GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

Doctoral Thesis

Self-Adaptive Testing in the Field

PhD Program in Computer Science: 36th cycle

Supervisor:

Prof. Patrizio Pelliccione
patrizio.pelliccione@gssi.it

Author:

Samira Santos da Silva
samira.silva@gssi.it

Co-supervisor:

Prof. Antonia Bertolino
antonia.bertolino@isti.cnr.it

February 2025

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

"Só uma coisa torna um sonho impossível: o medo de fracassar."

Paulo Coelho (O Alquimista)

Abstract

We are increasingly surrounded by systems connecting us with the digital world and facilitating our lives 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. This growing need to test systems post-release has led to extending testing activities into production environments, where uncertainty and dynamic conditions pose significant challenges. Field testing approaches, especially Self-Adaptive Testing in the Field (SATF), face hurdles like managing unpredictability, minimizing system overhead, and reducing human intervention, among others.

Body Sensor Networks (BSNs) offer a cost-effective way to monitor patients' health and detect potential risks. Despite the growing interest attracted by BSNs, there is a lack of testing approaches for them. Testing a Body Sensor Network (BSN) is challenging due to its evolving nature, the complexity of sensor scenarios and their fusion, the potential necessity of third-party testing for certification, and the need to prioritize critical failures given limited resources.

Despite its importance, SATF remains underexplored in the literature. In this thesis, we observe that even 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.

Additionally, we address the unexplored gap in the literature regarding the testing of BSNs. To gain insights into how to test BSNs, we propose three BSN testing approaches: PASTA, ValComb, and TransCov. These approaches share common characteristics, which are described through a general framework called GATE4BSN. PASTA simulates patients with sensors and models sensor trends using a Discrete Time Markov Chain (DTMC). ValComb explores various health conditions by considering all sensor risk level combinations, while TransCov ensures full coverage of DTMC transitions. We empirically evaluate these approaches, comparing them with a baseline approach in terms

of failure detection. The results demonstrate that PASTA, ValComb, and TransCov uncover previously undetected failures in an open-source BSN and outperform the baseline approach. Statistical analysis reveals that PASTA is the most effective, while ValComb is 76 times faster than PASTA and nearly as effective.

Finally, we introduce AdapTA (Adaptive Testing Approach), a novel SATF strategy tailored for testing BSNs. AdapTA employs an ex-vivo approach, using real-world data collected from the field to simulate patient behavior in in-house experiments. Field data are used to derive Discrete-Time Markov Chain (DTMC) models, which simulate patient profiles and generate test input data for the BSN. The BSN's outputs are compared against a proposed oracle to evaluate test outcomes. AdapTA's adaptive logic continuously monitors the system under test and the simulated patient, triggering adaptations as needed. Results demonstrate that AdapTA achieves greater effectiveness compared to a non-adaptive version of the proposed approach across three adaptation scenarios, emphasizing the value of its adaptive logic.

In memory of my father Valter Henrique da Silva

Acknowledgements

This thesis is the culmination of years of dedication, perseverance, and the invaluable support of many individuals. Had it not been for the guidance of my mentors, the encouragement of my friends and family, and the essential support from GSSI, this incredible journey would not have been possible.

I would like to express my deepest gratitude to my supervisors, professors Antonia Bertolino and Patrizio Pelliccione, for their guidance, support, and invaluable insights throughout this journey. I especially thank you for your constant encouragement and support over the last four years, for your patience, and for never giving up on me. I consider myself extremely fortunate to have had the privilege of meeting you, as you have since become an incredible source of inspiration to me. I also deeply thank Prof. Franz Wotawa, from Graz University of Technology, and Prof. Sasikumar Punnekkat, from Mälardalen University. I sincerely appreciate the time and effort you dedicated to reviewing my thesis. Your valuable insights and suggestions greatly contributed to improving this work.

I'm grateful to everyone I've met in the last four years who has contributed to L'Aquila feeling like home. To the friends I met through GSSI, Adiel Tuyishime, Emerson Sales, Luciana Rebelo, Mario Prado, Renan Greca, and the professors, Prof. Catia Trubiani, Prof. Ludovico Iovino, Prof. Martina De Sanctis, I am truly thankful for the encouraging and unforgettable environment you fostered. I also want to thank my Brazilian friends in L'Aquila, Allyne Paz, Geisa Serpa, Oiram Santos, Paulo Bruno, Ricardo Caldas, and Rickson Pereira, for bringing a little bit of Brazil into my life here in Italy. I am also grateful to my teachers at CPIA, Antonella di Rocco, Chiara Marrelli, Fabio Iuliano, Stefania Maurizio, for helping me to discover my passion for the Italian language.

I would like to express my deep gratitude to those who, even from afar, never stopped offering me support and love whenever I needed it throughout this journey, my family in Brazil. I am particularly thankful to my mother, Eni Silva, and my brother, Hudson Silva. I also extend my thanks to my son, Saymon Meireles, who, despite his young age, had to understand the many times I had to be away to focus on my PhD. Finally, I am profoundly grateful to my beloved husband, Sincler Meireles, for his unwavering support from the beginning to the end of my PhD. Even before it became my dream, it was his, and now it is a reality. Everything I have achieved over these years would not have been possible without his constant encouragement.

Finally, I am deeply grateful to GSSI and its entire staff for their constant support, both financial and academic, throughout these four years. I am incredibly proud to hold a PhD from such an outstanding institution.

Contents

List of Figures

List of Tables

Abbreviations

ACTS	**A**dvanced **C**ombinatorial **T**esting **S**ystem
AdapTA	**Adap**tive **T**esting **A**pproach
BMI	**B**ody **M**ass **I**ndex
BSN	**B**ody **S**ensor **N**etwork
CBS	**C**omponent-**B**ased **S**ystem
CFM	**C**ontext-aware **F**eature **M**odel
CI&CD	**C**ontinuous **I**ntegration **and** **C**ontinuous **D**eployment
CMC	**C**ontrolled **M**arkov **C**hains
CPS	**C**yber-**P**hysical **S**ystem
CT	**C**ombinatorial **T**esting
DSR	**D**TMC **S**ensor **R**eading
DTMCs	**D**iscrete-**T**ime **M**arkov **C**hains
FSR	**F**ield **S**ensor **R**eading
GATE4BSN	**G**eneric **A**pproach to **TE**sting **for BSN** applications
HMMs	**H**idden **M**arkov **M**odels
ICU	**I**ntensive **C**are **U**nit
MCbt	**M**odel-**C**ombinatorial **b**ased **t**esting
MM	**M**arkovian **M**odels
PASTA	**PA**tient **S**imulation for **T**esting of bsn **A**pplications
ROS	**R**obot **O**perating **S**ystem
SA-BSN	**S**elf-**A**daptive **B**ody **S**ensor **N**etwork
SaaS	**S**oftware **a**s **a** **S**ervice
SASs	**S**elf-**A**daptive **S**ystems
SATF	**S**elf-**A**daptive **T**esting in the **F**ield
SOA	**S**ervice-**O**riented **A**rchitecture

SUT	**S**ystem **U**nder **T**est
SVS	**S**mart **V**acuum **S**ystem
TransCov	Sensor **Trans**itions **Cov**erage
ValComb	Sensor **Val**ues **Comb**ination

Chapter 1

Introduction

Traditionally, the software testing task has always been thought of as a fault-finding process performed during the development cycle [3]. Since the activities in this process are commonly performed in the development environment, the work in [1] refers to such activities as *in-house* or *in-vitro* software testing. However, in recent years, academics, 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]. 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 [8] due to the tremendous complexity, evolvability, and interconnection of current software-intensive systems.

The work in [1] defines this tendency of moving testing activities from the development to the production environment with different nuances. *Testing in the field* [1] 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 separate 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 [1] 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 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*" [1], and that, concerning the oracles for field testing, these "*need to adapt to the unknown execution conditions that can emerge in the field*" [1].

The definition of adaptive testing is also investigated in the recent area of "software cybernetics" [9], 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 [10]. 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 [1], 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 recent years [11–13]. 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 [11]. 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 to test Self-Adaptive Systems (SAS) are inherently designed to be used at runtime. Nonetheless, we do not believe, and this is confirmed by some of the works we collected by reviewing the literature in Chapter 3 (e.g., [14–17]), that the SATF scope is restricted to SAS. From a more pragmatic perspective, based on the primary studies we collected, 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., [18]), (ii) systems that could be updated over time, due to the origination of new needs, among other reasons (e.g., [17]), and (iii) systems for which the configuration may be changed by the user at runtime (e.g., [14]). Of course, we could even have systems that present more than one of the above-mentioned dynamic characteristics.

Examples of systems that evolve over time are the Body Sensor Networks (BSNs). A BSN consists of a network of wearable or implantable sensor nodes carefully positioned on or inside the human body to track physiological parameters or surrounding environmental factors [19]. Such systems are frequently employed in healthcare to monitor critical health metrics, including heart rate, body temperature, and glucose levels. Furthermore, BSNs find applications in areas such as the healthcare industry [20–22], the military-industrial sector [23], sports and entertainment [24–26], and the social public field [27, 28]. BSNs are good exemplars of evolving systems since, for example, their sensors may be deactivated/activated over time. Their environment, which includes the monitored patient, may also change, for example by replacing a patient with a normal Body Mass Index (BMI) that was previously connected to the sensors with an Obesity-3 one. Finally, the health conditions of a patient also change over time, requesting different levels of attention.

Testing Body Sensor Networks (BSNs) presents several challenges due to their dynamic, sensitive, and highly interconnected nature. The environment where the sensors operate (the human body) is constantly changing, making it difficult to predict all possible conditions. Wearable devices and body sensors operate with limited batteries and may turn off when the battery runs out. Finally, trusting in the sensor accuracy and the often integration of BSNs with different operating systems, smartphones, and hospital systems, are also important challenges. In-house testing approaches may not be able to fully replicate the diverse conditions encountered during the real-world operation of BSNs, highlighting the need for field-based testing. Additionally, self-adaptive testing strategies may

enhance resilience by dynamically adjusting to the uncertainties and variations present in the field.

In the current literature, though, there is a shortage of proper Software Engineering (SE) approaches focused on BSNs. Besides, notwithstanding the extensive body of research on testing in SE, relatively little of this knowledge has been applied to BSNs, as we better explain in Chapter 6.

Concerning BSNs testing, since the application consists of elaborating the vital signals as inputs that come from different sensors, testers need to consider different combinations of values that could be read. This scenario seems to be the typical case for using Combinatorial Testing [29]. However, combinatorial testing provides a static approach to test case generation, whereas BSNs work by monitoring and analyzing some vital measures that are captured over time. In particular, the sensors could at any moment detect some alarming values and react accordingly, e.g., by issuing an alarm system. Hence, we would like to continue testing dynamically a BSN over differing combinations of parameter values. Testing a BSN also involves other challenges, e.g.: (i) ensuring the proper testing of data fusion, (ii) since third-party testing could be required for testing at the system level, e.g., to certify the compliance to medical ISO standards, testing should be feasible also without code accessibility, and (iii) given potential limited physical and human resources, testing should also prioritize the most critical failures.

This thesis tackles the identified gap in Self-Adaptive Testing in the Field (SATF) by conducting a comprehensive literature review, defining the concept, proposing a taxonomy, and presenting a reference architecture for SATF. Additionally, we also identify and analyze the key challenges associated with existing SATF approaches, as extracted from the literature.

To gain insights into how BSN testing should be conducted, we also propose the in-house approach *GATE4BSN*, a novel approach to testing BSN applications. *GATE4BSN* is a generic and abstract approach that is instantiated in three approaches, namely: (i) *PASTA*, which simulates patients by considering a set of sensors and by mimicking the trend of each sensor via a Discrete-Time Markov Chain (DTMC); (ii) *ValComb*, which makes use of all possible combinations of sensor risk levels to explore different behaviors of a patient; and (iii) *TransCov*, which employs a dummy patient in which all the transitions

in the DTMCs are labeled with equal probability. Then, their DTMCs are executed until all the transitions are covered.

Finally, we also introduce a novel SATF approach, *AdapTA* (Adaptive Testing Approach), specifically designed for testing BSNs. *AdapTA* is an ex-vivo testing approach, in which actual patient data, which we here call Field Sensor Reading (*FSR*), is *collected* from the field and *preprocessed* to be transformed into DTMCs that represent the condition of each sensor over time. Next, these DTMCs are used to *simulate* patient profiles. For each patient profile, we run its DTMCs and input the obtained data, the DTMC Sensor Reading (*DSR*), into the BSN, while a *Monitor* component is responsible for monitoring the environment (e.g., the patient profile and condition) and the SUT (e.g., activation or deactivation of sensors). If a change occurs in either of them that triggers adaptation, the *Analyse* component decides which adaptation should be made based on the changed object. Then, a *Plan* for the adaptation of Test Cases, Oracle, or Test Strategy is made and *Executed*. The *Test Strategy* component decides when it is time to launch a test session, during which the *BSN Outcomes* together with *DSR*s are *logged*, or otherwise the *Patient Simulation* is continued. When testing, based on each *DSR*, the *Expected Outcome* is *computed* and *compared* to the actual *BSN Outcome*, to decide whether the Test Cases (pairs of *DSR* and *expected outcome*) pass or fail.

For the empirical evaluation, we use both approaches, *GATE4BSN* and *AdapTA*, to test a self-adaptive system from the literature, the SA-BSN (Self-Adaptive Body Sensor Network) [30], which already has been used in the literature [31], but not for testing.

In summary, this thesis provides the following original contributions:

- We conduct a review of the literature and propose a definition, a taxonomy, and a reference architecture for SATF [32, 33];
- We introduce *GATE4BSN* as a generic approach composed of the main abstract steps needed to test BSNs [34, 35];
- We provide formal definitions for *GATE4BSN* and instantiate it in three different approaches: *PASTA*, *ValComb* and *TransCov* as potential alternative solutions [34, 35];

- For *GATE4BSN*, we investigate the failures to understand whether they might be due to the same faults, by clustering the non-passing test cases from *PASTA* and *ValComb*;
- For *GATE4BSN*, we conduct extensive experimentation (including the three approaches, and replicating each experiment 25 times);
- We introduce *AdapTA*, a novel self-adaptive ex-vivo approach to test BSNs [36];
- We describe three adaptation scenarios by providing the trigger activation and the adaptation policy for each of them [36];
- For *AdapTA*, we include extensive experimentation (covering the three scenarios, the baseline for each of them, and replicating each experiment 5 times);
- We conduct a rigorous statistical comparison of the results (using either the Mann-Whitney U Test or the Kruskal-Wallis test, and the Vargha-Delaney A measure);
- Last, but not least, we test a new fixed version of the SA-BSN.

1.1 Document Structure

This thesis is organized as follows. Chapter 2 overviews background concepts of the work described in this thesis. Specifically, we start by describing the model we use to simulate a patient in *GATE4BSN* and *AdapTA*, the Discrete Time Markov Chains (DTMCs). Then, we define the concept of Combinatorial Testing, which is a characteristic of *ValComb*. We also distinguish the different types of field-testing. The MAPE-K loop, a well-known model to represent self-adaptive systems, is also outlined. Then, we also present the definition of BSN, illustrate the concept of testing BSNs, and present our System Under Test, the Self-Adaptive BSN (SA-BSN). Chapter 3 outlines our review of the literature on SATF. We present our research methodology, the complete protocol employed, the validation with experts, and the results of reviewing the literature, that is, a definition, a taxonomy, and a reference architecture for SATF. We also provided a list of challenges when proposing a SATF approach. In addition, we show how we validate our proposal with the experts, and present the limitations and threats to the validity of this review of the literature. In Chapter 4 we describe our attempt to gain insights into how to test BSNs by proposing the *GATE4BSN*. Specifically, we present a simple example of BSN to

illustrate next how the *GATE4BSN* should be employed. Then, we also present the three instances of *GATE4BSN*, namely *PASTA*, *ValComb*, and *TransCov*. We also report the experiments we perform and possible threats to their validity. Chapter 5 consists of the description of *AdapTA* and its scenarios. We also report the experiments we conducted and the threats to their validity. Chapter 6 provides a comparison not only of our literature review (described in Chapter 3) with existing secondary studies on SATF, but also of *GATE4BSN* and *AdapTA* with works in the literature. In Chapter 7 we discuss some of the results obtained during the development of the work described in this thesis. Finally, Chapter 8 presents our conclusions, future works, and the publications obtained during the development of this thesis. In Appendix A we report the e-mail sent to experts to validate the taxonomy and reference architecture (as explained in Section 3.1.2), the questionnaire mentioned in the e-mail, and a link for the replication packages of our contributions in this thesis.

Chapter 2

Background

This chapter introduces the key concepts needed to understand the work in this thesis. It covers Discrete-Time Markov Chains and Combinatorial Testing, as well as Testing in the Field and the MAPE-K Loop. It also looks into Body Sensor Networks (BSNs), illustrates how they are tested, and describes the SUT employed in this work, the Self-Adaptive Body Sensor Networks (SA-BSNs). These topics help set the stage for the studies and approaches discussed later.

2.1 Discrete-Time Markov Chains

A Markov process refers to a stochastic process for which the future probabilistic behavior depends only on the present state and is not affected by how this state has been reached [37]. When the state space is finite, the Markov process is said a Markov Chain, and if evolving in discrete steps (or, we observe the process at a discrete set of time points), it becomes a Discrete-Time Markov Chain (DTMC) ([37], Ch.7). In this paper, since a BSN samples sensor readings at discrete time intervals, we use DTMCs to simulate the patient health status and depict for each sensor a DTMC as a transition system in which transitions are annotated with probabilities.

A DTMC is a tuple (S, P) where S is a set of states, and $P : S \times S \rightarrow [0, 1]$ is a function that given $s, s\prime \in S$ as input, calculates the probability of going from s to s', p, with $0 \leq p \leq 1$.

Moreover, for each $s \in S$

$$\sum_{s\prime \in S} P(s, s\prime) = 1 \tag{2.1}$$

A finite DTMC, containing finite s ∈ S, is typically represented by a probability transition matrix (TM),

$$TM = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0m} \\ p_{10} & p_{11} & \cdots & p_{1m} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m0} & p_{m1} & \cdots & p_{mm} \end{bmatrix}$$

where $p_{ij} = P(s_i, s_j)$. Each row $p_{i0}, \ldots, p_{im}$ in TM corresponds to the probabilities labeling all arcs that exit from state i, and according to Equation 2.1 above, their sum is equal to 1.

The work in [38] provides a method for statistical testing based on a Markov chain model of software usage. In their work, Markov chains have two important applications. First, compared to many other methods currently in use, it is more general since it enables test input sequences to be created from numerous probability distributions. Second, an additional Markov chain is created to capture the test history, including any observed failure information. The test input sequences that are formed from the chain and applied to the program constitute a stochastic model in and of themselves. Analytical calculations on this chain are used to evaluate the impact of the failures.

2.2 Combinatorial Testing

As a BSN receives as input a set of values provided by independent sensors, a proper testing strategy should examine systematically the combinations of sensor values; this is the basic concept behind Combinatorial Testing (CT).

CT is a well-known testing strategy providing a good balance between cost and effectiveness [29]. To cover specific parameter value combinations, it samples the vast combination space using a condensed test suite. T-way CT is a strategy in this sense, relying on the empirically proven intuition that most failures are caused by a single parameter value or

by interactions between a small number of parameters [39], because not every parameter affects every failure. Indeed, empirical studies showed that faults are generally caused by the interaction among at most six parameters [40].

A t-way test set satisfies the property that for any subset of t parameters, all possible combinations of their values are covered by at least one test in the test set. In particular, when t=2 (which is commonly named *pairwise testing*), for any two input parameters, the test set includes at least one test case that covers each combination of their relevant values.

Various tools exist that automate test generation for CT. In our approach, we adopted the Advanced Combinatorial Testing System (ACTS) [41], a well-known test generation tool for constructing t-way combinatorial test sets. It currently supports test set generation for values of t ranging from 1 to 6. According to the authors, empirical studies indicate that a value of t up to 6 is adequate for most practical applications. Basically, when generating a new set of test cases, it is required the definition of the possible parameters, their value ranges, and any constraints associated with these parameters. The tool offers both command line and GUI interfaces for user interaction. Figure 2.1 depicts a cropped screenshot of the ACTS tool.

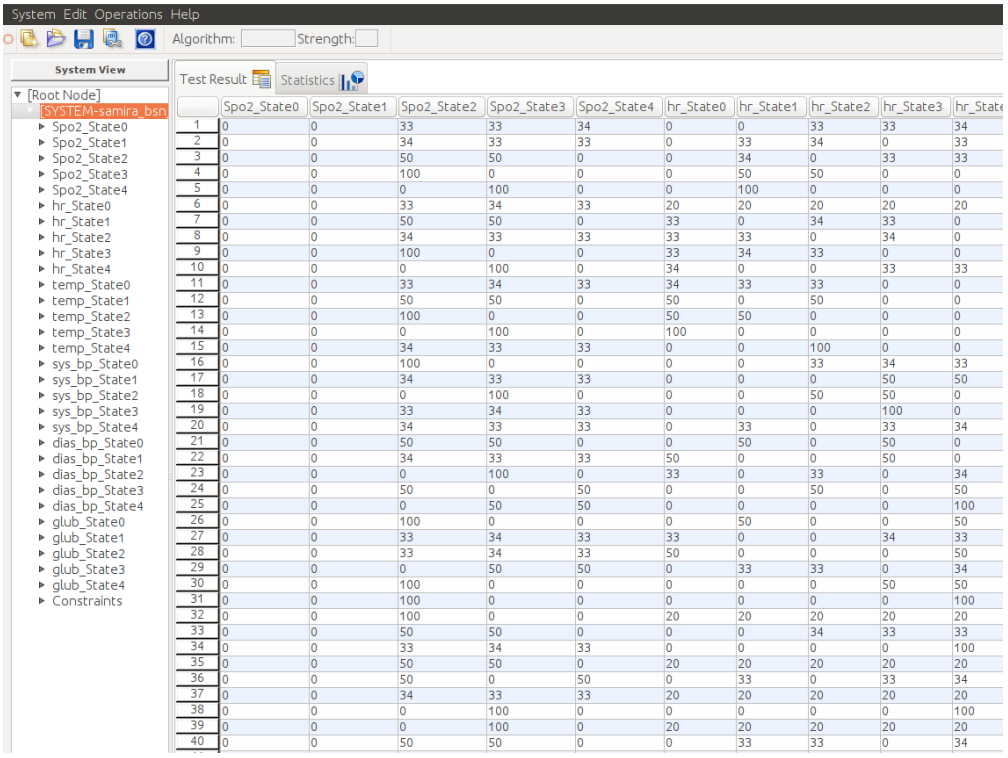


FIGURE 2.1: Illustration of the ACTS tool in action.

2.3 Testing in the Field

Regardless of whether testing activities are intended to uncover development bugs or investigate study field failures, they are typically conducted in the development environment. These activities are commonly referred to as *in-house* or *in-vitro* software testing [1]. The inability to address all faults using traditional in-house testing methods has sparked interest in testing software systems in the field, bridging the gap between in-house validation and real-world execution. As depicted in Figure 2.2, extracted from [1], this shift of testing activities from the laboratory to the production environment can take on various forms. Test cases can be executed: (1) directly in the field on the same instance of the software used in production, which is referred to as *online testing* (the rightmost flow in Figure 2.2); (2) on a separate instance still running in production, known as *offline testing* (middle flow); or (3) *in-house*, but using data collected from the field, termed *ex-vivo testing* (leftmost flow).

