orrú, Mauro Pezze, and Martin Tappler. 2022. Testing Software in Production Environments with Data from the Field. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 58–69.
[27] Luca Gazzola, Leonardo Mariani, Fabrizio Pastore, and Mauro Pezze. 2017. An exploratory study of field failures. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 67–77.
[28] Alberto González, Eric Piel, and Hans-Gerhard Gross. 2009. A model for the measurement of the runtime testability of component-based systems. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 19–28.
[29] Antonio Guerriero, Raffaela Mirandola, Roberto Pietrantuono, and Stefano Russo. 2019. A hybrid framework for web services reliability and performance assessment. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 185–192.
[30] Elaheh Habibi and Seyed-Hassan Mirian-Hosseinabadi. 2021. On-demand Test as a Web Service Process (OTaaWS Process). In *2021 7th International Conference on Web Research (ICWR)*. IEEE, 16–23.
[31] Joachim Hänsel and Holger Giese. 2017. Towards collective online and offline testing for dynamic software product line systems. In *2017 IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design (VACE)*. IEEE, 9–12.
[32] Henner Heck, Stefan Rudolph, Christian Gruhl, Arno Wacker, Jörg Hähner, Bernhard Sick, and Sven Tomforde. 2016. Towards autonomous self-tests at runtime. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 98–99.
[33] Mark Kasunic. 2005. *Designing an effective survey*. Technical Report.
[34] Jeffrey O. Kephart and David M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50. https://doi.org/10.1109/MC.2003.1160055
[35] Barbara Kitchenham and Pearl Brereton. 2013. A systematic review of systematic review process research in software engineering. *Information and software technology* (2013).
[36] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2015. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing* 17 (2015), 184–206.

[37] Mariam Lahami and Moez Krichen. 2021. A survey on runtime testing of dynamically adaptable and distributed systems. *Software Quality Journal* (2021), 1–39.
[38] Mariam Lahami, Moez Krichen, and Mohamed Jmaiel. 2016. Safe and efficient runtime testing framework applied in dynamic and distributed systems. *Science of Computer Programming* 122 (2016), 1–28.
[39] Lucas Leal, Andrea Ceccarelli, and Eliane Martins. 2019. The SAMBA approach for Self-Adaptive Model-BAsed online testing of services orchestrations. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 495–500.
[40] Nianyu Li, Javier Cámara, David Garlan, Bradley Schmerl, and Zhi Jin. 2021. Hey! Preparing Humans to do Tasks in Self-adaptive Systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 48–58. https://doi.org/10.1109/SEAMS51251.2021.00017
[41] Majid Makki, Dimitri Van Landuyt, and Wouter Joosen. 2016. Automated workflow regression testing for multi-tenant saas: integrated support in self-service configuration dashboard. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation*. 70–73.
[42] Andreas Metzger, Eric Schmieders, Osama Sammodi, and Klaus Pohl. 2012. Verification and testing at run-time for online quality prediction. In *2012 First International Workshop on European Software Services and Systems Research-Results and Challenges (S-Cube)*. IEEE, 49–50.
[43] Zachary Munn, Micah DJ Peters, Cindy Stern, Catalin Tufanaru, Alexa McArthur, and Edoardo Aromataris. 2018. Systematic review or scoping review? Guidance for authors when choosing between a systematic or scoping review approach. *BMC medical research methodology* 18, 1 (2018), 1–7.
[44] Christian Murphy, Gail Kaiser, Ian Vo, and Matt Chu. 2009. Quality assurance of software applications using the in vivo testing approach. In *2009 International Conference on Software Testing Verification and Validation*. IEEE, 111–120.
[45] Patrizio Pelliccione, Eric Knauss, S. Magnus Ågren, Rogardt Heldal, Carl Bergenhem, Alexey Vinel, and Oliver Brunnegård. 2020. Beyond connected cars: A systems of systems perspective. *Science of Computer Programming* 191 (2020), 102414. https://doi.org/10.1016/j.scico.2020.102414
[46] Micah Peters, Christina Godfrey, Hanan Khalil, Patricia McInerney, Deborah Parker, and Cassia Baldini Soares. 2015. Guidance for conducting systematic scoping reviews. *International Journal of Evidence-Based Healthcare* 13, 3 (2015), 141 – 146. https://doi.org/10.1097/XEB.0000000000000050
[47] Roberto Pietrantuono, Stefano Russo, and Antonio Guerriero. 2020. Testing microservice architectures for operational reliability. *Software Testing, Verification and Reliability* 30, 2 (2020), e1725.
[48] Liliana Marie Prikler and Franz Wotawa. 2022. Challenges of Testing Self-Adaptive Systems. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B* (Graz, Austria) *(SPLC '22)*. Association for Computing Machinery, New York, NY, USA, 224–228. https://doi.org/10.1145/3503229.3547048
[49] Y Mohana Roopa and M Ramesh Babu. 2017. Self-test framework for self-adaptive software architecture. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, Vol. 2. IEEE, 669–674.
[50] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)* 4, 2 (2009), 1–42.
[51] Gerald Schermann, Dominik Schöni, Philipp Leitner, and Harald C Gall. 2016. Bifrost: Supporting continuous deployment with automated enactment of multi-phase live testing strategies. In *Proceedings of the 17th International Middleware Conference*. 1–14.
[52] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2007. *Guide to advanced empirical software engineering*.
[53] Samira Silva, Antonia Bertolino, and Patrizio Pelliccione. 2022. Self-adaptive testing in the field: are we there yet?. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 58–69.
[54] Bento Rafael Siqueira, Fabiano Cutigi Ferrari, Marcel Akira Serikawa, Ricardo Menotti, and Valter Vieira de Camargo. 2016. Characterisation of challenges for testing of adaptive systems. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. 1–10.
[55] Bento R Siqueira, Fabiano C Ferrari, Kathiani E Souza, Daniel SM Santibáñez, and Valter V Camargo. 2020. Fault Types of Adaptive and Context-Aware Systems and Their Relationship with Fault-based Testing Approaches. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 284–293.
[56] I STANDARD. 1990. STANDARD 610-1990,". *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries* (1990).
[57] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.
[58] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering*. Springer.