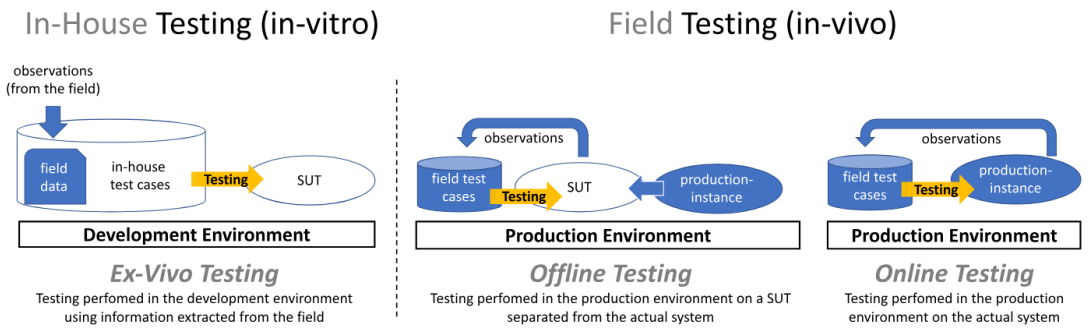


FIGURE 2.2: Classes of field testing approaches according to [1].

Traditional in-house (in-vitro) testing is a well-established field with studies spanning several decades, with the first specialized workshop held in the mid-1970s [1]. In contrast, research in field testing has emerged more recently and has not yet been thoroughly explored as a distinct discipline [1].

2.4 MAPE-K Loop

Self-Adaptive Systems (SASs) have the capability to autonomously adjust their behavior during runtime in response to changes or unforeseen environmental conditions [11].

These systems must consistently maintain acceptable performance throughout their operation. As software is deployed in more widespread and critical applications, enabling self-adaptation is increasingly considered essential to prevent expensive disruptions in system repair, maintenance, and evolution [42]. SASs are commonly employed in industries such as automotive, telecommunications, and environmental monitoring.

Typically, a SAS is composed of two key components [2]: the Managed Sub-System and the Manager (Autonomic) Sub-System. The Managed Sub-System is responsible for the core domain functionality, while the Manager Sub-System oversees the adaptations needed to meet specific quality objectives. This adaptation is achieved through the implementation of a feedback control loop. According to IBM's vision of autonomic computing [43], the autonomic behavior is realized by applying the MAPE-K reference model.

The MAPE-K Control Loop is the most widely used model for incorporating self-adaptation into software-intensive systems [44]. Figure 2.3 illustrates an Autonomic Element, where the Manager (Autonomic) Sub-System implements the adaptation logic using the well-established MAPE-K model, while the Managed System handles the system functionalities. It is composed of sensors and effectors, along with four components that facilitate the feedback loop: *Monitor*, *Analyze*, *Plan*, and *Execute* [43]. These components are supported by a Knowledge Base, which enables the exchange of information among them. The Sensors, located within the Managed Sub-System, monitor its state and gather data that reflects the system's condition, while the Effectors implement changes to the Managed Sub-System [2].

The *Monitor* component is responsible for collecting, aggregating, filtering, and reporting data such as metrics and topologies from a managed resource [43]. The *Analyze* component correlates and models complex scenarios (e.g., time-series forecasting and queuing models). These capabilities enable the autonomic manager to understand the IT environment and make predictions about future conditions [43]. The *Plan* component is responsible for formulating the necessary actions to achieve goals and objectives, guided by policy information [43]. Finally, the *Execute* component is responsible for managing the execution of a plan, taking into account the need for dynamic updates [43].

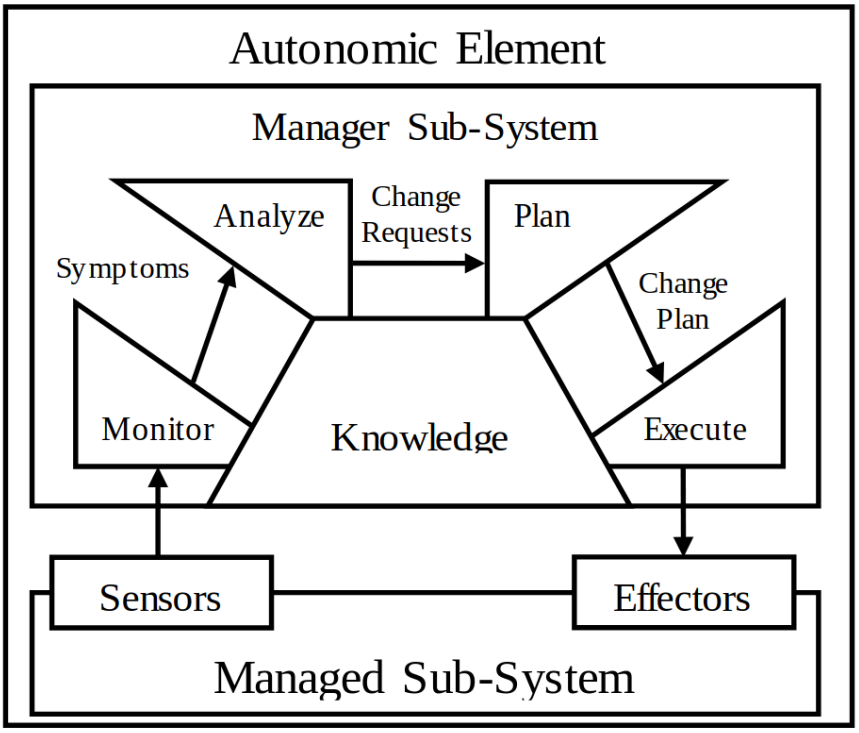


FIGURE 2.3: MAPE-K control loop from [2] .

2.5 Body Sensor Networks

Wireless Sensor Networks (WSNs) are spatially dispersed networks of sensors that track their surroundings and send the information they gather to a central processor. According to [45], current WSNs are deployed on land, underground, and underwater. Depending on the environment, a sensor network encounters various challenges and limitations. There are five classical types of WSNs [45], each with distinct characteristics: Terrestrial WSN, consisting of hundreds to thousands of inexpensive wireless sensor nodes deployed in a given area; Underground WSN, where sensor nodes monitor underground conditions; Underwater WSN, involving sensor nodes and vehicles deployed underwater; Multi-media WSN, which monitors and tracks events using media such as video, audio, and imaging; and Mobile WSN, a network of sensor nodes that move independently and interact with the physical environment.

Body Sensor Networks (BSNs) are a particular kind of WSN that resulted from a recent push on wearable biosensors technology development [46, 47]. BSNs attract much interest, mostly from the healthcare industry [20–22], but also from the military-industrial sector [23], sports and entertainment [24–26], and the social public field [27, 28]. BSNs

are a subcategory of WSNs but come with specific challenges due to their use on the human body. They require smaller sensors, lower power consumption, low latency, enhanced security, and greater reliability compared to traditional WSNs.

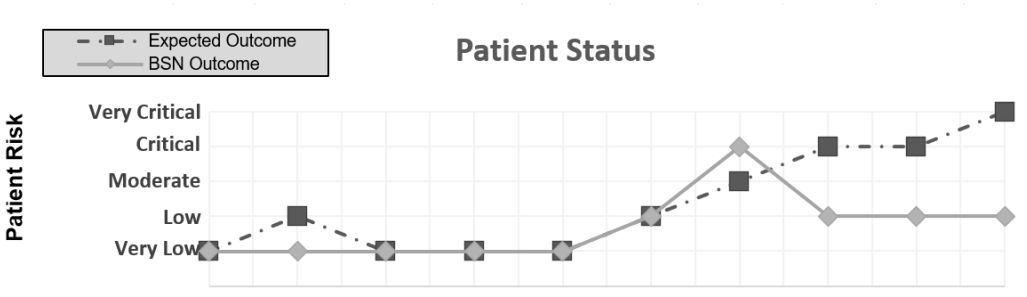
BSNs have also steadily emerged as a hotbed of research. Developing comprehensive BSN applications requires an interconnection between physical signals with cybernetic devices, middleware components, and data fusion algorithms. In the medical domain, for example, vital signals (such as heartbeat rate and blood pressure) are captured by a variety of physiological parameter sensors positioned in or around the human body, on the body surface [48], or even by mobile devices, including smartwatches or smartphones. The collected data are then processed in real-time and are finally distributed to e.g., caregivers, emergency personnel, or medical servers that may assist the patient.

2.6 Testing Body Sensor Networks

Testing BSNs is the activity of finding (test) cases that describe a set of vital signs (i.e., test inputs) provided by the sensors, which, once processed and fused, give a diagnostic of the patient's health status (i.e., test outcomes).

To exemplify an instance of the problem, Figure 2.4 illustrates the Patient Risk levels over time for a fictitious patient from 13:45 to 16:00. The patient's status is characterized by a combination of streamed vital signs that vary over time (a.k.a. vital signals). Each signal is captured by a sensor designated to measure the specific sign in terms of key-value pairs. Keys are timestamps (e.g., Time = 13:45) and values are real numbers indicating the intensity of the signal (e.g., Heartbeat rate = 90bpm). Thus, representing the behavior of a patient is a matter of describing a composite of streamed values of vital signs and how they evolve through time. The continuous line provides the actual diagnosis provided by the BSN, while the dashed line refers to the expected one.

Figure 2.4 shows a strong decrease in blood oxygenation ($\leq$95%) after 15:30. Accordingly, the expected outcome for the BSN would be to reach a very critical risk for the patient's health status. This oracle verdict is, e.g., the diagnosis given by the experts (doctors) based on the observed vital signs. However, the patient risk level diagnosed by the BSN under test (depicted as BSN Outcome) stays at low risk: such behavior evidences a failure.



Time	13:45	14:00	14:15	14:30	14:45	15:00	15:15	15:30	15:45	16:00
Heart beat rate [bpm]	90	85	92	88	87	100	99	114	120	120
Oxygenation [%]	99	98	99	98	98	98	97	95	93	88
Temperature [°C]	36.5	36.5	36.5	37	37	37.2	38.2	36.5	36.1	36.5

FIGURE 2.4: Representation of the vital signs of a patient.

When testing BSNs, there are two approaches regarding the variation of vital signs values over time. One strategy involves analyzing the progression of vital signs over a period to assess a patient's risk level - meaning that the current risk assessment is influenced by how the vital signs have evolved in the preceding seconds. Alternatively, a simpler approach determines the patient's risk level based solely on the vital signs measured at a specific instant of time, without considering prior variations. In this work, we consider the latter.

2.7 Self-Adaptive Body Sensor Network (SA-BSN)

The Self-Adaptive Body Sensor Network (SA-BSN) [30], which is publicly available[1], is designed to monitor the patient's health and detect emergencies in real time. It incorporates six sensors: an electrocardiograph (ECG) for heartbeat rate and electrocardiogram waveform, a pulse oximeter (Oxi) for measuring blood oxygen saturation, a thermometer (Term) to record body temperature in Celsius, a sphygmomanometer for diastolic (Abpd) and systolic (Abps) blood pressure readings, and a glucose sensor (Glc) to monitor blood glucose levels.

Table 2.1 outlines the defined risk level ranges for the sensors used by the SA-BSN. For the SA-BSN, three are the risk level ranges (i.e., Low, Medium-1, and High-1) presented by each of the Oxi, Abps, and Abpd sensors. The risk level ranges of the Ecg, Term, and Glc sensors are five (i.e., Low, Medium-0, Medium-1, High-0, and High-1). We use the suffixes -0 and -1 to distinguish the different risk ranges that are associated with the

[1] https://github.com/rdinizcal/sa-bsn

same risk level for some of the sensors (i.e., Ecg, Term, and Glc). For example, in the table, we can notice that the Ecg sensor has two risk ranges for "medium", Medium-1 corresponds to the range (97,115) and Medium-0 corresponds to the range (70,85), i.e., the suffix -1 is used with the ranges that are greater than the "Low" one, and the suffix -0 with the ranges with lower values. For the sensors with a single risk range for each risk level (e.g., Oxi, Abps, and Abpd), we use the suffix -1. This distinction, which is inherent to the way the SA-BSN was designed, is taken into account in the experimental procedure of the approaches we propose in this thesis.

TABLE 2.1: Data ranges for risk levels in SA-BSN sensors.

Sensor	**Data Ranges for Risk Levels**
Oxygen Saturation (Oxi)	100>low>65>medium>55>high>0
Heartbeat Rate (Ecg)	300>high>115>medium>97>low>85>medium>70>high>0
Temperature (Term)	50>high>41>medium>38>low>36> medium>32>high>0
Systolic Body Pressure (Abps)	300>high>140>medium>120>low>0
Diastolic Body Pressure (Abpd)	300>high>90>medium>80>low>0
Glucose (Glc)	200>high>120>medium>96>low>55>medium>40>high>20

SA-BSN is a self-adaptive system, thus, based on changes in the environment and internal state, it adapts itself by adjusting the central hub (i.e., a component to fuse vital signs and classify the overall health situation of the patient), adjusting sensor frequency, or deactivating/activating sensors.

In this work, we employed a version of the SA-BSN fixed by us and that is made available in our replication package for *GATE4BSN*[2] and *AdapTA*[3]. The original version contained three bugs related to the sensor readings printing function, the DTMC state transition function, and the lack of message sending to other ROS (Robot Operating System) nodes whenever a sensor is activated/deactivated. These issues, which affected the accuracy of the results, were promptly reported to the developers. For our experiments, we utilized the corrected version where these bugs have been resolved.

[2]https://github.com/samirasilva/Paper_JSS
[3]https://github.com/samirasilva/AdapTA_Paper_AST

Chapter 3

Literature Review and SATF Definitions

Given the limited research on SATF in the literature, this chapter seeks to address this gap by providing a comprehensive scoping review of its state of the art. Our contributions include defining SATF, developing a taxonomy, and proposing a reference architecture for SATF approaches. Additionally, we also identify key research challenges and gaps in SATF.

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 definition and 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.

Section 3.1 describes the research methodology followed in this study, the design research method [49]. Section 3.2 provides a definition for SATF based on the literature. Section 3.3 reports the results of the study in outlining a taxonomy for SATF. Section 3.4 presents the reference architecture for SATF approaches. Section 3.5 discusses the identified open challenges. Section 3.6 describes the activities we performed to validate the taxonomy and the reference architecture, and presents the outcome of the validation. Section 3.7 characterizes limitations and threats to the validity of this study. The study reported in this chapter is currently published in [32], but it is an extension of our previous work [33].

3.1 Research Methodology

In this work, we aim to define a taxonomy for SATF, contribute a reference architecture for SATF approaches, and identify and discuss the main challenges. To achieve these objectives we followed the design research method [49], which is mainly used for creating new knowledge and artifacts that are required to address a particular phenomenon or problem.

As shown in Figure 3.1, 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 paper [33] (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 [33], and are the ones that we explore in this chapter and in [32]. 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:

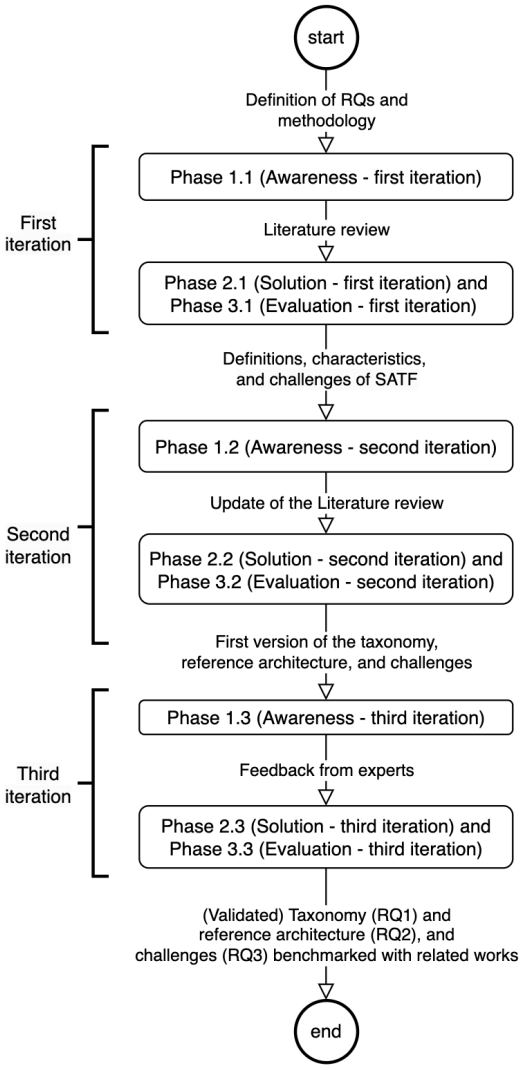


FIGURE 3.1: Overview of the research methodology mapped to the results.

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 analyzed 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 [33]. In this iteration, we also complemented the review with snowballing to achieve a higher level of completeness. The identified primary studies have been then analyzed 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 analyzed 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 fully reported in [33].

Phase 2.2 - Solution, Second Iteration: We contributed a first version of the taxonomy of SATF, reference architecture, and challenges. This first version is used to gather feedback from the experts.

Phase 2.3 - Solution, Third Iteration: We contributed a final and validated version of the taxonomy of SATF, reference architecture, and challenges. This final version is the one reported in this thesis and in [32].

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

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

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 3.1.1, we describe the scoping review made in step 1.1 and its subsequent update made in step 1.2. Then, in Section 3.1.2 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 on GitHub[2].

3.1.1 Scoping Review

In this work, we employed the scoping review research methodology [50–52] to address the research questions mentioned in the introduction. The scoping review research methodology is popular in the medical field [50–52], but it is also gaining popularity in the field of software engineering [53]. Scoping reviews, according to [50], are analogous to systematic literature reviews and they adhere as well to a rigidly 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 [50].

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 [50], 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 Section 3.1.1.7.

[2] https://tinyurl.com/2kdb5jef

3.1.1.1 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 [1], 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", incorporated 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 3.1. Three of the most popular databases, IEEExplore, ACM Digital Library, and Scopus, were employed in our scoping review.

Table 3.1: 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[3].

[3]The literature search was conducted in more subsequent dates, precisely the searches for the previous version [33] 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 find any primary study to be added.

3.1.1.2 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 were then eliminated. Next, after removing the duplicates and combining the results from screening for several databases into a single set, we obtained 121 publications.

3.1.1.3 Selection Criteria

The selection criteria are summarized in Table 3.2. The inclusion criteria are based on the research questions, while the exclusion ones are standard quality criteria (similar to those from [1]). 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. We also notice that even when limiting the year during the search, papers outside the specified range may still be retrieved. Unpublished papers may also be found, such as the preprint versions of works. Our selection criteria prevent this from occurring.

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

Table 3.2: Inclusion and exclusion selection criteria.

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

3.1.1.4 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.

3.1.1.5 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 [54]. First, we performed 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 were also performed. The 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 paper 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. Note that in all three iterations/phases of snowballing, both backward and forward snowballing are performed. Then, the total number of papers collected by the snowballing phases[4] 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**.

3.1.1.6 Data Extraction

The author and her supervisors read the entire papers and then individually extracted the information needed to respond to the three research questions (RQs) outlined in the introduction of this chapter. Thus, we had two independent readings and classifications

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

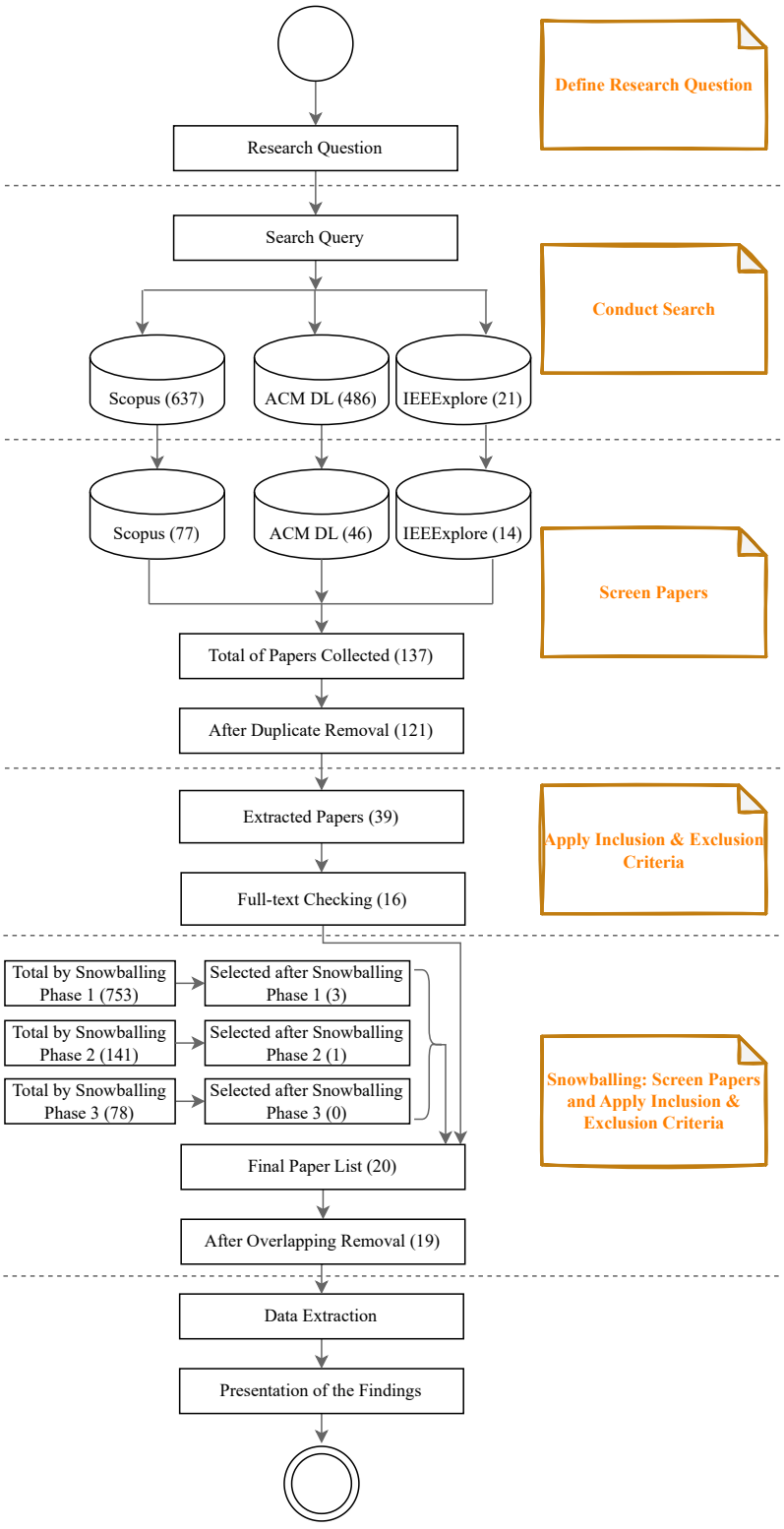


FIGURE 3.2: Summary of the selection protocol for this scoping review.

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

Table 3.3: SATF approaches gathered in this study.

Final List of Papers			
Ref.	Year	Venue	Title
[55]	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
[56]	2013	International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)	Towards run-time testing of dynamic adaptive systems
[15]	2014	IEEE Transactions on Services Computing (TSC)	Dynamic test reconfiguration for composite web services
[57]	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
[18]	2015	International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)	Automated generation of adaptive test plans for self-adaptive systems
[17]	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
[58]	2016	International Middleware Conference (Middleware)	Bifrost – Supporting continuous deployment with automated enactment of multi-phase live testing strategies
[59]	2016	Elsevier Science of Computer Programming (SCP)	Safe and efficient runtime testing framework applied in dynamic and distributed systems
[60]	2016	International Workshops on Foundations and Applications of Self* Systems (FAS* W)	Towards autonomous self-tests at runtime
[61]	2017	International Conference of Electronics, Communication and Aerospace Technology (ICECA)	Self-test framework for self-adaptive software architecture
[62]	2017	International Workshop on Variability and Complexity in Software Design (VACE)	Towards collective online and offline testing for dynamic software product line systems
[16]	2019	International Symposium on Software Reliability Engineering Workshops (ISSREW)	A hybrid framework for web services reliability and performance assessment
[63]	2019	Computer Software and Applications Conference (COMPSAC)	The SAMBA approach for self-adaptive model-based online testing of services orchestrations
[14]	2020	International Conference on Software Testing, Validation and Verification (ICST)	A framework for in-vivo testing of mobile applications
[64]	2020	Wiley Software Testing, Verification and Reliability (STVR)	Testing microservice architectures for operational reliability
[65]	2021	International Conference on Web Research (ICWR)	On-demand Test as a Web Service process (OTaaWS process)
[66]	2021	Elsevier Information and Software Technology (IST)	Runtime testing of context-aware variability in adaptive systems
[67]	2022	International Conference on Automation of Software Test (AST)	Microservices integrated performance and reliability testing
[68]	2022	International Conference on Software Testing, Verification and Validation (ICST)	Testing software in production environments with data from the field

of meetings that the author and her supervisors 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.

3.1.1.7 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 3.2. 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 3.3. Comparing this final list against the list of selected primary studies on testing in the field in [1], we notice that: *(i)* only four studies are common to the two surveys, namely [15, 18, 57, 59], while the remaining 13 papers of this review were not included in [1]; and *(ii)* 8 papers of this review appeared after the search conducted in the previous survey.

3.1.2 Validation with Experts

This section describes 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 were not strongly grounded in the collected data but that included our subjective interpretation and beliefs.

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 did research and 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 validating 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. We make available the e-mail sent to the experts and a copy of the questions contained in the questionnaire in Appendix A.

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 $E1$,...,$E19$), 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. Table 3.4 reports the experts' publishing experience in years.

TABLE 3.4: Years of research experience for the experts.

Publication Experience	Number of Experts
0-5 years	1 (E10)
6-10 years	2 (E5, E19)
11-15 years	4 (E1, E2, E13, E17)
16-20 years	3 (E6, E12, E18)
21-25 years	4 (E4, E7, E8, E15)
26+ years	5 (E3, E9, E11, E14, E16)

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.

3.2 A Definition for SATF

In this section, we present the definitions of SATF provided by the literature (Section 3.2.1) and summarize them in order to create a single and coherent definition for this term (Section 3.2.2).

3.2.1 Definitions from the Literature

Each of the works we analyzed 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 [57] and [18] modify test cases that match the specification at runtime based on system and environmental conditions. The work in [15] responds to changes in operations, operation arguments, and service composition. Likewise, the work in [16] examines modifications made to how the service is utilized or provisioned to trigger testing sessions. On the other hand, the work in [14] monitors the system to find untested configurations and activate the testing. Similarly, the work in [68] 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 [62] 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 [60] support the notion that an "autonomous self-organizing 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 behavior 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 [55] 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 [56] 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" [56]. 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 [61]. 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 [64] 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 [67] 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 behavior model of the testing session. Accordingly, to test different scenarios of a microservice-based case study application, the work in [58] defines and automatically enacts live testing based on data collected by metrics providers or external services. In [66] 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 [63], focuses on functional and regression testing of service orchestrations during runtime with the goal of identifying errors caused by evolutionary behaviors, 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 [17] 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 [65] 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 [59] 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".

3.2.2 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 [1], 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 on SATF aforementioned and the common main concepts provided by it, we adapt the above general definition as follows:

Definition 3.1 (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.

3.3 A Taxonomy for SATF

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. In this chapter, we present a graphical representation of the proposed taxonomy using a feature model. Consequently, throughout the text, we use the terms "features", "components", and "dimensions" interchangeably to refer to the characteristics that a SATF approach may encompass.

We found that the main components of a SATF approach are: Monitor (optional), Test Cases, Oracle (optional), Human Involvement(optional), Test Strategy, and Adaptation. They are described in the following subsections.

3.3.1 Monitor

Monitoring is the process of continuously acquiring, analyzing, and collecting information regarding the SUT execution [1]. 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 behavior 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 Continuous Integration and Continuous Deployment (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 dimension 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" [11], or *Discontinuous* (named Adaptive in [11]) 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 [14–16, 55, 57–64, 66–68], and one of the papers in the survey use discontinuous monitor [18]. Some approaches, strictly speaking, do not use a monitor of any kind [17, 65], but they may rely on a human operator that acts as an external monitor. This is the case, for instance, in [17], 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* [16, 57–59]. 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, 62, 64, 67] or *External Component Change* [58]. 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' behavior in using the system. Lastly, the *SUT Adaptation or Evolution* [15, 55, 61, 63, 66, 68], *Model Activities* and *Test Results* [18, 60] 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 [17] and [65] do not present a monitor.

C. *Foreseeability* concerns whether "change can be predicted ahead of time" [13]. The approaches are categorized based on their degree of foreseeability: *Foreseen* (taken care of) and *Foreseeable* (planned for) [13]. 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 [14–16, 18, 55, 57, 59–64, 66–68]. One approach is classified as foreseen [58]. As aforementioned, the works described in [17] and [65] do not present a monitor.

3.3.2 Test Cases

Test cases are a mandatory dimension in a SATF approach. According to [69], 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 [56] 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 [66], 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 [68] 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 (presented later in this chapter).

One important component of a test case is the expected result, that, is, the Oracle. For instance, the work in [18] 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 below), because we observed that not all papers actually consider the oracle as embedded within the test cases.

3.3.3 Oracle

We qualify Oracle as an optional dimension since it is not 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 the similarity of the answer with a standard error message. In the work described in [68], 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 a test-dependent oracle might be found in [66], where an oracle is also an object of the adaption as explained in the following (see Section 3.3.6). 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 [68].

3.3.4 Human Involvement

This optional dimension describes who is the agent of adaptation. There is *No Human Involvement* [16, 18, 55, 57, 59–61, 63, 65] in some approaches. However, we also found works that call for *Human Involvement* [14, 15, 17, 58, 62, 64, 66–68]. Testing a Multi-Tenant Software as a Service (SaaS) is the main goal of the work in [17]. 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 [14] necessitates the assistance of developers when unknown configurations are discovered. The work in [15] demands human involvement for analyzing test reports by the service provider. Then, the work in [62] relies on humans not only to analyze reports but also to choose whether to trigger online tests. 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 [58], that uses a dashboard for the developer to visualize the outcome of executed checks and, based on this, takes decisions.

3.3.5 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 [1] categorizes field-based testing approaches into *Ex-vivo* [16, 67], *Offline* [61, 68] and *Online* [14, 15, 17, 18, 55, 57–66], 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" [1]. The approach in [16], 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" [1]. The work in [68] conducts offline testing, by launching the tests in a sandboxed environment, whereas the work in [61] performs both offline and online testing. Finally, *Online Testing* comprehends testing approaches "performed in the production environment on the actual system" [1]. 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* [14, 15, 17, 18, 55, 57–63, 65, 66, 68] or *Non-Functional* [16, 58, 64, 67] faults in order to fulfill the requirements. The work in [67] assesses two non-functional requirements of microservice systems, reliability and performance. On the other hand, the work presented in [60] employs self-tests at runtime to identify a malfunction of a self-organized 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* [16, 55, 58, 59, 61, 63, 66], 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* [55, 62, 64, 67, 68], i.e., the testing is driven by the usage profile in operation; *Specification-based* [17, 60, 65], in which the tests are black-box functional ones and are derived from the system requirements or an informal specification; *Evolutionary* [18, 57], 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 [15], and another one applies *Combinatorial* criteria [14].

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* [14, 55, 58–60, 68], i.e., each module or component is tested in isolation; *Integration Level* [14, 55, 59, 63, 65], in which a group of related modules is tested in each iteration; and *System Level* [14–18, 56, 57, 61–67], that is the level where the whole integrated application is examined in its entirety. For instance, the work in [60] 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 [65] employs integration testing to the complete testing of composing services. Finally, the approach in [57] 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.

3.3.6 Adaptation

In a 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) [18, 57, 61, 62] or *creating new ones* [14–17, 63, 65, 66, 68]. 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 [16]. In other scenarios, it is unavoidable because test cases may become invalid as a result of system adaptation [57]. Existing test cases may also be adapted, as mentioned, e.g., in [18, 57, 62]. Some approaches also include multiple objects that need to be adapted. The work in [18], 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., [66]). Different types of oracles are used by field-based techniques to determine the results of tests [1]. 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 [66]. In their work, oracles are built at runtime using the extended Context-aware Feature Model (eCFM) [70], 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 of 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, 17, 18, 55, 58–61, 64, 67]. Possible adaptations include, for example, the adaptation of the test plan or timing when test cases are periodically scheduled for execution. The work in [59] 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 [17], which adapts the test case selection and the workflow regression testing for multi-tenant 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" [1]. The adaptation may be triggered or activated (i) *periodically* [65], (ii) by internal events of the *SUT* or changes in its environment or technical resources [14, 16, 18, 55, 57, 59, 61–63, 66, 67], (iii) by some *policy* [15, 18, 57, 58, 68], or (iv) on a *request* of a testing session from testers, runtime infrastructure/container, etc. [17, 57, 62, 64, 65]. One of the experts we interviewed (the expert E10) suggested considering also (v) "*changes in the users' behavior*" [16, 67] (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 [14, 57], a change in a technological resource [18], or changes or adaptations made to the SUT [59, 66], 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 [57]

or the tenant administrator [17]. Lastly, the adaptation trigger for the work in [60] 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 [12], can be separated into two sub-dimensions: *Reactive* [14–17, 55, 58, 59, 61, 63, 65–68] and *Proactive* [18, 55, 57, 60, 62, 64]. By the general description in [11] 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 [55], where external and local system faults are foreseen and the testing process self-adapts to address them. The work in [62], 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* [14–18, 55, 57–68] and *Dynamic*. This categorization was proposed in the taxonomy for self-adaptive software described in [11]. 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 behavior connected to both functional and non-functional software requirements [11]. 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?" [12], i.e., what adaptations are necessary based on what is observed. The study in [12] divides adaptation techniques into three categories: *Parameter* [18, 64, 67], *Structure* [14–18, 57, 59–63, 65, 68], and *Context* [55, 58, 66]. 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 [15, 62], test placement [59] or test plan and identification of where to conduct the test [61]. Changes in how technological resources, environment/users, or other components relate to one another are also comprehended [12]. Lastly, context techniques deal with changes in the context, such as adapting the testing rate in accordance with the

usage rate [55]. There is also the option of combining these three adaption strategies, as in the work in [18] 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 [11]. In this way, approaches can be divided into *Internal* [14–18, 55, 57–63, 65–67] or *External* [64, 68]. 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 [64], 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 [11]. A *closed-adaptive* system [14–18, 55, 57–65, 67, 68] has a predetermined number of adaptive actions; hence, no additional action may be added at runtime. In contrast, an *open-adaptive* system [66] 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 [66], which is open, all of the surveyed approaches are closed. 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 [12], without considering the application logic. A *centralised adaptation* [15–18, 55, 57, 59, 61–68] logic is advised for small systems with minimal resources to handle. In contrast, a *decentralised adaptation* [58, 60] 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* [14] 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 [60] 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 [14] performs decentralised in-vivo tests, it is considered as hybrid since the overall testing operations are coordinated by a central server.

3.3.7 Relations among the Dimensions

In this section, we provide an overall view of the various dimensions described in Sub-Section 3.3. Figure 3.3 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 one 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 parent is defined, so all the children must be. It is important to highlight that, both *Alternative* and *Or* symbols also mean that once the parent has been defined, at least one child also must be. As described above, and visible in Figure 3.3, 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[5]:

[5]The character "." is used to navigate the feature model in Figure 3.3. Also, we assume that a feature is TRUE when it is active. For instance, SATF.Monitor=TRUE when a monitor is present, FALSE otherwise.

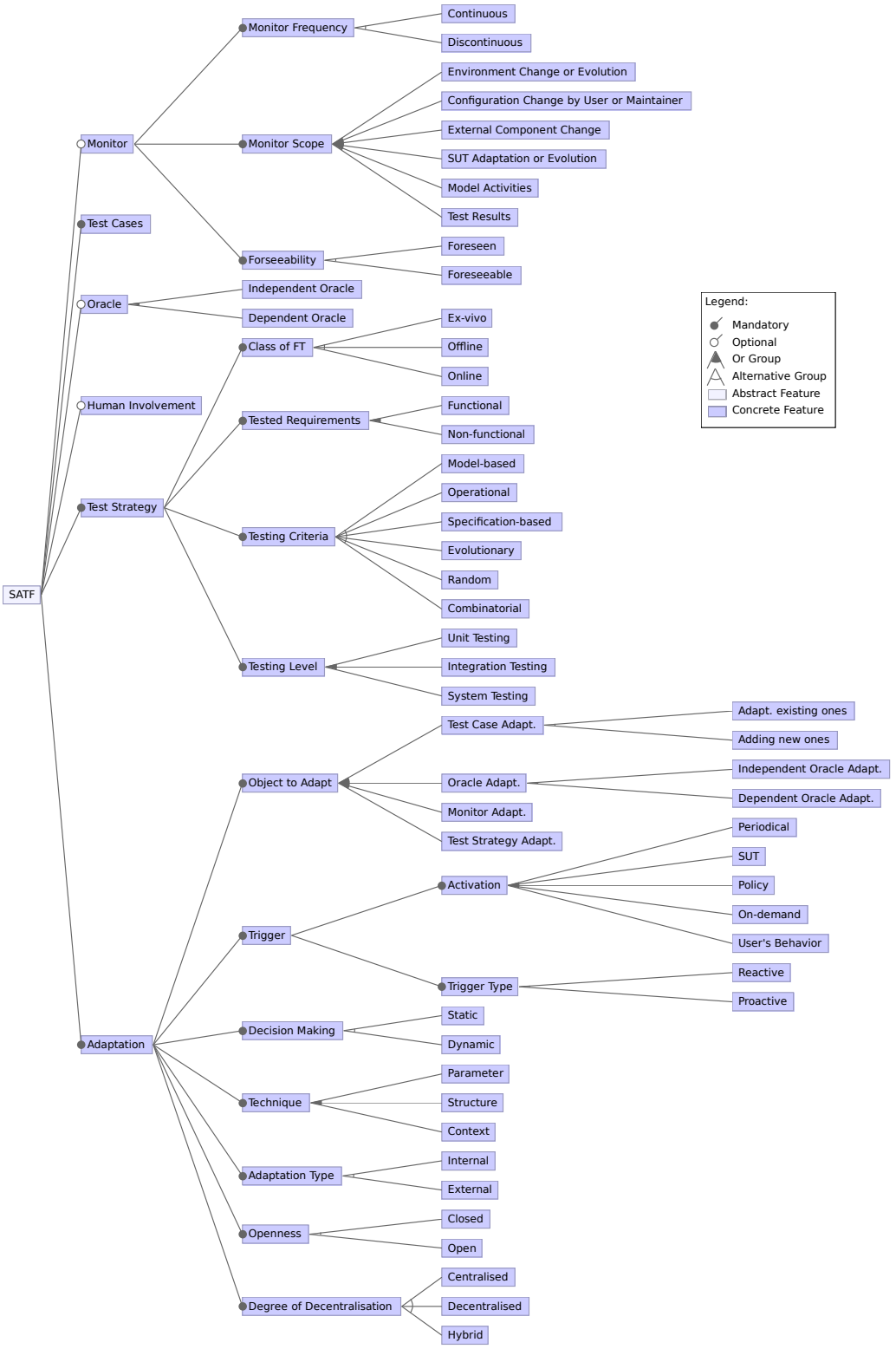


FIGURE 3.3: Feature model of the dimensions for SATF approaches.

- SATF.Adaptation.“Object to adapt”.“Oracle Adapt.”.“Independent Oracle Adapt.” $\Longrightarrow$ 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.” $\Longrightarrow$ 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 $\Longrightarrow$ 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 $\Longrightarrow$ 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” $\Longrightarrow$ 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.

- 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.

3.4 A Reference Architecture for SATF

In this section, we aim to provide 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 [71];

then, a concrete architecture can be considered as a specific instance of a software reference architecture. For instance, the work in [72] presents a reference architecture for connected vehicles. In this work, we contribute a reference architecture for SATF approaches. It has been built by taking into account the primary studies that have been identified and analyzed in the literature review. Since we made an effort to generalize and interpret the data collected by analyzing 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 Sub-Section 3.1 and the results of the validation are described in Sub-Section 3.6.

An architecture can be described in various ways and it can be organized in various views and viewpoints [73]. 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 3.4.

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-analyze-Plan-Execute over a shared Knowledge) feedback loop as style.*

Rationale: The MAPE-K loop is the preeminent standard control model for autonomic and self-adaptive systems [74, 75]. 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, SATF 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

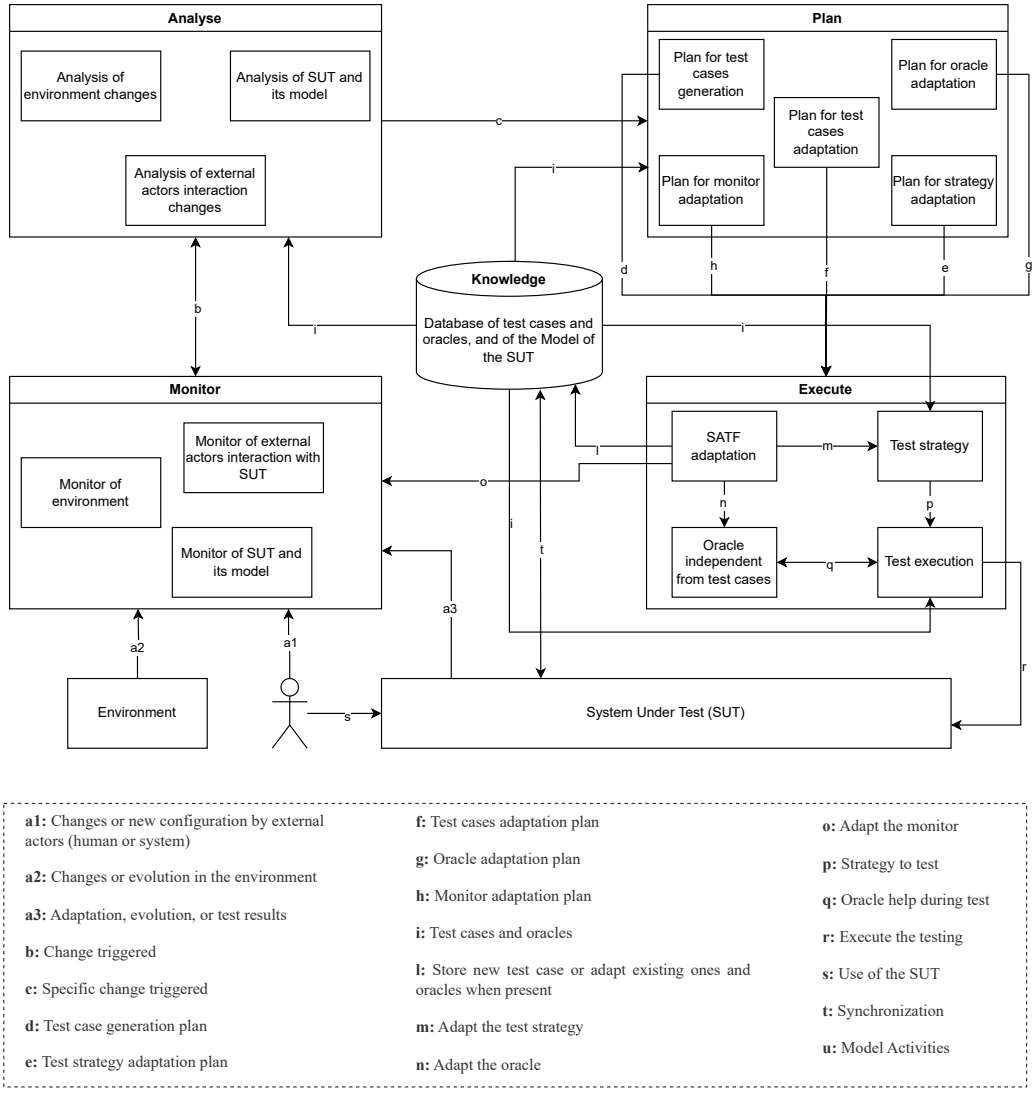


FIGURE 3.4: Reference architecture for SATF approaches.

unnecessarily complex. However, we have examples of human-in-the-loop involvement in the surveyed papers [14, 15, 17, 62] 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 3.3 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 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 [65] (or simple component) since the trigger of adaptation might come directly from a CI&CD release [58] or a user action [67].

AD3: *The analysis component should decide which adaptation should be planned and, then, it should analyze (i) environmental changes, (ii) external actors' interaction changes, and (iii) SUT changes.* External actors' interaction changes can concern changes in user behavior 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 analyzed. The connection between the monitor and the analyze phases is bidirectional since the analysis might trigger the need for further data to be monitored. The *Analysis* 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, based on the change observed by the monitor that triggered the need for adaptation, this component decides which type of adaptation should be planned. Referring to Figure 3.4, this aspect is hidden within the *c* label connecting the *analyze* 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 3.3, 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 3.4, 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 specialized 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 [14], in which tests are only executed when an untested configuration is observed and new test cases are generated. As shown in Figure 3.4 we have a dedicated component in the *Execute* phase to manage the SATF 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 3.4 and as discussed in AD4, the SATF adaptation component is responsible for the adaptation of the Monitor, Test strategy, and 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 the MAPE-K loop. This was suggested by an expert (E10).

It is important to highlight that Figure 3.4 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, analyzer, 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 3.4 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 analyze 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 serve as a conceptual model, allowing us to delegate the enhancement and optimization of specific testing approaches to developers, particularly those tailored for particular classes of SUTs.

3.5 Challenges in SATF

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 present a list of these challenges and gaps, along with additional specific and minor issues related to SATF that were identified.

3.5.1 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 [57], and of course, the testing system itself reflects such uncertainty. For example, there may be an exponential number of setups when testing mobile applications [14]. The study in [18] 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 [59], and the testing methodology should be equipped to manage them. In [56], 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. Besides, we might never be certain about the underlying state of a self-adaptive system we are testing.

3.5.2 Overhead

Several testing strategies are concerned with the overhead in terms of memory, network, and execution time [76]. 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 [14]. The method in [14] examines the overhead imposed by the monitoring stage and determines that it is imperceptible to negligible in their case. The research in [59] 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 [66]. The strategy outlined in [56] 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 affect system performance. For the strategy described in [16], 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 [18] 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 [65] states that the user's response time has increased as a result of an overhead that they are unable to handle.

3.5.3 Human Intervention

In general, self-adaptive systems might benefit from human involvement, since human operators can complement the capabilities of systems [77]. 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., [14, 15, 17, 62].

3.5.4 Test Isolation

Test isolation is a well-known problem for any field testing strategy, as noted in [1]. 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" [1]. 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 [66]. The system's sensitivity indicates

which testing operations, when carried out, interfere unfavorably with the environment or the system that is now running [78]. The work in [56] isolates specific test cases to stop failures from impairing the functionality of the real system. Additionally, runtime adaptations of Component-Based Systems (CBSs) may cause new defects to appear, which could cause malfunctions and put the execution of the system in an unsafe state [59]. Even though the work proposed in [16] 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 [17] 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 [14] isolates the runtime testing session from the regular user session by using managed profiles[6].

3.5.5 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 3.5. 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.

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

- Proper reaction to identified malfunctions [60];
- Provisions for reliable traceability between test cases and requirements [56];
- Constraints arising from the specific techniques employed [15, 62] (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 [64];
- Building strengthening oracles and increasing the level of automation of the approach [68];

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

- 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 [58];
- Specification of adaptation constraints [56]; and
- Strategies to make test events indistinguishable from normal events [60].

3.6 Validation

In this section, we discuss the validation of the proposed taxonomy and reference architecture conducted with experts through a questionnaire and further discussions through online meetings. The experts are the 19 authors who responded to the questionnaire, as described in Section 3.1.2. It is important to notice that the same person may be the author of more than one work. Also, some of these experts were co-authors of the same paper. The questionnaire contained five main questions and a last question asking whether the respondent was available for an interview during an online session. We, then, conducted the interviews with the 5 experts who responded positively to this last question, aiming 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 on 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

Table 3.5: Summary of the main challenges in SATF.

Challenge	Cause	Impact	Possible Solutions
Uncertainty	• SUT may encounter unforeseen operational challenges [57] (e.g., exponential number of setups for the SUT [14]).	• Difficulty adapting testing to a self-adaptive system as it reconfigures [18]. • System fails due to configurations never tested [59]. • Environment or requirement changes impact test cases and/or oracle behavior [56]. • Difficult to determine what to monitor and when to test [56].	-
Overhead	• Use of resources to make adaptations.	• Causes overhead that could impair the system's performance.	• The overhead incurred should be minimal to ensure acceptable testing for end-users [14]. • Monitor overhead to ensure it remains imperceptible [14]. • Reduce the frequency of dynamic adaptations [59]. • Small test sequences might incur little overhead [66]. • Schedule tests to minimize the impact on system performance. [56].
Human Intervention	• Testing of critical systems may demand human intervention to complement the system's capabilities [77].	• Incurring computational and financial costs.	• Make the approach as self-adaptive as possible.
Test Isolation	• Testing process and the SUT run in the same environment.	• Testing interferes with production processes and data [1]. • Give unintended access to critical system components. • Introduction of defects, leading to malfunctions and unsafe system states [59]. • Risk the proper running of the service [16]. • Operational database may be compromised [17]. • Risk of modifying mission-critical data or sharing it with unauthorized parties [17].	• Minimize system sensitivity to runtime execution to reduce risks [66]. • Isolate test cases to prevent failures from affecting the real system. [56]. • Use of managed profiles to isolate the runtime testing session [14].

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 a monitor, e.g. [77]. 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 who 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 the 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. 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 3.6 presents the proportion of responses for each question in the questionnaire. Interestingly, we observed that some co-authors of the same paper provided different responses to the same questions.

TABLE 3.6: 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)

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 analyzed;
- Bidirectional communication between "analyze" 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 of SATF approaches and that we are heading in the right direction in order to establish common definitions and terminologies for this still-evolving topic.

3.7 Limitations and Threats to Validity

This research has been performed by following well-established guidelines for empirical studies in software engineering including systematic studies [79], surveys [80], and interviews [81]. We identified the main threats to validity and mitigation strategies according to [82], as described in the following subsections.

3.7.1 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 [82]. We defined and followed some strategies to mitigate these threats 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 previously been co-authors with one of the supervisors of this thesis. However, either the collaboration covered a quite different topic than SATF (3 experts), or the only paper co-authored was the secondary study in [1] (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 author of this thesis, who had no previous acquaintance with them.

3.7.2 Threats to Construct Validity

Threats to construct validity focus on the relation between the theory and the observation [82]. 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 guaranteeing that at least two individuals (among the thesis's author and her supervisors) look 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.

3.7.3 Threats to External Validity

External validity threats relate to the generalisability of the final observed results and outcomes of the study [82]. 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 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.

3.7.4 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 [82]. 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 hypotheses, 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 applicability.

Chapter 4

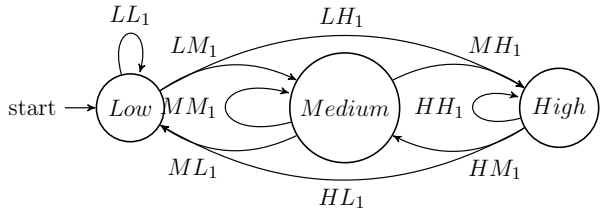
GATE4BSN - Generic Approach to TEsting for BSN applications

In this chapter, we present our findings aimed at understanding effective testing strategies for Body Sensor Networks (BSNs)[46]. The testing strategies we present in this chapter are in-house, since we simulate the data coming from a patient, and are not adaptive. Self-adaptive in the field testing approaches will be presented in the next chapter. Specifically, in this chapter we introduce *GATE4BSN*, an in-house approach designed for testing BSN applications. It will be then instantiated in three different approaches (see Sections 4.3, 4.4, and 4.5), *PASTA*, *ValComb*, and *TransCov*. Before presenting *GATE4BSN* (Sec. 4.2), we make use of a running example, which is introduced in Sec. 4.1.

4.1 Example of a Simple BSN

As a running example, we consider a simple BSN containing two sensors, Sensor 1 and Sensor 2. We also suppose that the possible risk levels for Sensor 1 are *Low*, *Medium*, or *High-risk*, while Sensor 2 can range from *Low* to *High-risk*. The behavior of these sensors may be graphically described by DTMCs and their respective transition matrices, as shown in Figures 4.1a and 4.1c, and Figures 4.1b and 4.1d for Sensor 1 and Sensor 2, respectively. The states in these DTMCs are labeled with the risk level they are representing and the transitions with the probability of a determined sensor moving from one

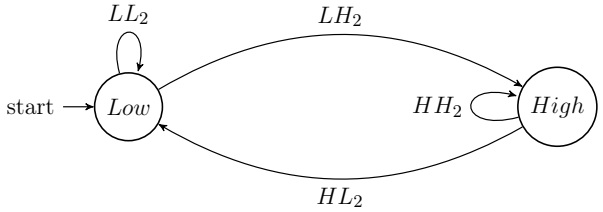
risk level to another. In the context of BSNs, the rows and columns of a transition matrix represent the risk levels associated with a sensor, and cells represent the probability of transition from one risk level to another. For instance, in Figure 4.1b, $TM1_{3,2} = HM_1$ indicates that Sensor 1 varies from *High* to *Medium* risk with a probability equal to the value of HM_1.



(A) DTMC of Sensor 1

$$TM1 = \begin{bmatrix} LL_1 & LM_1 & LH_1 \\ ML_1 & MM_1 & MH_1 \\ HL_1 & HM_1 & HH_1 \end{bmatrix}$$

(B) TM of Sensor 1



(C) DTMC of Sensor 2

$$TM2 = \begin{bmatrix} LL_2 & LH_2 \\ HL_2 & HH_2 \end{bmatrix}$$

(D) TM of Sensor 2

FIGURE 4.1: Example of DTMCs and transition matrices (TMs) for Sensor 1 and Sensor 2.

The same risk levels for different sensors may also be associated with different risk value ranges. In this running example, to make it simple, let us suppose that: for Sensor 1, the risk levels *Low*, *Medium*, and *High* correspond to the risk value ranges [0,10], [11,20], and [21,30], respectively; for Sensor 2, the risk levels *Low* and *High* correspond to the risk value ranges [0,15] and [16,30], respectively. Thus, for each sensor, each risk level corresponds to a single risk value range.

The representation of the sensor behavior as a DTMC is employed in *TransCov* and *PASTA*, while for *ValComb*, there is no need to use DTMCs. The idea behind the representation of a patient through DTMCs is that patients more prone to diseases may have higher probability values in the arcs that enter the "High-risk" state. On the other hand, healthy patients may be represented by higher probabilities in the arcs to the "Low-risk" state.

4.2 A Generic Approach to TEsting for a BSN application (*GATE4BSN*)

Figure 4.2 provides an overview of *GATE4BSN*, which takes 4 main stages. First, in stage ①, we either collect data from sensors that are connected to a real patient or simulate the patient's behavior through the use of some strategy. Then, this data is sent to a running BSN that in ② logs at predefined intervals the patient's situation (Sensors Reading) and the patient's diagnosis provided by the BSN (BSN outcome). Next, based on the Sensor Reading, the expected diagnosis (Expected Outcome) is computed in ③, and compared to the BSN outcome in ④. The testing results are finally provided. It is important to emphasize that *GATE4BSN* may be easily adapted to test other systems with characteristics similar to BSNs. In the following, we walk through all stages while referring to the running example in Figure 4.1.

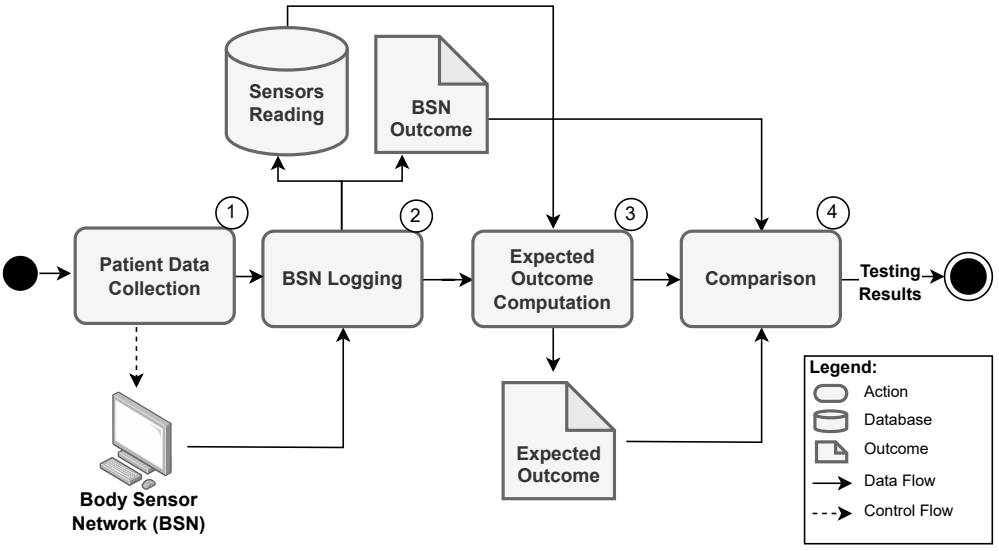


FIGURE 4.2: Overview of the Generic Approach to TEsting for a BSN application (GATE4BSN).

Patient Data Collection. Stage ① corresponds to the collection of sensor data from a patient. This patient may be either a real patient to which there are some sensors connected or a determined model or structure that simulates the behavior of a patient for a determined period of time. Sections 4.3, 4.4, and 4.5 provide a detailed description of this stage for the three proposed approaches, *PASTA*, *ValComb* and *TransCov*, respectively. The control flow arrow launches the function in a running BSN to process the collected data. Note that even if in the proposed approaches we allow the simulation of

unusual and edge cases, aiming for a more comprehensive evaluation of the BSN, for all of them the Patient Data Collection stage may be easily adapted by adding a constraint to it to avoid the generation of unrealistic inputs for the BSN.

BSN Logging. Stage ② consists of logging the BSN while it is being executed to collect sensor readings, which are the data about the different patients at every instant of time, and the BSN Outcome. We provide a more formal definition for Sensor Reading as follows.

Definition 4.1 (Sensors Reading)**.** The reading of all the sensors of a patient $p \in P$ at time t is denoted as $SR_t^p = (SV_{p,t}^1, SV_{p,t}^2, \cdots, SV_{p,t}^n)$, where P is the set of patients, and $SV_{p,t}^i$ is the value of sensor i at time t for patient p, with $1 \leq i \leq n$ and n is the number of sensors.

Then, we can define BSN Outcome. Let $Patient_{Risks} = \{o_1, \cdots, o_k\}$ be the set of possible outcomes of the BSN, that is, the possible patient risk levels. The definition of BSN outcome is formally provided as follows.

Definition 4.2 (BSN Outcome)**.** BSN outcome $BO(p, t, SR_t^p) = o$ is a function that takes as input a patient $p \in P$, an instant of time $0 \leq t$ and SR_t^p, i.e., the sensors reading of patient p at time t, and returns the BSN outcome $o \in Patient_{Risks}$.

Table 4.1 provides some fictitious instances of Sensors Readings and BSN Outcomes for the running example previously described in Section 4.1. The columns indicate the patient ID, the instant of time, the sensor values, and the BSN Outcome based on the sensor values, respectively. The first four columns compose Sensors Readings. For instance, the first row refers to patient 0 and time 0, and the values of sensors reading by SR are as follows: $SR_0^0 = (5, 10)$. The last column contains the labels which are the BSN outcomes. Still referring to the first row, "Low Patient Risk" is the value calculated by $BO = (0, 0, (5, 10))$. Note that BO correspond to a patient risk level in $Patient_{Risks}$. $Patient_{Risks}$ encompasses all potential patient risk levels for a given BSN, representing the varying degrees of patient criticality. It is important to note that this is distinct from the sensor risk level, which is determined by an individual sensor.

This stage is very relevant since when simulating a patient through a set of DTMCs, for example in *PASTA* and *TransCov*, to understand what the current patient risk level is,

we may need to know at each instant of time in which state the DTMC is in. By logging in this set of information from BSN, we are able to understand the patient's situation according to the simulated sensors.

TABLE 4.1: Examples of fictitious Sensors Readings and BSN Outcomes.

Sensors Reading				**BSN Outcome**
Patient	**Time**	**Sensor 1**	**Sensor 2**	**Label**
0	0	5	10	Low Patient Risk
0	1	15	18	Moderate Patient Risk
1	0	22	30	Critical Patient Risk
1	1	21	28	Critical Patient Risk

Expected Outcome Computation. In Stage ③, the Expected Outcome, that is, the Oracle, is computed based on the Sensors Readings. In other words, this phase provides the expected overall risk level of a patient at a determined instant of time according to the data provided by each Sensors Reading entry. The Expected Outcome is formally defined as follows.

Definition 4.3 (Expected outcome)**.** Expected outcome $E\_BO(p, t, SR_t^p) = e\_o$ is a function that gets as input a patient $p \in P$, a time instant $0 \leq t$, and a sensors reading for the patient p at time t, i.e., SR_t^p, and returns the expected outcome $e\_o \in Patient_{Risks}$.

The computation of the $E\_BO(p, t, SR_t^p)$ function should be performed by using a rule created by the domain expert (i.e., a doctor). That is, the expert, based on her/his knowledge about the domain, creates a function that takes as input the sensor values and outputs the expected outcome. The Expected Outcome is computed for all the Sensor Readings generated in the previous stage. An example of this function for the running example could be:

$$E\_BO(p,t,SR_t^p) = \begin{cases} \text{Low PR}, & \text{if } \forall SV_{t,p}^i,\ SV_{t,p}^i \leq 10, \\ \text{Moderate PR}, & \text{if } \forall SV_{t,p}^i,\ 11 \leq SV_{t,p}^i \leq 20, \\ \text{Critical PR}, & \text{if } \forall SV_{t,p}^i,\ 21 \leq SV_{t,p}^i \leq 30. \end{cases}$$

where $1 \leq i \leq n$, PR stands for "Patient Risk" and the set of possible patient risk levels $Patient_{Risks}$ is defined as {"Low PR", "Moderate PR", "Critical PR"}.

This function states that when all sensors provide values lower than or equal to 10, the expected outcome is "Low PR". If the sensors provide values from 11 to 20, the expected outcome is "Moderate PR". Otherwise, the expected outcome is "Critical PR". These rules must be created so that they cover all possible values within the sensor-value ranges RL_i.

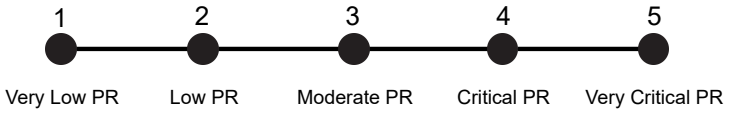


FIGURE 4.3: Example of possible Patient Risk Levels.

Comparison. Finally, in stage ④ the BSN outcome and the Expected Outcome (i.e., the Oracle), for a determined Sensors Reading, are compared. For each test case, it is decided if the test passed or not, whereby we define a Test Case as follows:

Definition 4.4 (Test Case)**.** A test case $TC = (SR_t^p, E\_BO(p, t, SR_t^p))$ with a patient $p \in P$, the set of patients, and $0 \leq t$, is a tuple where:

- SR_t^p is a Sensors Reading as described in Def. 4.1;
- $E\_BO(p, t, SR_t^p)$ is the Expected Outcome as described in Def. 4.3.

Then, for each TC so defined, we compare its Expected Outcome against the actual BSN outcome using the function $Comp$.

Definition 4.5 (Comparison function)**.** Given a patient $p \in P$, an instant of time $0 \leq t$ and SR_t^p, $Comp$ is a comparison function that takes as input the BSN outcome $BO(p, t, SR_t^p)$ and the Expected Outcome $E\_BO(p, t, SR_t^p)$, and returns a Testing Result $\in (Pass, Fail)$.

We implement the $Comp$ function based on a generic notion of distance between expected and actual outcomes. Let us suppose that the possible patient risk levels $Patient_{Risks}$ is {"Very Low PR", "Low PR", "Moderate PR", "Critical PR", and "Very Critical PR"}. We then assign to each of these risk levels an integer number, as for instance in Figure 4.3. To account for possible low oscillations of sensor values we assess that a test case passes if the difference between the Expected Outcome number and the BSN Outcome number is lower than 2. Otherwise, it does not pass. Thus, for example, if the BSN provides as outcome "Low PR" (2) and the Expected Outcome is "Moderate PR" (3), we consider the test successful (3-2=1); if instead, the BSN provides "Low PR" (2) and the Expected Outcome is "Critical PR" (4), the test fails (4-2=2).

4.3 PASTA - PAtient Simulation for Testing of bsn Applications

In this section, we show how *PASTA* implements the stage ① of *GATE4BSN* described in Section 4.2, the Patient Data Collection. This stage is composed of the following substages: ⓐ Transition Matrices Definition, ⓑ Transformation of Matrices into Test Patients, and ⓒ Running Test Patient, as shown in Figure 4.4. We describe them as follows.

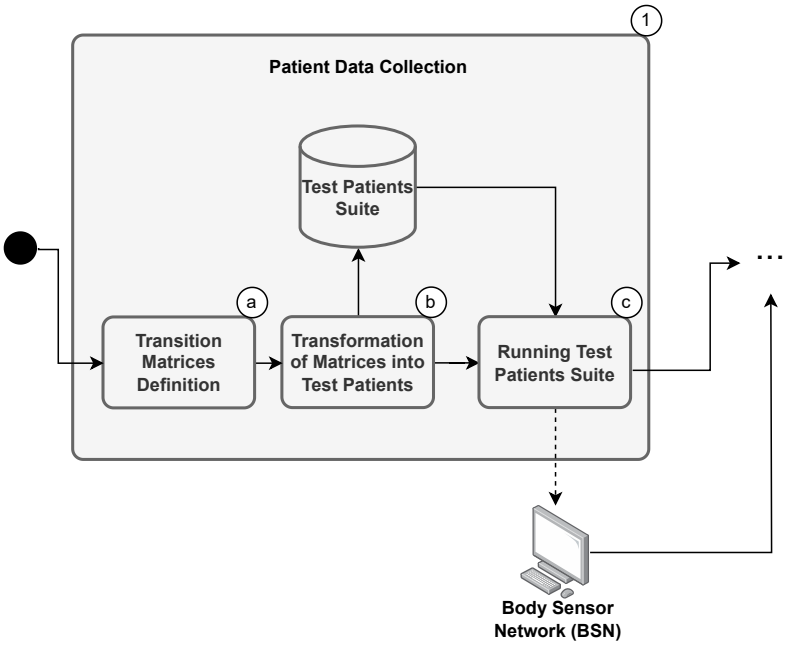


FIGURE 4.4: Patient Data Collection for *PASTA*.

Transition Matrices Definition. In stage ⓐ testers using *PASTA* define a set of transition matrices, which we use to simulate, during testing, the vital signs that would be read by the sensors while monitoring one person. Formally, in *PASTA* we define a sensor as follows.

Definition 4.6 (Sensor)**.** A Sensor is defined as a quadruple $S = (Freq, m, RL, TM)$ where:

- $Freq$ is the change frequency in the states of the sensor DTMC;
- m is the number of states in the DTMC;

- $RL = ((Min^1, Max^1), \cdots, (Min^m, Max^m))$ are the Risk Levels of the sensor. Each risk level is defined as a range, i.e., minimum and maximum values, for each state in the DTMC of the sensor;
- $TM = (p_{11}, p_{12}, \cdots, p_{mm})$ are the values assigned to the transition matrix of the sensor.

To control the probability of going to a determined state regardless of the state of origin, we restrict the possible variations of each transition matrix TM of a Sensor S, by requiring that all the rows of the same matrix are equal. In other words, the probability Pc_j, with $1 \leq j \leq m$, of each column j in the transition matrix TM is:

$$Pc_1 = p_{11} = p_{21} = \cdots = p_{m1}$$
$$Pc_2 = p_{12} = p_{22} = \cdots = p_{m2}$$
$$\cdots$$
$$Pc_m = p_{1m} = p_{2m} = \cdots = p_{mm}$$

Thus, referring to the example in Figure 4.1, for Sensor 1 we assume that Pc_1=LL_1= ML_1=HL_1, Pc_2=LM_1=MM_1= HM_1 and Pc_3= LH_1=MH_1=HH_1. In this way, the transition matrix TM of Sensor 1 is fully defined by three values: Pc_1, Pc_2, and Pc_3. Note that this is a practical restriction we made to better control our approach, but that can be easily removed. Also, for practical reasons, we consider that the value of the probability Pc_j is given in percentage (i.e, values from 0 to 100) instead of fraction (i.e., values from 0 to 1).

Stage ⓐ aims at defining the values of TM (i.e., of Pc_1, ..., Pc_m). As a test strategy, we considered three different scenarios:

i) All arcs in the DTMC have the same probability, i.e., from the current sensor risk level, any risk level can be reached next, or intuitively the patient has no specific medical profile.

ii) All arcs in the DTMC that enter one determined state have a probability equal to 100% and all the remaining arcs are set to 0. In other words, from wherever in the DTMC, the same risk level is always reached, or the patient has a very clear diagnosis.

iii) All arcs in the DTMC that enter one determined state have a probability equal to 0, i.e. this risk level is never reached, and all the other remaining arcs have equal probabilities.

As the values of TM are transition probabilities, we also define a constraint to ensure that the sum of values referring to the transitions from the same sensor state is equal to 100%:

$$Pc_1^i + Pc_2^i + \cdots + Pc_m^i = 100 \tag{4.1}$$

where i denotes the sensor and m the number of states in the DTMC of sensor i. Considering the example in Figure 4.1, the constraint states that $Pc_1^1 + Pc_2^1 + Pc_3^1 = 100$ and $Pc_1^2 + Pc_2^2 = 100$.

Transformation of Matrices into Test Patients. Stage ⓑ consists of generating and transforming transition matrices describing the behavior of all sensors into a "Test Patient": by this term, we consider the description of the vital signs' behavior collected by a whole set of sensors that are all applied to one person. Formally, we define a Test Patient as follows.

Definition 4.7 (Test patient)**.** A test patient $TP = \{S_1, \ldots, S_n\}$ is a set of n sensors.

Considering the example in Figure 4.1, the Test Patient there is defined as $TP = \{S_1, S_2\}$, where $S_1 = (Freq_1, TM1, RL_1)$ and $S_2 = (Freq_2, TM2, RL_2)$. The frequencies $Freq_1$ and $Freq_2$ must be defined as well as the pairs of risk values RL_1 and RL_2. For RL_1, we should define the minimum and maximum values for the *Low*, *Medium*, and *High* risk levels. As for RL_2, it is necessary only to define the *Low* and *High* risk levels. Finally, the matrices $TM1$ and $TM2$ are the outcomes of this process.

In particular, a test patient is characterized by the combination of transition probabilities in the TMs associated with the sensors applied to the patient's body, which we refer to as a t-way combination.

Definition 4.8 (T-way combination)**.** A t-way combination $\mathbf{C} = (PC^1, PC^2, \cdots, PC^n)$ is a tuple containing the probability of each column in the transition matrix TM of the n sensors connected to a Test Patient. In turn, for $1 \leq i \leq n$, $PC^i = (Pc_1^i, Pc_2^i, \cdots, Pc_{m_i}^i)$

is a tuple containing the probability of each column in the transition matrix TM of a sensor i, and m_i the number of states in the DTMC of sensor i.

We apply combinatorial testing techniques to derive a minimum set of t-way combinations of the parameters in **C**. Considering again the example in Figure 4.1, its t-way combination is $\mathbf{C} = (Pc_1^1, Pc_2^1, Pc_3^1, Pc_1^2, Pc_2^2)$. The combinations generated by the combinatorial strategy are then converted into transition matrices (as the ones in Figure 4.1) by positioning the corresponding Pc_j^i values in the matrices as previously explained. Then, by combining the different TMs that are generated for each sensor we can obtain the data of a patient.

In *PASTA*, to test the BSN behavior, we assume that the same BSN can be used to monitor the health status of different individuals with differing health characteristics. Hence, we generate a collection of test patients that share the same set of sensors equally configured (the same frequency and risk levels) but may expose different health statuses; this is reflected in defining different values for the transition matrices across different persons. To denote this set of test patients we introduce the term "Test Patients Suite".

Running Test Patients Suite. In stage ⓒ, the BSN is executed by taking each of the test patients in the test patient suite as input. *PASTA* makes use of the test patient data to simulate a patient. In this sense, it creates and executes the DTMCs corresponding to the sensors as defined for a certain amount of time, which is a parameter set by the tester. It is important to highlight that the simulation of a patient works by randomly sampling values within the range of the current state (risk level) of a DTMC. For example, if the current state of Sensor 1 DTMC is "Low", and the range (Min^L, Max^L) for the RL "Low" has been set to [65,100], a possible number that could be generated for this sensor is 70. Then, from time to time, the BSN provides an overall patient risk level (*BSN Outcome*, as explained before) based on the combination of these values for all the sensors.

4.4 ValComb - Sensor Values Combination

In this section, we show how the *ValComb* (Sensor Values Combination) approach implements the stage ① of our generic approach described in Section 4.2, the Patient Data

Collection. This stage is composed of the following substages: ⓐ Generation of Combinations and ⓑ Running Combinations Suite, as shown in Figure 4.5. We describe them as follows.

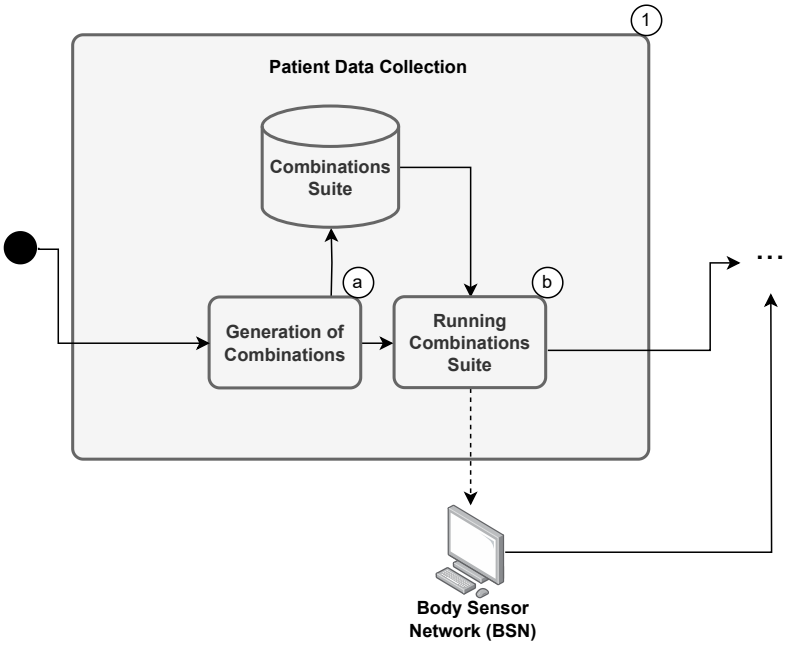


FIGURE 4.5: Patient Data Collection for *ValComb*.

Generation of Combinations. In *ValComb*, the stage ⓐ consists of generating all the possible combinations between sensors' risk levels. If we consider, for instance, the example provided in Section 4.1, for Sensor 1 the possible risk levels are: Low, Medium, and High, while for Sensor 2 they are: Low and High. Table 4.2 depicts the set of combinations for that simple example of BSN. All the combinations are stored in what we call the "Combinations Suite".

TABLE 4.2: Combinations for a simple running example of BSN.

Combination	Sensor 1	Sensor 2
#1	Low	Low
#2	Low	High
#3	Medium	Low
#4	Medium	High
#5	High	Low
#6	High	High

Running Combinations Suite. In *ValComb*, stage ⓑ consists of walking through the Combinations Suite and executing each combination individually on the BSN under

test. This is done by sampling values within the ranges of each risk level in the combination. For example, if we consider Combination #1 in Table 4.2, for Sensor 1 a value within its Low-level range should be randomly sampled. The same applies to Sensor 2. Then, the BSN provides an overall patient risk level (*BSN Outcome*, as explained before) based on the combination of these values for all the sensors.

4.5 TransCov - Sensor Transitions Coverage

In this section, we show how the *TransCov* (Sensor Transitions Coverage) approach implements the stage ① of our generic approach described in Section 4.2, the Patient Data Collection. This stage is composed of the following substages: ⓐ Generation of a Dummy Patient and ⓑ Coverage-based Running of Dummy Patient, as shown in Figure 4.6. We describe them as follows.

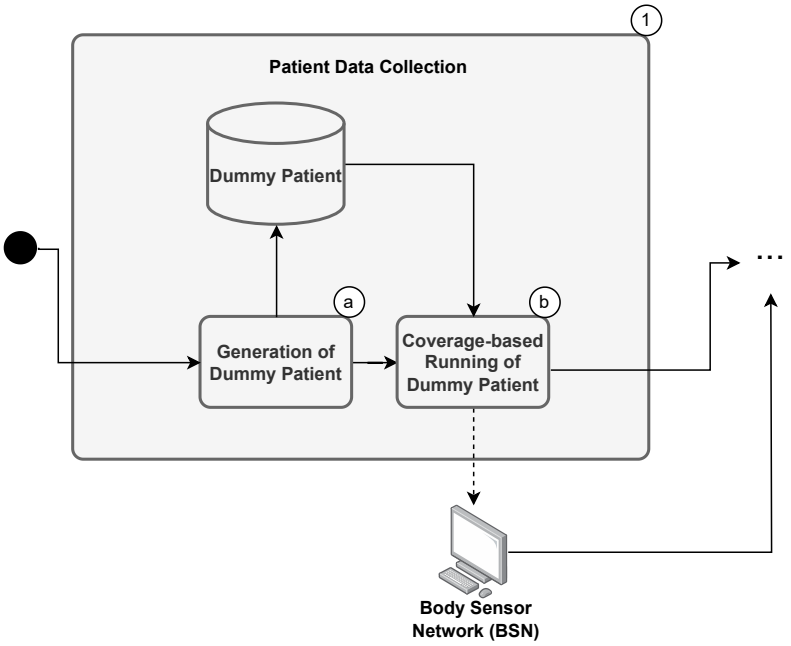


FIGURE 4.6: Patient Data Collection for *TransCov*.

Generation of a Dummy Patient. In *TransCov*, the stage ⓐ comprises the generation of what we call a Dummy Patient. It consists of defining a Testing Patient TP, as presented in Definition 4.7, in which regardless of the number of sensors, all their Transition Matrices TM contain equal probabilities, that is, the probability of going from one state to another is always the same, no matter what is the current state.

If we consider, for instance, the example provided in Section 4.1, for Sensor 1, the transition matrix TM1 contains the same probability values at each of its cells, that is, LL_1=LM_1=LH_1=ML_1=MM_1=MH_1=HL_1=HM_1=HH_1. The same goes for TM2, corresponding to Sensor 2.

Coverage-Based Running of a Dummy Patient. In *ValComb*, stage ⓑ consists of running the Dummy Patient taking into consideration the percentage of transition coverage as a stopping criterion. In other words, the Dummy Patient is run until all the arcs/transitions of the transition matrices TM of all the sensors are covered. The arcs in a transition matrix are randomly covered since the Dummy Patient has equal probability for all of the transitions of its transition matrices, but once one arc is covered, we check this arc as already covered and distribute its probability among the remaining uncovered arcs that share the same origin as this arc. We stop running the patient when all the arcs of the DTMCs of all sensors have been covered, that is, once we achieve 100% of coverage.

4.6 Experiment Setting

To evaluate how effective are the three approaches (instances of *GATE4BSN*) in failure detection we performed some experiments. Specifically, we measure the Passing Test Case Rate (PTCR) and the Execution Time (ET) for the three instances of *GATE4BSN* and the baseline (defined in the next subsection). PTCR measures the percentage of test cases that pass in the SUT. Thus, the smaller the better, that is, the more effective in finding failures. In this case, we apply the function *Comp* (Def. 4.5) that results in *Fail* for *difference* ≥ 2 and *Pass* for *difference* < 2 when comparing the BSN Outcome (Def. 4.2) and the Expected Outcome (Def. 4.3). ET comprises the total time spent to perform one execution of a testing approach. To compute ET, we calculate the time taken for running the approach in the case of the three instances of *GATE4BSN*, or run the baseline for an equivalent amount of time.

In the remainder of this section, we describe the baseline to which we compare the three approaches, and the experimental setup and procedure.

4.6.1 Baseline

To the best of our knowledge, there is no testing approach in the literature for BSNs to which we can compare the three instances of *GATE4BSN*. As discussed later in Chapter 6, we identified some approaches in the literature that are related to our proposal. However, these approaches would need significant adaptation to effectively handle BSN testing. Additionally, most of them do not provide a replication package, making it difficult to extract the source code and perform experiments. For this reason, we propose the random generation of data from a patient as a baseline. To develop the random approach, we make use of a function that generates random numbers following a normal distribution. Therefore, whenever the BSN requests vital sign values from the patient, our function provides random numbers. These random numbers are generated respecting the limits of each sensor (vital sign), as shown in Table 2.1. For example, Oxygen Saturation values range from 0 to 100. Accordingly, we generate values within this range anytime the BSN requests values coming from the Oxi sensor. On the other hand, if the BSN requires data from the Ecg sensor, the random value generated is within the range [0,300].

The simulation of a patient in the random approach runs for the same amount of time as *PASTA*, since PASTA is the approach among the proposed ones that takes more time. Notice that this time is different from the Execution Time (ET) that is explained and computed later in this work. This time is a parameter for the BSN and corresponds to the amount of time for which the simulation of the patient from which we collected data will run. On the other hand, ET comprises the entire time spent by the approach, including, for example, reading and writing operations and delays in the script. As for *PASTA* we run 278 patients, each of them for 30 seconds, we end up simulating patients for 8340 seconds. Thus, we also run Random using the number 8340 as a parameter for the amount of patient simulation time. To guarantee the integrity of the results, we run the Random approach 25 times.

4.6.2 Experimental Setup and Procedure

The experiments have been carried out on a computer with Ubuntu 20.04.6 LTS, 16GB of RAM Memory, and an Intel i7-1165G7 processor. To run the SA-BSN, it is also necessary to install the ROS Noetic[1] for Ubuntu 20.04. To install the SA-BSN, we followed the instructions available here[2]. Because we employ DTMCs and generation of random numbers, each run of *GATE4BSN* exhibits non-deterministic behavior. Therefore, to ensure the validity of the experiments, we performed 25 runs of each instance of *GATE4BSN*. The experimental procedure has been defined according to the guidelines in [82] and it follows the same four stages described in Section 4.2, namely Patient Data Collection (for *PASTA*, *ValComb*, and *TransCov*), BSN Logging, Expected Outcome Computation and Comparison.

Patient Data Collection for *PASTA*. We first define the transition matrices for the SA-BSN. We report the parameters used to the ACTS tool in Table 4.3. Its first column contains the sensor to which the parameter is associated. In the second column, we present the parameter name, which comprises the vital sign name combined with the specific risk level associated with the parameter. The possible risk levels for the SA-BSN, which are High-0, Medium-0, Low, Medium-1, and High-1, are represented here as State0, State1, State2, State3, and State4, respectively. We create a parameter for each combination of vital signs (or sensors) and possible risk levels (or states). Then, the "Parameter Value", in the third column, provides the values that may be combined in the t-way combinations for each specific parameter. Notice that, since the sensors Oxi, Abps, and Abpd present solely 3 risk level ranges, for the parameters related to High-0 and Medium-0, that is, State0 and State1, the only value that they can assume is zero. It means that these states will not be considered during the SA-BSN execution.

Besides the parameters, in the ACTS tool, we may need to define possible constraints. We restrict the parameters according to Equation 4.1 by providing the list of constraints in Equations (4.2) to (4.7).

[1] http://wiki.ros.org/noetic
[2] https://github.com/lesunb/bsn

Table 4.3: Parameters of the SUT in the ACTS tool for *PASTA*.

Sensor	Parameter Name	Parameter Value
Oxygen Saturation (oxi)	oxi_State0	[0]
	oxi_State1	[0]
	oxi_State2	[0, 33, 34, 50, 100]
	oxi_State3	[0, 33, 34, 50, 100]
	oxi_State4	[0, 33, 34, 50, 100]
Heartbeat Rate (Ecg)	hr_State0	[0, 20, 33, 34, 50, 100]
	hr_State1	[0, 20, 33, 34, 50, 100]
	hr_State2	[0, 20, 33, 34, 50, 100]
	hr_State3	[0, 20, 33, 34, 50, 100]
	hr_State4	[0, 20, 33, 34, 50, 100]
Temperature (Term)	temp_State0	[0, 20, 33, 34, 50, 100]
	temp_State1	[0, 20, 33, 34, 50, 100]
	temp_State2	[0, 20, 33, 34, 50, 100]
	temp_State3	[0, 20, 33, 34, 50, 100]
	temp_State4	[0, 20, 33, 34, 50, 100]
Systolic Body Pressure (Abps)	abps_State0	[0]
	abps_State1	[0]
	abps_State2	[0, 33, 34, 50, 100]
	abps_State3	[0, 33, 34, 50, 100]
	abps_State4	[0, 33, 34, 50, 100]
Diastolic Body Pressure (Abpd)	abpd_State0	[0]
	abpd_State1	[0]
	abpd_State2	[0, 33, 34, 50, 100]
	abpd_State3	[0, 33, 34, 50, 100]
	abpd_State4	[0, 33, 34, 50, 100]
Glucose (Glc)	gluc_State0	[0, 20, 33, 34, 50, 100]
	gluc_State1	[0, 20, 33, 34, 50, 100]
	gluc_State2	[0, 20, 33, 34, 50, 100]
	gluc_State3	[0, 20, 33, 34, 50, 100]
	gluc_State4	[0, 20, 33, 34, 50, 100]

$$100 = oxi\_State0 + oxi\_State1 + oxi\_State2 + oxi\_State3 + oxi\_State4 \tag{4.2}$$

$$100 = hr\_State0 + hr\_State1 + hr\_State2 + hr\_State3 + hr\_State4 \tag{4.3}$$

$$100 = temp\_State0 + temp\_State1 + temp\_State2 + temp\_State3 + temp\_State4 \tag{4.4}$$

$$100 = abps\_State0 + abps\_State1 + abps\_State2 + abps\_State3 + abps\_State4 \tag{4.5}$$

$$100 = abpd\_State0 + abpd\_State1 + abpd\_State2 + abpd\_State3 + abpd\_State4 \tag{4.6}$$

$$100 = gluc\_State0 + gluc\_State1 + gluc\_State2 + gluc\_State3 + gluc\_State4 \tag{4.7}$$

Finally, in generating the t-way combinations, we choose t=2 (i.e., pairwise), as detailed in Section 2.2. For the remaining setup in the ACTS tool, we use the default values.

We then run the ACTS tool with the parameters and constraints previously defined. In total, the ACTS generates 278 combinations. An excerpt of the combinations generated is illustrated in Table 4.4. For example, the table indicates that the rows of the transition matrix for Oxi of the first combination ("C.#1") contain the sequence: [0 0 33 33 34][3], which are the values, for this combination, of the parameters: *oxi_State0*, *oxi_State1*, *oxi_State2*, *oxi_State3*, and *oxi_State4*, respectively.

Table 4.4: Combinations generated by ACTS for *PASTA*.

Parameter	**C.#1**	**C.#2**	**C.#3**	**C.#4**
oxi_State0	0	0	0	0
oxi_State1	0	0	0	0
oxi_State2	33	34	50	100
oxi_State3	33	33	50	0
oxi_State4	34	33	0	0
hr_State0	0	0	0	0
hr_State1	0	33	34	50
hr_State2	33	34	0	50
hr_State3	33	0	33	0
hr_State4	34	33	33	0
temp_State0	20	33	34	50
temp_State1	20	33	0	50
temp_State2	20	34	0	0
temp_State3	20	0	33	0
temp_State4	20	0	33	0
abps_State0	0	0	0	0
abps_State1	0	0	0	0
abps_State2	33	34	50	100
abps_State3	33	33	50	0
abps_State4	34	33	0	0
abpd_State0	0	0	0	0
abpd_State1	0	0	0	0
abpd_State2	33	34	50	100
abpd_State3	33	33	50	0
abpd_State4	34	33	0	0
gluc_State0	20	33	34	50
gluc_State1	20	33	0	50
gluc_State2	20	34	0	0
gluc_State3	20	0	33	0
gluc_State4	20	0	33	0

Next, we perform the conversion of transition matrices into a test patient TP. The patient is the input for SA-BSN and is represented by a configuration file with a predefined header as shown in Figure 4.7. It may contain information concerning: whether the

[3]The sequence 33, 33, 34 is an integer approximation for the case of all transitions having the same probability.

sensor values will be shown on the screen or saved in a "log file"; the node scheduling frequency; the vital sign names; and for each DTMC (i.e., sensor) the change frequency.

```
<launch>
<node name="patient" pkg="patient" type="patient" output="screen" />

<param name="frequency" value="10" />

<param name="vitalSigns" value="oxigenation,heart_rate,temperature,abps,abpd,glucose" />

<!-- Change frequency in states of each Markov -->
<param name="oxigenation_Change" value="0.2"/>
<param name="heart_rate_Change" value="0.1"/>
<param name="temperature_Change" value="0.1"/>
<param name="abps_Change" value="0.1"/>
<param name="abpd_Change" value="0.1"/>
<param name="glucose_Change" value="0.1"/>
```

FIGURE 4.7: Header of the configuration file for *PASTA*.

Following the header, we define the transition matrix (TM) and the ranges of risk levels (RL) associated with each of the 5 states in the sensor DTMC. The current state represents the current status of the patient concerning that sensor, that is, if the DTMC for the Oxygen Saturation is in the "LowRisk" state, the patient presents low-risk values according to this specific sensor.

Let us consider the first combination ("C.#1") of Table 4.4 as an example from now on. Each transition matrix defined in the previous stage and generated here is converted into a fragment of the input configuration file. Figure 4.8 shows an excerpt of the configuration file containing the DTMC transition matrix for the Oxygenation sensor and the risk level ranges of each state in the DTMC. Notice that we do not use State0 and State1 since there is only one range for the medium and high-risk levels, as previously shown in Table 2.1. So, we fill with zeros the rows in the transition matrix corresponding to these states. Concerning the risk levels, we set their value to -1. These states are useful for sensors with multiple ranges for the same risk level.

In Figure 4.9 we report the DTMC transition matrix for the Heartbeat Rate sensor and the risk level ranges of each state in the DTMC. Each row of the transition matrix consists of the sequence {0, 0, 33, 33, 34} corresponding to the heartbeat rate parameters of "Combination #1" in Table 4.4. Notice that, for this sensor, we assign values to State0 and State1 since there are two possible ranges for the medium and high-risk levels, as previously shown in Table 2.1.

```
<!-- Markov chain for oxigenation -->
<param name="oxigenation_State0" value="0,0,0,0,0" />
<param name="oxigenation_State1" value="0,0,0,0,0" />
<param name="oxigenation_State2" value="0,0,33,33,34" />
<param name="oxigenation_State3" value="0,0,33,33,34" />
<param name="oxigenation_State4" value="0,0,33,33,34" />

<!-- Risk values for oximeter -->
<param name="oxigenation_HighRisk0" value="-1,-1"/>
<param name="oxigenation_MidRisk0" value="-1,-1"/>
<param name="oxigenation_LowRisk" value="65,100" />
<param name="oxigenation_MidRisk1" value="55,65" />
<param name="oxigenation_HighRisk1" value="0,55" />
```

FIGURE 4.8: Oxygenation setup in the configuration file for *PASTA*.

```
<!-- Markov chain for heart frequency -->
<param name="heart_rate_State0" value="0,0,33,33,34" />
<param name="heart_rate_State1" value="0,0,33,33,34" />
<param name="heart_rate_State2" value="0,0,33,33,34" />
<param name="heart_rate_State3" value="0,0,33,33,34" />
<param name="heart_rate_State4" value="0,0,33,33,34" />

<!-- Risk values for heart frequency -->
<param name="heart_rate_HighRisk0" value="0,70" />
<param name="heart_rate_MidRisk0" value="70,85" />
<param name="heart_rate_LowRisk" value="85,97" />
<param name="heart_rate_MidRisk1" value="97,115" />
<param name="heart_rate_HighRisk1" value="115,300" />
```

FIGURE 4.9: Heartbeat Rate setup in the configuration file for *PASTA*.

We repeat this process for all 6 sensors of the SA-BSN, and the transition matrices corresponding to the same t-way combination are converted into excerpts that together make the configuration file, that is, the test patient. The combinations generated by the ACTS tool become test patients, and this leads to 278 patients.

Each of the generated patient configuration files is used as input for the SA-BSN execution. Besides this file, it is also necessary to inform how long the SA-BSN should run. In our study, we leave each test patient to run for 30 seconds (since this amount of time has proved to be sufficient for the SA-BSN to produce a substantial amount of data).

Patient Data Collection for *ValComb* To generate all the possible combinations of sensor risk levels we make use of the ACTS tool. The parameters we use in the tool are shown in Table 4.5. They may be either 100, meaning that in the combination the sensor is in that state (risk level), or 0 otherwise.

TABLE 4.5: Parameters of the SUT in the ACTS tool for *ValComb*.

Sensor	Parameter Name	Parameter Value
Oxygen Saturation (oxi)	oxi_State0	[0]
	oxi_State1	[0]
	oxi_State2	[0, 100]
	oxi_State3	[0, 100]
	oxi_State4	[0, 100]
Heartbeat Rate (Ecg)	hr_State0	[0, 100]
	hr_State1	[0, 100]
	hr_State2	[0, 100]
	hr_State3	[0, 100]
	hr_State4	[0, 100]
Temperature (Term)	temp_State0	[0, 100]
	temp_State1	[0, 100]
	temp_State2	[0, 100]
	temp_State3	[0, 100]
	temp_State4	[0, 100]
Systolic Body Pressure (Abps)	abps_State0	[0]
	abps_State1	[0]
	abps_State2	[0, 100]
	abps_State3	[0, 100]
	abps_State4	[0, 100]
Diastolic Body Pressure (Abpd)	abpd_State0	[0]
	abpd_State1	[0]
	abpd_State2	[0, 100]
	abpd_State3	[0, 100]
	abpd_State4	[0, 100]
Glucose (Glc)	gluc_State0	[0, 100]
	gluc_State1	[0, 100]
	gluc_State2	[0, 100]
	gluc_State3	[0, 100]
	gluc_State4	[0, 100]

We then run the ACTS tool with the parameters and constraints previously defined, and t=6. In total, the ACTS generates 3375 combinations. An excerpt of the combinations generated is illustrated in Table 4.6. All of these combinations are stored in a file that we call "Combinations Suite" and, individually, they are sent to the running SA-BSN, which randomly samples values within a determined range for each sensor depending on the state (risk level) that contains the value 100.

Patient Data Collection for *TransCov*. The generation of a dummy patient for the Patient Data Collection of *TransCov* consists of giving the same probability for all the transitions of transition matrices representing sensors. In Figure 4.10, we report the DTMC transition matrix for the Heartbeat Rate sensor and the risk level ranges of each state in the DTMC.

We do the same for all the sensors of the SA-BSN and create a dummy patient configuration file. This configuration file is used as input to the SA-BSN execution. The SA-BSN

TABLE 4.6: Combinations generated by ACTS for *ValComb*.

Parameter	C. #1	C. #2	C. #3	C. #4
oxi_State0	0	0	0	0
oxi_State1	0	0	0	0
oxi_State2	0	100	0	100
oxi_State3	0	0	100	0
oxi_State4	100	0	0	0
hr_State0	0	100	0	0
hr_State1	0	0	100	0
hr_State2	100	0	0	0
hr_State3	0	0	0	100
hr_State4	0	0	0	0
temp_State0	0	100	0	0
temp_State1	0	0	100	0
temp_State2	100	0	0	0
temp_State3	0	0	0	100
temp_State4	0	0	0	0
abps_State0	0	0	0	0
abps_State1	0	0	0	0
abps_State2	100	0	0	0
abps_State3	0	100	0	100
abps_State4	0	0	100	0
abpd_State0	0	0	0	0
abpd_State1	0	0	0	0
abpd_State2	100	0	0	100
abpd_State3	0	100	0	0
abpd_State4	0	0	100	0
gluc_State0	0	100	0	0
gluc_State1	0	0	100	0
gluc_State2	0	0	0	100
gluc_State3	100	0	0	0
gluc_State4	0	0	0	0

```
<!-- Markov chain for heart frequency -->
<param name="heart_rate_State0" value="20,20,20,20,20" />
<param name="heart_rate_State1" value="20,20,20,20,20" />
<param name="heart_rate_State2" value="20,20,20,20,20" />
<param name="heart_rate_State3" value="20,20,20,20,20" />
<param name="heart_rate_State4" value="20,20,20,20,20" />

<!-- Risk values for heart frequency -->
<param name="heart_rate_HighRisk0" value="0,70" />
<param name="heart_rate_MidRisk0" value="70,85" />
<param name="heart_rate_LowRisk" value="85,97" />
<param name="heart_rate_MidRisk1" value="97,115" />
<param name="heart_rate_HighRisk1" value="115,300" />
```

FIGURE 4.10: Heartbeat rate setup in the configuration file for *TransCov*.

runs until all the arcs of transition matrices are covered.

Logging the BSN. During the execution of the SA-BSN, we collect and store all the values generated by the sensors (the sensor readings SRs). We also gather the BSN outcome BO, that is, the patient risk level provided by the SA-BSN according to the

sensors' values. The possible outcomes ($Patient_{Risks}$) for the SA-BSN are: "Very Low PR", "Low PR", "Moderate PR", "Critical PR" and "Very Critical PR". Table 4.7 reports some examples of the collected data for *PASTA*. Note that time and patient ID in the table start from zero.

TABLE 4.7: Examples of Test Cases for the SA-BSN and the results using *PASTA*.

	TC #1	**TC #2**	**TC #3**
Patient	0	0	1
Time	0	1	0
Oxi	41.8212 (High risk)	72.5934 (Low risk)	7.03972 (High risk)
Ecg	99.2647 (Moderate risk)	97.9146 (Moderate risk)	95.6635 (Low risk)
Term	37.3749(Low risk)	37.3749 (Low risk)	38.1103 (Moderate risk)
Abps	23.6058 (Low risk)	66.6665 (Low risk)	118.236 (Low risk)
Abpd	25.6534 (Low risk)	18.6827(Low risk)	95.0196 (High risk)
Glc	86.6656 (Low risk)	90.227 (Low risk)	65.6257 (Low risk)
Expected Outcome (Risk)	Critical PR	Low PR	Very Critical PR
BSN Outcome (Risk)	Critical PR	Very Low PR	Moderate PR
Difference	0	1	2
Passed?	**Yes**	**Yes**	**No**

Expected Outcome Computation and Comparison. We also compute the oracle, that is, the expected outcome $E\_BO(p, t, SR_t^p)$, for each of the sensor readings SR_t^p collected. The oracle, in this work, has been defined as Table 4.8 describes.

TABLE 4.8: Oracle definition.

Risk Level	**Definition**
Very Low Risk	All the sensors are "Low Risk".
Low Risk	One sensor is "Moderate Risk" and the remaining are "Low Risk".
Moderate Risk	At least two sensors are "Medium Risk" and no sensor is "High Risk".
Critical Risk	Exactly one sensor is "High Risk".
Very Critical Risk	More than one sensor is "High Risk".

Each sensor reading SR and its corresponding expected outcome E_BO make a test case TC. Note that to compute the oracle, instead of using the SR purely, we convert the values into risk levels (i.e., low, moderate, or high risk) based on the ranges considered by the SA-BSN, as previously shown in Table 2.1. Then, we count the number of sensors at each risk possible level. We opted for this conversion because it made the oracle simpler

to understand and better aligned with the way the SA-BSN functions. However, this approach can be easily adapted for other BSNs that use different sensor types, ranges, or even no specific ranges at all. The 10th row of Table 4.7 reports some examples of expected outcomes for sensor readings collected from the SA-BSN. In the 11th row, we provide the BSN outcomes BO. Then, as shown in the following row of the table, we present the result of the function $Comp$ that computes the absolute difference between the BSN outcome and the expected outcome corresponding numbers, and decides if the TC passed or not. We recall, as stated in Def. 4.5, that a TC passes if the difference is lower than 2, otherwise, it fails.

The complete collection of data we gathered in each stage of *GATE4BSN* can be found on Github[4].

4.7 Experimental Results

This section encompasses the experimental findings (Section 4.7.1) and the statistical analysis we performed (Section 4.7.2). The experiments measured the Passing Test Case Rate (PCTR) and the Execution Time (ET) of our three proposed approaches, showing that in comparison to the random baseline, they provide an effective means for testing Body Sensor Networks. Specifically:

- *PASTA* resulted in an average of $92.8\% \pm 0.59$ of PTCR and presented an ET on average of 22014.44 seconds.
- *ValComb* resulted in an average of $95.47\% \pm 0.32$ of PTCR and presented an ET on average of 288.88 seconds.
- *TransCov* resulted in an average of $94.96\% \pm 5.84$ of PTCR and presented an ET on average of 339.24 seconds.
- The Random approach (Baseline) resulted in an average of $98.18\% \pm 0.03$ of PTCR and presented an ET on average of 8389.16 seconds.

[4]https://github.com/samirasilva/Paper_JSS

The complete experiment results are presented in Table 4.9. To analyse the data, we followed the guidelines in [83, 84]. For reproducibility, the R scripts, and datasets are made available[4].

TABLE 4.9: Passing Test Case Rate (PTCR) and Execution Time (ET) for the three instances of *GATE4BSN* and the Random Approach (Baseline).

	PASTA		**ValComb**		**TransCov**		**Random**	
	PTCR(%)	ET(s)	PTCR(%)	ET(s)	PTCR(%)	ET(s)	PTCR(%)	ET(s)
Ex1	93.59	22010	95.41	253	100.00	341	98.15	8390
Ex2	93.10	22013	95.38	342	79.67	336	98.18	8389
Ex3	93.34	22009	95.67	281	98.77	337	98.15	8389
Ex4	93.29	22015	95.55	226	99.19	338	98.19	8389
Ex5	93.36	22011	94.72	292	97.50	335	98.15	8389
Ex6	93.02	22012	95.67	278	98.56	332	98.16	8389
Ex7	93.23	22017	96.24	313	84.12	348	98.19	8389
Ex8	93.59	22012	95.88	245	97.33	330	98.19	8389
Ex9	93.79	22008	95.70	335	87.89	347	98.14	8389
Ex10	93.03	22013	95.41	155	100.00	340	98.25	8389
Ex11	93.24	22019	95.32	312	99.11	327	98.20	8389
Ex12	93.22	22018	95.49	324	95.75	335	98.15	8389
Ex13	92.16	22011	95.20	297	99.61	340	98.16	8389
Ex14	92.18	22022	95.49	260	98.85	358	98.23	8389
Ex15	92.45	22021	95.20	301	96.86	343	98.13	8389
Ex16	91.44	22010	95.47	313	94.88	327	98.16	8389
Ex17	92.22	22025	95.11	381	99.13	337	98.20	8390
Ex18	92.71	22026	95.79	155	83.70	353	98.21	8389
Ex19	92.47	22016	95.08	260	97.33	335	98.19	8389
Ex20	92.16	22016	95.82	307	90.41	338	98.20	8389
Ex21	92.24	21999	95.14	361	95.83	350	98.22	8389
Ex22	92.54	22011	95.64	283	88.00	355	98.18	8390
Ex23	92.59	22013	95.32	284	93.79	335	98.18	8390
Ex24	92.86	22018	95.70	325	98.69	327	98.20	8389
Ex25	92.12	22016	95.32	339	99.11	337	98.16	8389
Avg	**92.80**	**22014.44**	**95.47**	**288.88**	**94.96**	**339.24**	**98.18**	**8389.16**
Std	**0.59**	**5.80**	**0.32**	**54.34**	**5.84**	**8.54**	**0.03**	**0.37**

4.7.1 Empirical Data

The data for analysis is collected by logging sensor data, the BSN outcome, and timestamps in the format of execution traces, as well as computing the expected outcome. For the three approaches and the baseline, the difference between the BSN outcome and the expected outcome is calculated for each test case. Table 4.7 shows some examples of test cases for SA-BSN, the expected outcome, the outcome obtained using, for example, *PASTA*, and the calculated difference. Note that, given the different methods applied to derive the test cases, the number of test executions performed for the three proposed approaches differ widely. For *PASTA*, each of the 25 experiments consisted of

278 patients resulting in approximately 88227.52 test cases in each experiment. The experiments for the baseline were not guided by patients, but by execution time intending to pair with *PASTA*, which is the approach that takes more time, for a fair comparison, resulting in 25 samples of 495888.12 test cases on average. *ValComb* presents 1 test case for each possible combination of sensor risk labels. For SA-BSN, three sensors have 3 possible risk levels and three sensors have 5 possible risk levels. Thus, the total number of combinations of risk labels is 3375 which is the number of test cases in *ValComb*. Finally, *TransCov* runs until the complete transition coverage is achieved. This approach presents an average of 4234.2 test cases per experiment.

For the approaches that employ DTMCs to represent sensors, i.e., *PASTA* and *TransCov*, we also compute the percentage of DTMC's arcs that have been covered. Table 4.10 reports the percentage of coverage for these approaches considering each sensor and the average coverage. By calculating coverage for *PASTA*, we were able to observe that the arcs/transitions of sensor DTMCs are being little explored. This fact served as inspiration for proposing an approach that guaranteed full transition exploration, i.e., the *TransCov* approach.

TABLE 4.10: Coverage of DTMC's transitions for each sensor and the average coverage for *PASTA* and *TransCov*.

Approach	**Oxi**	**Hr**	**Temp**	**Abps**	**Abpd**	**Gluc**	**Average**	**Std**
PASTA	0.91	0.63	0.65	0.85	0.86	0.66	0.76	0.13
TransCov	1	1	1	1	1	1	1	0.00

4.7.2 Data Analysis and Discussion

To choose an approach to statistical analysis, we carefully examine the properties of the dataset [83]. After ascertaining that our data does not follow a normal distribution, we apply the Kruskal-Wallis test [85] to assess at a significance level of 5% the null hypothesis that the differences in PTCR for the different approaches are not statistically significant. The Kruskal-Wallis test is a non-parametric test used to determine whether there are statistically significant differences between three or more independent groups when the assumptions of normality and homogeneity of variances are violated. When considering the baseline and the proposed approaches, the resulting p-value for the test was 1.491e-11, with H=53.42 and freedom degree equal to 3, meaning that we can reject

the null hypothesis that no significant difference in effectiveness exists, at least at the 95% confidence level.

A significant Kruskal-Wallis test indicates that at least one testing approach stochastically dominates another one, but does not identify the dominance relationship among pairs of techniques. To determine which testing approaches are different, we performed pairwise comparisons with the Vargha-Delaney effect size (A_{12} measure) [86] after the Kruskal-Wallis test. The results are shown in Table 4.11. If the value in cell (i,j) is lower than 0.5, it means that the approach i dominates the approach j. Contrariwise, if the value in cell (i,j) is greater than 0.5, the approach j dominates the approach i. Finally, if the value in the cell (i,j) is exactly 0.5, this means that there exists no difference between the scores of the two approaches. By looking at the data in the table, we can conclude that the order of domination is: *PASTA*>>*ValComb*>>*TransCov*>>Random, where the symbol >> means "dominates".

TABLE 4.11: Vargha-Delaney effect size (A_{12} measure) for the baseline and instances of *GATE4BSN*.

	PASTA	**ValComb**	**TransCov**	**Random**
PASTA	-	0	0.2408	0
ValComb	1	-	0.328	0
TransCov	0.7592	0.672	-	0.44
Random	1	1	0.56	-

According to the statistical analysis, *PASTA* is the most effective approach in terms of the capability to detect failures, as measured by PTCR. However, among the instances of *GATE4BSN*, according to the ET analysis shown in Table 4.9, it is also the approach that takes the longest execution time. Therefore, *PASTA* should be used if effectiveness is the most important variable when testing a BSN. Also considering statistical analysis, the second most effective approach is *ValComb*, which is still the fastest one. Therefore, in cases where a quick approach to testing BSNs is desired, *ValComb* is a good option, because it achieves a good balance between PCTR and ET. Statistically speaking, *TransCov* is in third place in terms of effectiveness, being a very unstable approach, which can be confirmed if we observe the PTCR value throughout the 25 experiment repetitions. In our interpretation, this might be due to the fact that in addition to the generation of random numbers when the DTMC is in a given state, the transition coverage is also carried out completely randomly. Therefore, as this is the approach with more non-deterministic elements, there is a large variation in PTCR compared to other

approaches. Finally, the random generation of values coming from the sensors in the Baseline is the least effective approach.

As the effectiveness values, i.e., PTCR, of *PASTA* and *ValComb*, even though the former dominates statistically the latter, are anyhow not very distant, given their respective execution time values ET are very different, with ($ET_{PASTA} > ET_{ValComb}$), we tried to investigate whether the failures raised by *PASTA* and *ValComb* could be associated to the same faults or descend from different causes. In fact, if by running *PASTA* for a longer time we continue to detect again and again failures that are due to the same fault, then the additional effort would not be worthwhile. However, due to the huge amount of collected data and the complexity of the system under test, we have not (yet) been able to localize the faults that are responsible for the observed failures, neither so the team of SA-BSN developers to whom we reported the failures. Therefore, we decided to focus on the "symptoms" and tried to cluster the cumulative set of failing test cases from both approaches to verify whether the two approaches identified the same or different clusters of failures. Our intuition was that if we were able to observe some clusters of failing test cases belonging to only one approach, then we could hypothesize that these are symptoms of possible faults that the other approach did not reach.

With this objective, we used a clustering algorithm well known in the literature, K-means [87]. However, to use K-means it is necessary to define the value of K, that is, the number of clusters. To choose a good value for K, we use two approaches, the Silhouette Score [88] and the Davies-Bouldin Score [89]. The silhouette score is a measure used to evaluate the quality of clusters created by a clustering algorithm. It measures how well-separated the clusters are, with scores ranging from -1 to 1. A score closer to 1 indicates that the data points are well-clustered and far away from neighboring clusters, while a score closer to -1 indicates that data points have been assigned to the wrong clusters. A score around 0 indicates overlapping clusters. It is commonly used in clustering analysis to determine the optimal number of clusters. The silhouette score threshold is commonly set at 0.5. A score of more than 0.5 indicates a high-quality cluster, whereas a score of less than 0.5 indicates a low-quality cluster. After systematically computing the Silhouette Score with increasing values of K, we chose K=360 since it is the smallest value of K that produces a Silhouette Score greater than 0.5, which is equal to ~ 0.51 and indicates high-quality clustering.

To confirm that K=360 is a good choice, we also compute the Davies-Bouldin Score. The value of the Davies-Bouldin Score varies from 0 to infinity, the lower the value, the better the quality of the clustering. An ideal value for the Davies-Bouldin Score is between 0 and 1, where lower values indicate better separation of clusters. In our analysis, the value of the Davies-Bouldin Score is equal to ~ 0.78 for K=360, which confirms that 360 is a good choice.

Considering K=360, we run K-means with all non-passing test cases from *PASTA* and *ValComb*. We found that there are many clusters containing only test cases from *PASTA* and a few containing test cases only from *ValComb*. Both approaches also shared some clusters. Then, we investigated the clusters containing only *PASTA* test cases. Since we clustered the raw sensor values of test cases, we then converted them into risk labels to make the analysis of the clusters easier. In this way, we could check that the clusters are meaningful since they do not present much discrepancy among the combinations of risk level labels belonging to the same cluster. Finally, we noticed that there are some test cases belonging only to PASTA clusters that correspond to risk level combinations that fail in PASTA but pass in *ValComb*. This may be explained by the fact that *ValComb* contains exactly one test case instance for each possible risk level combination, while in *PASTA*, even if we do not have the guarantee of exploring all possible combinations, for the same combination we may have more than one example. Table 4.12 provides an excerpt of the test case suites of *PASTA* and *ValComb*. As we can notice, even if combination #1 is present in both test case suites, in *ValComb* it passes while in *PASTA* it fails. The same applies to Combination #2.

Hence we can conclude that - when time permits - it can be useful to continue testing for a longer time employing *PASTA* because this might reveal issues that *ValComb* may skip. This is even more important for safety-critical applications, as SA-BSN is. On the other hand, under tight time constraints, *ValComb* already provides good effectiveness.

The three instances of *GATE4BSN* explore different and coherent characteristics of input. *PASTA* explores patients with 100% or 0% probability of presenting a determined sensor risk level and patients with equal probability of being at any sensor risk level. On the other hand, *ValComb* explores all the possible combinations of sensor risk values in a very short time. Finally, *TransCov* covers the transitions from one sensor risk level to

TABLE 4.12: Examples of risk level combinations that fail for *PASTA* and pass for *ValComb*.

	Combination #1		Combination #2	
	PASTA	**ValComb**	**PASTA**	**ValComb**
Experiment	#1	#1	#1	#1
Patient	233	0	67	0
Time	71421	129	20539	488
Oxi	low risk (69.2052)	low risk (73.714222)	high risk (7.74788)	high risk (42.542467)
Ecg	high risk (137.158)	high risk (282.742014)	moderate risk (70.8799)	moderate risk (104.522312)
Term	high risk (31.8519)	high risk (16.608634)	low risk (37.3749)	low risk (37.721862)
Abps	high risk (159.522)	high risk (202.050148)	low risk (96.6736)	low risk (3.913506)
Abpd	low risk (76.5002)	low risk (67.233322)	high risk (127.512)	high risk (278.977966)
Glc	moderate risk (46.5979)	moderate risk (44.084136)	moderate risk (52.7125)	moderate risk (54.968508)
Expected Outcome	Moderate PR	Critical PR	Moderate PR	Critical PR
BSN Outcome	Very Critical PR	Very Critical PR	Very Critical PR	Very Critical PR
Difference	2	1	2	1
Passed?	**No**	**Yes**	**No**	**Yes**

another. The Random choice of number does not guarantee that we are covering any category of patient and, for this reason, might fail more frequently. .

4.8 Threats to Validity

Notwithstanding our greatest efforts, there are potential threats to the validity of the presented results. We take into consideration four aspects following the classification in [90].

Construct validity: whether the experiments we design are suitable to assess the effectiveness of the three proposed instances of *GATE4BSN*'s, particularly in comparison to Random. Our validation shows that *GATE4BSN* identified unknown failures in SA-BSN and has better effectiveness than Random. We selected two metrics (PTCR and ET), which are widely used to assess effectiveness, but we cannot exclude that other metrics might have led to different results. Moreover, PTCR may end up detecting only test cases that fail due to the same bug. Also, we compared *GATE4BSN* only with the Random baseline. This limitation stems from the scarcity of testing approaches tailored for BSNs. Another possible threat to validity is the fact that in the approaches that make

use of DTMCs, that is, *PASTA* and *ValComb*, the initial state is always the same, the one representing the low risk level. However, this comes from the original implementation of SA-BSN.

Internal validity: whether, in reality, the "treatment" is what caused the observed results rather than other factors. We followed guidelines for performing experiments and the statistical analysis. A frequent internal threat involves the precision of measures influenced by random elements. To address this, we replicated the experiments 25 times, aiming to mitigate this potential threat.

External validity: whether and how far the observations lend themselves to generalization. *GATE4BSN* is general, even though, as we explained in the introduction, for the sake of exposition, we consider a BSN in the healthcare domain. However, so far our results only rely on the SA-BSN case study, and more experiments are needed to better support our conclusions on *GATE4BSN* effectiveness, and also to ascertain if we can use it in other BSN application domains.

Reliability: whether and how much other researchers can replicate the observations. To guarantee reproducibility, we make available all data and settings information.

Chapter 5

AdapTA - Adaptive Testing Approach

In this section, we introduce the Adaptive Testing Approach, or *AdapTA* for short. For better clarity, we first describe the basic Testing Approach without adaptation, and then we present the components of the Adaptation logic. For illustration, we refer to Figure 5.1 in which the stages of *AdapTA* are connected through either dashed arrows that indicate the flow of stages or commands (for commands, we use specific guards), or continuous arrows that indicate the transmission of data. We also present the three adaptation scenarios handled in *AdapTA*, and the baselines we propose for each of them. Finally, we describe the experiment setting, results, and the threats to the validity of the experiments.

5.1 Testing Approach

In this section, we describe the seven basic components of the testing approach. The sole use of them, without the adaptation components, makes up the non-adaptive version of *AdapTA*, which will later serve as a baseline.

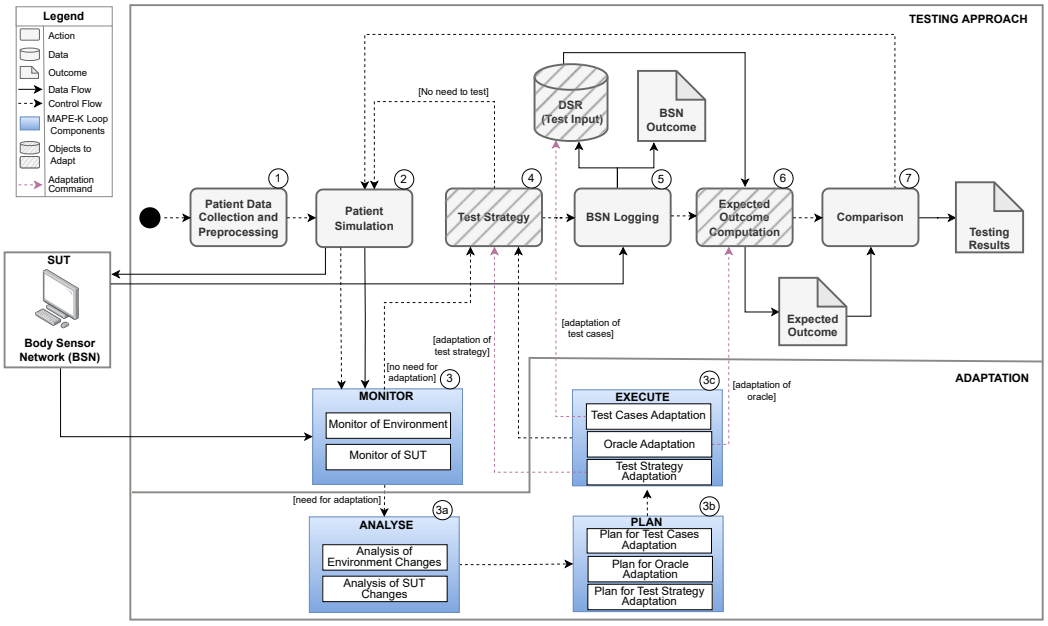


FIGURE 5.1: Adaptive Testing Approach (AdapTA).

5.1.1 Patient Data Collection and Preprocessing

With reference to Figure 5.1, Stage ① consists of collecting and preprocessing vital signs data from patients that will be later used to generate the test inputs to the BSN. Note that this is a stage that we do only once, as shown in the figure (i.e., it is outside of the iteration). As said, we here perform ex-vivo testing [91], which means that we get actual data from the field (*FSR*s), but use this data to generate DTMCs representing a patient, and later run the DTMCs to generate *DSR*s and perform test sessions in-house. As an aside comment, online testing would correspond to directly launching the test cases while the BSN is in use and monitors a patient. However, due to the safety-critical nature of BSNs, it would be of course risky to conduct testing directly online.

Field data could be either collected directly from the patients and then promptly processed for testing in the subsequent steps, or otherwise they could be collected over some period of time, and saved into a database for later processing. This work applies the second case (data are taken from a database), even though either case would work for *AdapTA*.

An advantage of deferring the testing sessions after collecting a bulk of data into a database is that we can merge data coming from different patients so that we can obtain more general *patient profiles*. A profile corresponds to a category of patients yielding

similar physical and health-related characteristics (e.g., age, gender, height, weight, or some monitored pathology). We can thus test the behavior of the BSN over different patient profiles; in traditional functional testing, this roughly corresponds to testing over different equivalence classes (e.g., [92], Ch.6). Therefore, we could test for instance the behavior of the BSN when monitoring a young healthy athlete or when instead monitoring an elderly patient with critical pathologies.

Patient profiles are modeled through DTMCs. Precisely, for a considered patient profile, a DTMC is constructed for each vital signal (i.e., each sensor of the BSN). The DTMCs are designed to represent the possible states of a sensor (defined by the selected values) and the transition probabilities which reflect the likelihood of changes in vital signals over time. For example, the DTMC for body temperature would model the transitions from a low-risk state (e.g., [36°C, 38°C]) to a medium-risk state (e.g., [38°C, 41°C]), among others. This approach allows for capturing variations across differing profiles, such as the higher probability of an elderly patient progressing to a high-risk scenario compared to a youthful patient, or the increased risk associated with obesity compared to normal weight.

5.1.2 Patient Simulation

Stage ②, Patient Simulation, consists of, for each patient profile, running the DTMCs of its sensors to simulate a patient. In *AdapTA*, a patient is simulated during a certain amount of time m, and the data produced, *DSR*, is sent as input to the BSN and the Monitor component.

5.1.3 Monitor

The monitoring component, in Stage ③, must efficiently collect and check data from both the BSN and Patient Simulation (environment). The Monitor is also a component of the MAPE-K loop, as explained later in Section 5.2. In case it observes changes in the SUT (the BSN) or in the environment (the Patient), we proceed to the next stage of the loop (Analyse - Stage ③a). Otherwise, if no adaptation is needed, it directs the flow to Test Strategy (Stage ④). For the baseline, the latter is always chosen, as explained later in this work.

5.1.4 Test Strategy

The Test Strategy (Stage ④ in Figure 5.1) decides when to test based on some condition. If this condition is satisfied, we move to the next stage, BSN Logging ⑤, and start testing. Otherwise, we go back to Stage ② and continue simulating the patient. Testing may be triggered periodically, that is, testing is launched with a determined frequency and duration, or triggered by some event (e.g., SUT adaptation). We then decided to use a periodical trigger as a default strategy which means that we test for t seconds every w seconds. Thus, the condition to be satisfied here to test is being within the testing interval.

5.1.5 BSN Logging

Stage ⑤ involves recording data from the BSN during its execution. This process allows the data used by the BSN, i.e., DTMC sensor readings (*DSR*), and its corresponding output provided by the BSN, to be stored for later use in testing. The BSN outcome may consist, for example, of the overall risk level of a patient according to the BSN at a determined instant of time.

5.1.6 Expected Outcome Computation

In Stage ⑥, the Expected Outcome, referred to as the Oracle, is derived from the DTMC sensor readings (*DSR*s). Specifically, if the BSN outcome is, for example, the overall risk level of a patient at a given moment, this stage comprises predicting it based on the data provided in the *DSR*. The pair of *DSR* and its Expected Outcome make up a Test Case.

5.1.7 Comparison

Finally, in Stage ⑦, the BSN Outcome is compared with the Expected Outcome (i.e., the Oracle) for a specific *DSR*. In this stage, a comparison function that computes the distance between the BSN and Expected outcomes may be defined. By comparing them, we are able to know whether a determined Test Case passes or not. *AdapTA* runs for a predefined duration, denoted as m, which corresponds to the duration of the patient simulation, as previously anticipated.

5.2 Adaptation

The adaptation logic of *AdapTA*, stages ③a, ③b, and ③c, is inspired by the reference architecture for SATF approaches defined in Section 3.4. Thus, our adaptation logic also uses the MAPE-K loop, the most influential reference control model for autonomic and self-adaptive systems [93]. The Knowledge components are the *DSR*, and the BSN and Expected Outcomes. The remaining components are described as follows. As will be described in more detail in the following, the entire MAPE-K loop is executed only when the monitor identifies the need for adaptation. Besides, even if the Monitor is part of the MAPE-K loop, we here make it part of the testing approach since it is needed even if the adaptation logic is removed from *AdapTA*.

5.2.1 Analyse

The Analyse component is responsible for deciding which adaptation should be planned based on the monitored data. This data may be environmental (patient) or SUT (BSN) changes. The Analyse component then informs the Plan component about its decision.

5.2.2 Plan

Knowing which adaptations should be made, the Plan stage formulates a plan with appropriate and concrete actions that make up these adaptations. For *AdapTA*, the possible adaptation mechanisms are Test Cases Adaptation, Oracle Adaptation, and Test Strategy Adaptation.

5.2.3 Execute

The Execute stage consists of performing the adaptation activities described in the plan. For *AdapTA*, we consider three possible objects to adapt: Test Strategy, Test Cases (more specifically, the *DSR*s), and the Oracle (i.e., the function that computes the expected outcome). The Execute component launches an adaptation function to change these objects following the plan. This adaptation function may change *DSR* values or replace the Testing Strategy or the Expected Outcome Computation functions with a new function according to what has been planned.

5.3 Scenarios

In this work, we propose three adaptation scenarios to illustrate the use of *AdapTA*. Table 5.1 summarizes them by providing the adaptation trigger, a more concrete description for it, the adaptation policy, and the object to adapt. We describe them in detail as follows.

Table 5.1: Adaptation scenarios for AdapTA.

Scenario	Adaptation Trigger	Trigger Description	Adaptation Policy	Object to Adapt
S1	Change in the SUT	Activation/Deactivation of sensors	Adjust the test cases	Test Cases
S2	Change in the environment	Change in the patient profile	Adjust the oracle	Oracle
S3	Change in the environment	Patient is in very critical condition	Adjust the frequency of testing	Test Strategy

5.3.1 Scenario 1 (S1) - Activation/Deactivation of Sensors

- **Description:** Let us suppose that the battery for a determined sensor is too low. Then, the BSN adapts itself by turning off this sensor. If we are testing this system, the test cases from this moment on should not include this sensor anymore. Also, the oracle, when computing the expected outcome, should not consider it. The opposite applies when a sensor gets activated.
- **Adaptation Policy:** At runtime, if a sensor is deactivated, the test cases to be exercised should not consider it. Also, the oracle should not use it to compute the expected outcome.

5.3.2 Scenario 2 (S2)- Different Patient Profiles

- **Description:** Suppose that the sensors have been detached from a determined patient and attached to a different one with different health conditions (i.e., patient profile). The oracle should adapt to handle this change and consider the characteristics of the new patient to compute the expected outcome.

- **Adaptation Policy:** If sensors are attached to a new patient, the oracle should take into consideration their health conditions (i.e., patient profile) to determine the expected outcomes for their sensor readings.

5.3.3 Scenario 3 (S3) - Patient in Critical Condition

- **Description:** Testing the BSN when the patient is in a very critical condition is needed first for patient safety reasons since the BSN must work properly in this scenario. Also, this is a scenario in which an alarm is launched, and a doctor/nurse should be called. Therefore, we do not want to require human resources, which are normally something very scarce in hospitals (and have a high cost) unnecessarily. This adaptation scenario allows an on-demand test strategy and it is useful when resources are limited.

- **Adaptation Policy:** Whenever at least two of the sensors monitoring a patient report "high risk", testing is triggered.

5.4 Experiment Setting

In this section, we discuss the setting of the experiments we conduct to evaluate the effectiveness of *AdapTA* in failure detection across the three proposed scenarios. Specifically, we measure the Passing Test Case Rate (PTCR) for the three scenarios of *AdapTA* and their corresponding baselines. PTCR consists of the percentage of test cases that pass in the SUT. Therefore, the lower the PTCR, the better, indicating greater effectiveness in detecting failures. The experimental procedure was designed following the guidelines outlined in [82] and adheres to the seven stages detailed in Sections 5.1 and 5.2 .

In the remainder of this section, we describe the MIMIC II Clinical Database, the database containing data from the field employed, the implementation of the Testing Approach components, and the proposed scenarios. Details about the implementation of the adaptation components are embedded in Section 5.4.3. Finally, we also describe the baseline for each scenario, the experiment settings and results, and the threats to the validity of the experiments.

5.4.1 MIMIC II Clinical Database

In this work, we use the Intensive Care Unit (ICU) data used for the cardiology challenge extracted from version 2.6 of the MIMIC II Clinical Database [94]. The dataset consists of records collected between 2001 and 2008 from a variety of ICUs (medical, surgical, coronary care, and neonatal) in a teaching hospital. From the whole dataset, we used the subset of data employed in the cardiology challenge on predicting mortality of ICU patients from PhysioNet [95, 96]. The data regards 12,000 subjects selected at random whose age at ICU admission was 16 years or over, and whose initial ICU stay was at least 48 hours long. Up to 41 variables were recorded at least once during the first 48 hours after admission to the ICU. Not all variables were available in all records. The records consist of admission records of the patients (i.e., age, gender, height, ICU type, and initial weight), and other 36 measurement signals (a.k.a. time series), from which, we selected and used as our field data: arterial blood pressure (systolic and diastolic), heart rate, oxygen saturation, and temperature.

5.4.2 Testing Approach

In this section, we define the experiment setting for the components of the Testing Approach in *AdapTA*.

5.4.2.1 Patient Data Collection and Preprocessing

To derive the patient profiles corresponding to the equivalence classes on which we test the SA-BSN, we first selected the attributes according to which the *FSR*s could be grouped. Based on our study of the entries recorded in the MIMIC II database, we selected two attributes relative to the patients, namely, Age and Body Mass Index (BMI). In the literature, the categorization of patient profiles by age is not defined by a unique standard but by different standards depending on the application. For the sake of this study, we defined our own categorization as follows: Youth (16y-29y), Adult (30y-59y), and Elderly (60y+). The BMI is defined by:

$$BMI = \frac{weight(kg)}{height(m)^2} \tag{5.1}$$

It yields a well-known categorization that was first proposed in [97], i.e.,: Underweight (0-18.4), Normal weight (18.5-24.9), Overweight (25-29.9), Obesity 1 (30-34.9), Obesity 2 (35-39.9), and Obesity 3 (40+). In addition, we use a third categorization according to the ICU attributes of the records, as clinical factors influencing the patients' vital signals. The ICU categories, as provided in the database, are: Cardiac Surgery Unit, Coronary Care Unit, Medical ICU, and Surgical ICU.

After categorizing the field data according to the patient profiles, we generate a DTMC model for each sensor belonging to a patient profile. The DTMC states are the ranges that represent the risk levels for the sensor considered. The transitions in the DTMCs correspond to the probability of changing from one risk level to another. The transformation of the field data into DTMCs is done by: (1) using as states the same risk level ranges employed by our SUT, the SA-BSN (as defined in [98]), and (2) assigning transition probabilities in the DTMCs based on the frequency with which these transitions occur in the field. The likelihood of a signal moving from one state to another may vary depending on the considered profile. For example, for the Temperature sensor, the transitions going to the low-risk state, which is fixed to [36°C, 38°C], may present a high probability in the DTMC for healthy patients.

It is important to highlight that, in general, it is challenging to attribute semantics to the states as we do in this explanation. For example, knowing that [38.1°C, 41°C] represents a medium risk to elderly patients requires domain expertise.

The transition probabilities must be computed for each sensor of each patient profile. To calculate these probabilities, for a determined sensor and patient profile, we compute a weight that is the number of transitions from one state to the others in the field data. Then, we normalize the weights of the transitions in each DTMC by dividing each weight by the maximum weight.

5.4.2.2 Patient Simulation

The patient simulation consists in running the DTMCs for a patient profile for m seconds. Due to limited resources and time constraints, we restrict m to 3600 seconds. The DTMCs of a patient profile execute to produce input for the BSN, the DTMC Sensor Readings (*DSR*s).

5.4.2.3 Monitor

The monitor is responsible for monitoring which is the current patient profile, the *DSR*s coming from the Patient Simulation, and the SA-BSN. Then, it decides whether there is a need for adaptation based on these observed objects. If the current patient profile or the patient conditions are critical, it activates the *analysis* of environmental changes. On the other hand, if it observes that the SA-BSN is adapting to deactivate/activate sensors, it activates the *analysis* of SUT changes is performed. Otherwise, if there is no need for adaptation, it directs the flow to Test Strategy.

5.4.2.4 Test Strategy

Test Strategy is responsible for determining when it is time to test. As a default strategy, testing runs for $t = 60$ seconds, pauses for $w = 300$ seconds, and then restarts in a continuous cycle. Therefore, when reaching the Test Strategy stage, if it is testing time, we proceed to the next stage (BSN Logging). Otherwise, if it is break time, we return to the Patient Simulation stage.

5.4.2.5 BSN Logging

During a testing session, we collect and store all *DSR*s that were input into the SA-BSN. Additionally, we capture the BSN Outcome, representing the patient's risk level determined by the SA-BSN based on the *DSR* values. For the SA-BSN, the possible patient's risk levels are: "Very Low Risk," "Low Risk", "Moderate Risk", "Critical Risk", and "Very Critical Risk".

5.4.2.6 Expected Outcome Computation

The Expected Outcome Computation consists in determining the Expected Outcome (oracle) for each *DSR* logged. As a default oracle for testing the SA-BSN, we use the same oracle as *GATE4BSN*, as defined in Section 4.6.2, more specifically, in Table 4.8.

Note that, to determine the oracle, we convert *DSR* values into their corresponding risk levels (i.e., low, medium, or high) based on the thresholds defined by the SA-BSN, as

outlined in Table 2.1. Then, we tally the number of sensors at each potential risk level and apply the rules defined in Table 4.8. We chose to do this conversion because it simplifies the oracle and aligns more closely with the operation of the SA-BSN. However, *AdapTA* is flexible and its oracle may be easily adjusted for other BSNs that use different sensor types or ranges, or lack specific ranges altogether. We recall that each *DSR* and its corresponding Expected Outcome make a Test Case.

5.4.2.7 Comparison

The comparison consists of contrasting the BSN Outcome and the Expected Outcome (oracle) for a determined test case in order to decide if it passes or fails. This comparison is made by assigning a numerical ID for each of the possible Patient Risk Levels as shown in Table 4.8 and computing the absolute difference between them. A test case passes if the absolute difference between the IDs of the Expected and the BSN Outcomes is lower than 2 and fails otherwise.

5.4.3 Scenarios

This section outlines the implementation of the three adaptation scenarios introduced earlier in Section 5.3.

5.4.3.1 Scenario S1 - Activation/Deactivation of Sensors

One of the adaptations performed by the SA-BSN is the dynamic activation or deactivation of sensors based on their battery level. A deactivated sensor switches to 'charge' mode and reactivates automatically once it reaches a determined battery level.

In the SA-BSN, whenever a sensor is deactivated, subsequent sensor readings retain the last recorded value for that sensor. However, we believe this is not an effective strategy. For instance, if the majority of sensors are deactivated and their last recorded values are all within the "Low risk" range, it could hinder the accurate detection of a critical condition in the patient. On the other hand, if they are all in the "High risk" range, it may cause unnecessary alarms.

To address this issue, whenever the Monitor observes that a sensor is deactivated, it triggers the need for adaptation and the Analyse component decides that a change in the *DSR*s should be made once it is stored. Then, the Plan component prepares a plan for the adaptation relying on replacing the outdated values from a deactivated sensor in the *DSR* with a "deactivated" label. This value will be later excluded from the computation of the Expected Outcome. The Execute component is responsible for launching a function that does it.

To better visualize the effects of adapting the testing in this scenario and increase the frequency of sensor deactivations/activations throughout the SA-BSN execution, we adjusted a parameter controlling the battery recharge/discharge rate, changing it from its default value of 0.05 to 0.65.

5.4.3.2 Scenario S2 - Different Patient Profiles

People with different health conditions exhibit different vital signs and physiological responses [99–101]. In this scenario, we take this fact into consideration and propose the adaptation of the Expected Outcome Computation function to handle patients with different profiles.

For the experiments, we chose to investigate the six patient profiles of BMI, namely Underweight, Normal Weight, Overweight, Obesity 1, Obesity 2, and Obesity 3, since the literature shows that BMI is very correlated to vitals. We here focus on two vital signs that are particularly affected by the BMI, body pressure [102], and heartbeat rate [103]. Both signs, heartbeat rate (Ecg) and body pressure (Abps, Abpd) are influenced by the BMI whether the weight is too high (e.g., [99, 100, 102, 103]) or too low (e.g., [102, 104, 105]). The idea in this scenario is the computation of the oracle by weighting the Ecg, Abps, and Abpd risk levels according to the patient profile, among the ones categorized by BMI. Thus, in this scenario we replace the default oracle described in Table 4.8 with a new one, aiming to simplify the adaptation. We describe the computation of this new oracle as follows.

Each sensor may provide values in a Low, Medium, or High-risk range. To make easier and more effective the application of weights, we define a scoring system for sensor risk levels. Thus, for a sensor i the transformation of its risk level RL_i into a score S_i is done

as follows:

$$S_i = \begin{cases} 5 & \text{if } \mathrm{RL}_i \text{ is “Low”} \\ 20 & \text{if } \mathrm{RL}_i \text{ is “Medium”} \\ 100 & \text{if } \mathrm{RL}_i \text{ is “Hard”} \end{cases} \tag{5.2}$$

Then, for each sensor i, we compute its contribution to the overall score $Overall\_Score$ based on its weight W_i. The formula for the overall score is:

$$Overall\_Score = \frac{\sum_{i=1}^{n}(W_i \times S_i)}{n}$$

where S_i and W_i are the score and the weight of sensor i, respectively, and n is the number of active sensors. The default value for W_i is 1.

Finally, based on the calculated overall score $Overall\_Score$, the thresholds for the Expected Outcomes are:

- **Very Low Risk**: $Overall\_Score < 8$
- **Low Risk**: $8 \leq Overall\_Score < 11$
- **Moderate Risk**: $11 \leq Overall\_Score \leq 20$
- **Critical Risk**: $20 < Overall\_Score \leq 36$
- **Very Critical Risk**: $Overall\_Score > 36$

Whenever the Monitor observes changes in the patient profile (e.g., a simulated normal-weight patient was replaced by an overweight one), it triggers the need for adaptation. The Analyse component decides that a change should be made to the function that computes the expected outcome (the oracle). Then, the Plan component prepares the adaptation based on the new patient profile. The plan consists of the rules for defining W_i based on the patient profile observed.

In this plan, the value of W_i is defined as follows:

- For non-priority sensors (i.e., Oxi, Term, and Glc):

$$W_i = 1; \tag{5.3}$$

- For priority sensors (i.e., Ecg, Abps, and Abpd):

$$W_i = \begin{cases} 1 & \text{if patient profile is “Normal Weight”} \\ 1.75 & \text{if patient profile is “Overweight” or “Underweight”} \\ 1.85 & \text{if patient profile is “Obesity 1”} \\ 1.90 & \text{if patient profile is “Obesity 2”} \\ 2 & \text{if patient profile is “Obesity 3”} \end{cases} \tag{5.4}$$

This approach allows us to prioritize specific sensors while still considering the overall input from all sensors, yielding a more nuanced and actionable criticality assessment. Finally, the Execute component is responsible for launching a command to send the suitable value of W_i to the Expected Outcome Computation. This value of W_i will be used in the computation of the oracle until the patient profile changes again.

5.4.3.3 Scenario S3 - Patient in Critical Condition

In this scenario, we suppose that human and physical resources are limited and we aim to test the SA-BSN only if the patient is in a very critical situation. Thus, we consider that the default testing strategy, described in Section 5.1.4 is replaced with a function that looks at a boolean parameter *ts* indicating when it is time to test. This parameter is always set to “false” unless an adaptation changes its value.

We implemented this scenario by using the Monitor component to observe *DSR*s coming from the Patient Simulation. If the patient is in a very critical situation, that is, more than one sensor value presents high risk level in a *DSR*, it triggers the need for adaptation. Then, the Analyse component decides that an adaptation should be made to the Testing Strategy component. Next, the Plan component describes how this adaptation should be made, that is, the *ts* should be changed to “true”. The Execute component launches a function to execute this adaptation to the Testing Strategy. This *DSR* will be then considered for testing.

5.4.4 Baseline

In our experiments, we consider one Baseline for each scenario (i.e., S1, S2, and S3). The baseline consists of the testing components of *AdapTA* with the same configuration as in the respective scenarios but without performing adaptations. In other words, referring to Figure 5.1 and specifically to the Monitor component, this means to always take the "no need for adaptation" arrow. The baseline is described in the following.

5.4.4.1 Baseline - S1

As previously mentioned, when a sensor is deactivated, the SA-BSN considers the last value of the sensor when computing the BSN outcome. Thus, the non-adaptive version of scenario S1, i.e., the baseline, consists of doing the same and using this value to compute the Expected Outcome instead of ignoring deactivated sensors.

5.4.4.2 Baseline - S2

Scenario S2 considers that the oracle for patients with different profiles should also be different. Thus, it acts by assigning weights (W_i) to each sensor based on the patient profiles. The baseline for this scenario corresponds to removing the weights from the oracle and considering that all patient profiles should be treated equally. In practice, it means assigning $W_i = 1$ regardless of the sensor or patient profile.

5.4.4.3 Baseline - S3

In scenario S3, the default test strategy condition described in Section 5.1.4 is replaced with a condition that corresponds to testing whenever a patient is in a critical situation. The baseline for this scenario is exactly the use of the default test strategy condition, that is, a testing session is launched every w seconds and performed for t seconds, consisting of a periodical test strategy. In our experiments, we use $t = 60$ and $w = 300$, since these values seemed to provide a feasible balance between testing and pause.

5.4.5 Experimental Setup and Procedure

The experiments were conducted on a computer running Ubuntu 20.04 LTS, equipped with 16GB of RAM and a processor with 4 cores, each running at a speed of 2.3 GHz. Additionally, the ROS Noetic[1] framework for Ubuntu 20.04 was required to execute the SA-BSN. The installation of the SA-BSN was carried out by following the instructions provided here[2]. For a more realistic BSN experience, we also disabled a parameter in the SA-BSN that causes a sensor's drained battery to recharge instantly. Since the *AdapTA* and the baseline utilize DTMCs and involve the generation of random numbers, they exhibit non-deterministic behavior in each execution. To ensure the validity of the experiments, we conducted 5 runs for each scenario proposed for both *AdapTA* and the baseline.

5.5 Experimental Results

This section covers the experimental data, the statistical analysis conducted, and the potential threats to the validity of the experiments. The experiments evaluated the Passing Test Case Rate (PCTR) of *AdapTA* across the three proposed scenarios (S1, S2, and S3). The outcomes of the experiments demonstrate that *AdapTA*, when compared to the baselines, offers an effective solution for testing Body Sensor Networks. Precisely:

- *AdapTA* achieved an average PTCR of $69.44\% \pm 6.08$ for S1, while the experiment with the baseline for the same scenario resulted in an average PTCR of $88.67\% \pm 2.28$.
- *AdapTA* achieved an average PTCR of $72.49\% \pm 5.67$ for S2, while the experiment with the baseline for the same scenario resulted in an average PTCR of $89.57\% \pm 2.75$.
- *AdapTA* achieved an average PTCR of $61.44\% \pm 3.20$ for S3, while the experiment with the baseline for the same scenario resulted in an average PTCR of $88.05\% \pm 1.29$.

[1] http://wiki.ros.org/noetic
[2] https://github.com/lesunb/bsn

The full experimental results are shown in Tables 5.2 and 5.3. For data analysis, we adhered to the guidelines outlined in [83, 84]. To ensure reproducibility, all scripts and datasets have been made available[3].

TABLE 5.2: Passing Test Case Rate (PTCR) for the different scenarios of *AdapTA* and the baselines.

	Ex1	Ex2	Ex3	Ex4	Ex5	**Avg**	Std
Baseline - S1	89.46	84.17	90.46	89.75	89.53	**88.67**	2.28
AdapTA - S1	73.3	68.12	69.35	59.12	77.30	**69.44**	6.08
Baseline - S2	92.22	90.72	89.68	84.33	90.92	**89.57**	2.75
AdapTA - S2	72.16	82.57	73.36	66.13	68.23	**72.49**	5.67
Baseline - S3	86.29	89.77	88.83	86.85	88.50	**88.05**	1.29
AdapTA - S3	56.76	65.65	61.98	63.78	59.04	**61.44**	3.20

5.5.1 Empirical Data

The data for analysis is gathered by logging sensor readings, the BSN outcome, and calculating the expected outcome. For both the proposed approach and the baseline in each scenario, the difference between the BSN outcome and the expected outcome is computed for each test case.

Due to the different behaviors of the approaches and the non-determinism of the SA-BSN, the number of test cases varies significantly across the different scenarios. For both *AdapTA* and the baseline in scenario 1, each of the 5 experiments, consisted of 13 patient profiles that ran for one hour each, resulting in approximately 340637.4 test cases in each experiment. Scenario 2 contains only the BMI patient profiles, resulting in 6 patient profiles that run for one hour in each of the 5 experiments, and an average of 204027 test cases per experiment for both *AdapTA* and baseline. Finally, scenario 3 consists of the 13 patient profiles that run for one hour in each of the 5 experiments, but results in different amounts of test cases for *AdapTA* and the baseline. While the baseline logs approximately 368442.6 test cases, *AdapTA* in this scenario collects an average of 212031.8 test cases, since it focuses on testing the SA-BSN only when the patient is in a critical situation and not periodically as the baseline.

[3]https://anonymous.4open.science/r/AdapTA_Paper_AST-3715/

TABLE 5.3: Passing Test Case Rate according to the different patient profiles for the Baselines and the scenarios we propose for *AdapTA*.

Patient Profile		Baseline-S1		S1		Baseline-S2		S2		Baseline-S3		S3	
		Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
ICU	MedicalICU	88.77	2.89	55.82	11.89	-	-	-	-	86.44	2.82	60.60	6.04
ICU	CoronaryCareUnit	91.90	2.10	72.74	11.58	-	-	-	-	90.96	3.10	66.68	1.71
ICU	SurgicalICU	84.65	3.82	60.47	6.34	-	-	-	-	81.58	6.07	65.00	5.97
ICU	CardiacSurgeryUnit	93.57	2.42	68.44	12.71	-	-	-	-	90.06	4.48	57.54	6.27
Age	16-29	91.79	2.15	62.47	9.64	-	-	-	-	88.11	2.60	70.39	8.52
Age	30-59	85.77	5.97	65.44	9.74	-	-	-	-	84.18	6.05	64.52	7.13
Age	60+	84.86	5.42	61.54	4.46	-	-	-	-	86.03	5.99	63.12	5.51
BMI	Obesity3	88.46	1.65	69.83	7.72	85.58	2.01	61.28	4.42	82.10	4.66	59.64	5.05
BMI	Obesity2	92.44	4.77	67.88	8.01	87.99	8.00	68.15	10.00	83.87	2.29	61.14	2.54
BMI	Obesity1	93.06	3.92	65.66	7.63	90.09	2.09	70.69	4.17	89.44	5.33	51.76	10.85
BMI	Overweight	91.54	5.17	63.79	14.03	88.52	3.70	65.45	6.93	89.97	2.31	58.81	5.57
BMI	Normalweight	89.09	4.76	69.52	12.33	86.14	6.30	86.14	6.30	85.21	6.52	58.27	6.82
BMI	Underweight	88.62	5.61	67.60	11.05	89.40	4.54	66.45	7.38	88.97	3.52	62.79	7.43

5.5.2 Data Analysis and Discussion

To select an appropriate approach for statistical analysis, we thoroughly assess the characteristics of our results [83]. After confirming that our data does not follow a normal distribution, we conduct the Mann-Whitney U Test [106] to evaluate at a 5% significance level the null hypothesis that there are no statistically significant differences in PTCR between *AdapTA* and the baselines. Comparing the baseline approach with the proposed method, the p-value obtained across all scenarios was 1.219e-2, with a U statistic of 25. This confirms a significant difference in effectiveness between the two approaches, at the 95% confidence level.

A significant Mann-Whitney U test indicates that one testing approach stochastically dominates the other, however it does not specify the precise relationship between the pairs of techniques. We conducted pairwise comparisons (testing approach with each scenario against its respective baseline) using the Vargha-Delaney effect size (A12 measure) [86] following the Mann-Whitney U test. For all the pairs, the value of A_{12} measure was equal to 1, which represents the maximum effect size, indicating extremely strong domination of the Baseline by *AdapTA*.

Based on the statistical analysis of the results presented in Table 5.2, *AdapTA* proves to be the most effective method for detecting failures compared to its non-adaptive counterpart, the baseline, as evidenced by the PTCR values. In Scenario 1, the significant difference in terms of PTCR between the Baseline and *AdapTA* provides evidence that using the last value received from a sensor when it gets deactivated at runtime may not be the best solution, since these old values could distort the actual patient's condition. Thus, adapting the test cases to no longer include these values may be more beneficial,

and can capture more failures. In Scenario 2, the notable difference in PTCR between the Baseline and *AdapTA* suggests that adapting the oracle to handle different patient profiles is important since a more precise and personalized oracle may result in finding more failures. Finally, Scenario 3, with the most significant difference in terms of PTCR between *AdapTA* and the Baseline, demonstrates the importance of testing when the patient is in a critical situation. This may be the context in which the SA-BSN should perform best, since its malfunction, in this case, could lead to serious harm to the patient or even death. Therefore, proper attention should be given to this scenario.

Table 5.3 reports the PTCR by patient profile and scenario for both *AdapTA* and the baseline. Since the adaptation performed in Scenario 1, the deactivation/activation of sensors, is not directly impacted by the patient profile, as in the other scenarios, we could not see a clear trend in the difference in PTCR between the Baseline and *AdapTA*. The "MedicalICU", "16-29", and "Overweight" patient profiles present the highest difference for ICU, Age, and BMI, respectively. For Scenario 2, as expected, the difference between the PTCR of *AdapTA* and the baseline grows as the patient's criticality level, based on BMI, rises. This highlights the increasing importance of adaptation, which becomes even more pronounced as the patient's condition becomes more critical. Scenario 3 intrinsically focuses on critical condition patients. Thus, as expected, the difference between the PTCR of *AdapTA* and the baseline is larger for patients considered at higher risk. Although all ICUs are important as they admit patients with varying degrees of criticality, we believe that the ICU for cardiac surgery patients is the most severe. Therefore, confirming this hypothesis, "CardiacSurgeryUnit" shows the greatest difference among all the ICU profiles. The same applies to "Age", as the "60+" patient profile is most impacted by the use of adaptation in this scenario. Finally, regarding the BMI, the greatest difference in PTCR was for "Obesity 1" and not "Obesity 3" as expected. However, this may be explained by the fact that it is not only the BMI that impacts the patient's condition but also other factors, such as age and lifestyle.

5.6 Threats to Validity

Despite our best efforts, potential threats to the validity of the results might remain. We present four validity key aspects following the classification in [90] as follows.

Construct validity: whether the designed experiments are appropriate for assessing the

effectiveness of *AdapTA* in all the scenarios. Our validation demonstrates that *AdapTA* successfully identified previously unknown failures in SA-BSN and outperformed the Baseline in terms of effectiveness. While we used PTCR, a widely accepted metric for measuring effectiveness, we acknowledge that the use of alternative metrics (e.g., Risk Coverage) could potentially yield different results. Besides, we compared *AdapTA* solely with our Baseline due to the scarce availability of comparable testing approaches. However, we consider the Baseline a strong choice in this case, as in this way we can demonstrate that incorporating adaptation logic into a testing approach can help uncover more failures. Another potential threat to validity not only in *AdapTA* but also in the Baseline approaches lies in utilizing DTMCs where the initial state is consistently set to the low-risk level. However, this choice follows the original implementation of SA-BSN.

Internal validity: whether the observed results were actually caused by the "treatment" rather than other factors. We adhered to established guidelines for conducting the experiments and performing statistical analysis. A common internal threat is the precision of measurements, which can be affected by random factors. To mitigate this potential issue, we repeated the experiments 5 times.

External validity: whether and to what extent the observations can be generalized. *AdapTA* is a broadly applicable approach, but for the sake of clarity, we focus on a BSN within the healthcare domain. Even so, to date, our results are solely based on the SA-BSN case study. Additional experiments are necessary to more thoroughly validate the effectiveness of *AdapTA* and explore its potential application in other BSN domains, or even, other evolving systems.

Reliability: Whether and to what degree other researchers can reproduce the results. To ensure reproducibility, we provide all data and configuration details in a replication package.

Chapter 6

Related Work

In this chapter, we first describe the state of the art for SATF according to some existing secondary studies. Then, we provide an overview of the literature related to the approaches we propose in this thesis, *GATE4BSN* and *AdapTA*. For the comparison with *GATE4BSN*, we examine the literature to look for approaches to testing BSNs. On the other hand, for the comparison with *AdapTA*, we focus on exploring Self-Adaptive Testing in the Field approaches.

6.1 State of the Art for SATF

As detailed in Chapter 3, to build awareness of the problem we surveyed the literature on approaches for self-adaptive testing in the field. In this section, we conduct the analysis of existing secondary studies that are related to this work and some additional works that we identified by performing a specific search. For example, we searched for works discussing the 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 analyzed 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) was carried out in 2016 aiming at recognizing and categorizing the challenges encountered when testing SAS [107].This paper gathered 25 primary studies

(published from 2003 to 2015) and identified 12 types of challenges to be faced. The challenges mentioned in these works 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), context-dependency (Challenges 6 and 7), or test cases maintenance (Challenge 8), among others. In Section 3.5 we referred 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 [108] 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 3.3.2 and 3.3.3). From what is described in the paper, it seems that they are mostly referring to independent oracles. It is worth mentioning that the oracle problem in general is investigated in [109].

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 mitigate some of these challenges, and discussed this more in depth in Section 3.5.

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 [1] 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 it is 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 [110] limits the scope to approaches to test systems that are dynamically adaptive and distributed, and in contrast to the SLR in [1], which did not limit the domain of the tested application, the authors actually chose fewer papers (precisely, 43 from 2006 to 2020 in [110], against 80 from 1989 to 2017 in [1]). In [110], the authors pose eight research questions about the properties of runtime test approaches, and 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 the adaptation of the testing approach itself as we do here.

Lastly, a recent SLR [111] 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 6.1 shows an overview of the above-mentioned secondary studies. It is important to notice that the work in [108] 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 [110], 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.

6.1.1 Comparison with Challenges Highlighted in Related Surveys

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 by other secondary studies.

The uncertainty challenge (Section 3.5.1) includes many of the challenges identified in [107] and previously mentioned, but from a different perspective. In fact, the challenges in that work surveying approaches testing adaptive systems are basically related

Table 6.1: 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
[107]	2016	Challenges when testing SAS	-	✓	2003-2015
[111]	2020	Faults when testing adaptive systems and context-aware systems	-	✓	-2019
[110]	2021	Testing in the field of dynamically adaptive and distributed systems	✓	✓	2006-2020
[1]	2021	Testing in the field	✓	-	1989-2017
[108]	2022	Challenges when testing a specific class of SAS	-	✓	-
This work	2023	Self-adaptive testing in the field	✓	✓	2012-2022

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 at the challenges reported in [1], 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 cases before going into 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 [1] 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 analyzed, however, these are important challenges that look relevant also for SATF approaches.

As previously discussed in this chapter, the work in [108] presents the main challenges of a specific class of self-adaptive systems. Our uncertainty challenge (Section 3.5.1) 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 the adaptation towards the changes of the SUT over time. This permits also to tune over time the testing approaches towards dealing with concurrent faults that can mask each other and nondeterminism. Moreover,

a self-adaptive testing approach enables the definition of 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 [108], since we could not identify them as important ones in the data we collected.

6.2 Comparing *GATE4BSN* to the Literature

In this section, we first explore the key characteristics that are essential for effectively testing BSNs. Then, we proceed to a comparative analysis of the existing literature.

6.2.1 Key Characteristics When Testing BSNs

When evaluating strategies for effectively testing BSNs, several key characteristics must be considered to ensure comprehensive and reliable assessment. These characteristics form the foundation for developing robust testing strategies for BSNs in real-world scenarios. We describe them as follows.

Dynamic Testing with Time Window. BSNs involve sensors monitoring real-time physiological data, which are sensitive to dynamic changes in the body and can vary significantly over relatively short periods. Thus, the Dynamic Testing of BSNs with a Time Window ensures that the system is tested under real-world conditions that may fluctuate over time. Testing approaches with this characteristic validate the BSN's ability to respond accurately and adaptively to varying physiological and environmental changes, ensuring reliable performance in real-time scenarios. When testing Self-Adaptive BSNs, this characteristic becomes even more important, due to the adaptive nature of these systems.

Black-Box. Black-box testing approaches focus on evaluating the functionality of a system or component based on its inputs and outputs, without any knowledge of its internal code or structure. In many cases, BSNs are tested by third parties or users who may not have access to or knowledge of the system's internal code. Black-box testing approaches allow these external testers to evaluate the system's functionality and ensure

that it meets user requirements and medical standards without needing to understand or access the code behind it.

Combinatorial Input. Since each sensor in a Body Sensor Network (BSN) measures a distinct vital sign and can generate multiple possible readings, the resulting combinations of sensor inputs multiply rapidly, leading to a vast and complex array of potential scenarios. Combinatorial testing is especially valuable for BSNs as it systematically manages this complexity, enabling efficient testing across diverse conditions to ensure accurate and reliable testing. By examining various combinations of sensor values, it addresses both predictable and unexpected interactions among multiple sensors, identifying potential issues that might only be triggered under specific combinations of conditions.

Risk-Based. Risk-based approaches prioritize tests based on the potential risks associated with the software, focusing on areas where failures would have the most significant impact. This strategy aims to identify, assess, and mitigate the highest-priority risks first, ensuring that limited testing resources are applied where they are most needed to protect against critical failures. Risk-based approaches are particularly interesting when testing BSNs since they are often used in critical health-related applications where failures can have serious consequences. By focusing on the highest-risk scenarios, this approach ensures that the BSN's most essential and failure-prone components are rigorously tested, providing a higher level of assurance in its reliability.

Fusion of Sensors (Input). Testing the data fusion function in BSNs, rather than sensors individually, is critical for enhancing the accuracy of health assessments and improving decision-making. By integrating information from multiple sensors, this function provides a comprehensive view of a patient's condition, helping to identify trends or anomalies that might be missed when analyzing individual sensor data. Thus, it must be extensively tested. The correct operation of this function reduces possible noise from the sensors, increases the reliability of the BSN, and ensures resilience against sensor failures, allowing the BSN to maintain functionality even if one sensor malfunctions.

6.2.2 Analysis of Prior Studies

Since we did not find testing approaches for BSNs in the literature, we discuss three related research areas: (i) gaining confidence through fault diagnosis of BSNs, (ii) model-based testing using Markovian Models (M.M.), and (iii) approaches combining models with combinatorial testing.

Table 6.2 provides a summary of the comparison between the proposed approaches and related studies in the literature. The symbols ✓ and ✗ indicate whether or not a strategy possesses a specific characteristic, respectively. The symbol ∼ denotes that the approach can be readily adapted to address this characteristic, while – signifies that the evaluation is not applicable, as the work is a secondary study and thus does not introduce new approaches. Given the importance of making available the source code or a replication package when proposing an approach for reproducibility, we also evaluate the related works based on this attribute. As shown in the table, only one secondary study provides a replication package [112], along with one paper that offers a tool [113], which significantly limits the use of most approaches.

TABLE 6.2: Comparison of *GATE4BSN* and the existing literature.

	Related Work								**GATE4BSN**		
	Fault Diagnosis of BSN		**Model-based testing with M.M.**				**Models and Comb. testing**				
Characteristics	[114]	[115]	[116]	[117, 118]	[112]	[113, 119]	[120]	[121]	PASTA	ValComb	TransCov
Dynamic testing with time window	✓	✓	–	✗	–	✓	✗	✗	✓	✗	✓
Black-box	✓	✓	–	✓	–	✓	∼	✓	✓	✓	✓
Combinatorial input	✗	✗	–	✗	–	∼	✓	✓	✓	✓	✗
Risk-based	✗	✗	–	✗	–	∼	✗	∼	✓	✗	✗
Fusion of sensors (Inputs)	✓	✗	–	✓	–	∼	✓	✓	✓	✓	✓
Replicable work	✗	✗	✗	✗	✓	[✓, ✗]	✗	✗	✓	✓	✓

When gathering confidence in BSNs, a prominent approach is to use fault diagnosis during system execution. The work in [114] uses multi-sensor fusion approaches to identify faulty sensors by assuming correlations between sensorial measurements and diseases. Their algorithm identifies permanent faults, intermittent faults, and transient faults, by re-executing the test cases with a set frequency. Based on the comparison between readings obtained from correlated sensors (e.g., Heartbeat Rate and Body Pressure), it decides whether a sensor is faulty. Instead, Zhang et al. [115] describe a method using hidden Markov models (HMMs) to identify faulty readings of individual sensors. Even if both approaches do not constitute a testing process, they still look for faults in the sensors over time (dynamic testing with a time window). Neither of the two previously

mentioned approaches is risk-based or considers combinatorial inputs. Also, they are both black-box since they evaluate a sensor based on input data. Finally, the work in [114] assesses the fusion of sensors by considering their spatial correlation, that is, the correlation between different sensors, to determine if a sensor is faulty. Differently, the work in [115] considers temporal correlation, that is, the correlation between the current value of a sensor and its history. Therefore, sensor fusion is not evaluated to any extent. None of these approaches address the data fusion algorithm at the application layer level, but at the body level, as opposed to ours.

A taxonomy for model-based testing approaches is proposed in [116] and their work discusses Markov chains for statistical testing, which can be used for test case generation or specifying oracles. One of the tools collected in their secondary study is JUMBL [117], a Java class library that supports all phases of model-based statistical testing and offers a command-line interface designed to interact with Markov chain usage models. Also, making use of Markov models as usage models, Prowell [118] defends the application of concurrency operators to the test cases generated from simple Markov chain usage models, to create sophisticated test cases when testing complex systems. Markov models are also employed to test case prioritization as shown by Barbosa G. et al. [112] in their systematic literature review. This study shows that Markovian models are used for test case prioritization in six contexts: controlled Markov chains, usage models, model-based testing, regression testing, statistical testing, and random testing.

Finally, Markov chains have been used in the application of cybernetic principles to software testing [113, 119]. Building on the concept of software cybernetics, the Controlled Markov Chains (CMC) approach treats software testing as a control problem. The work in [113] discusses how to do so and how the CMC approach to software testing determines an optimal testing strategy. In their subsequent work [119], the same authors examine the behavior of the optimal test profile identified through the CMC approach to software testing and introduce the concept of adaptive software testing. Adaptive software testing adjusts the software testing strategy online by using testing data collected during software testing in response to changes in the understanding of the software under test. Among these works, the works in [112, 116] are secondary studies, and therefore they do not propose novel testing approaches. On the other hand, the works in [117, 118] do not employ dynamic testing nor combinatorial input. Test cases are derived from a probabilistic usage model, making the approach black-box, and non-risk-based. Test

cases are a fusion of input data since they are paths in the usage model. Finally, the works in [113, 119] consider a dynamical system and the testing should also be dynamic to either adapt to changes in the system or to consider the result of previous test cases. They are also black-box since test cases are extracted from the optimal test profile. With respect to the other key characteristics, these approaches may be adapted to consider them.

Intertwining models and combinatorial test case generation to the systematic collection of safety evidence has shown effectiveness in the work by Nguyen et al. [120]. In this paper, test sequences are derived from inferred models and complemented with selective test input combinations. Dynamic testing is not addressed, and their approach does not incorporate risk-based strategies. However, if the inferred model is based on the specification, their method can be adapted to become a black-box approach. Additionally, they explore the t-way combinatorial fusion of input data as in our approach. Following this line of research, Gannous et al. [121] employ a previously proposed framework for testing safety-critical systems, known as Model-Combinatorial based testing (MCbt). MCbt is a framework that proposes an integration of model-based testing, fault analysis, and combinatorial testing to produce the maximum number of evidences for an efficient safety certification process but was never actually used to derive a specific testing approach. In this paper, they present a concrete application of MCbt with an application to a case study. As in the previous work, dynamic testing is not addressed, but their approach allows the incorporation of risk-based strategies by deriving failure paths for behavioral models of each component in the system. These paths may be used to generate high-risk test cases. Their approach is also black-box since the tester creates behavior models using system requirements and specifications. Test paths are also combined for two or more components making this strategy combinatorial and based on fusion of input data. T-way combinatorial testing is mentioned as future work to overcome the state space explosion problem.

6.3 Comparing *AdapTA* to the Literature

AdapTA is proposed as a model-based approach for Self-Adaptive Testing in the Field (SATF) of BSNs. Indeed, SATF is a branch of testing that is recently drawing interest, yet so far has not been extensively explored in the literature. For comprehensiveness, we

can leverage the already described systematic review provided in Chapter 3 and published in [32], which collected 19 papers over the period from 2012 until mid-2022. In search for more SATF papers that could have appeared after 2022, we launched the same query on the same repositories, filtering for the period 2022-2024. By applying the same selection criteria we used, we eventually found two more papers (namely [122] and [123]) that added to the original 19 ones make a total of 21 papers proposing SATF approaches.

We then reviewed these 21 papers: first, we noticed that while many of them are employed to test adaptive systems, none of them addresses BSNs and could be directly applied for comparison against *AdapTA*. Then, to also take into account similar works that could be possibly adapted for addressing BSNs, we looked at the features exposed by the reviewed approaches, including: the continuous use of a monitor to detect the possible need for adaptation; the type of adaptations that are performed; no human in the loop to manage the adaptation; possibly application of a model-based approach for test generation. For instance, Gazzola et al. [68] use data collected from the field to generate test cases that could trigger yet unexplored execution states. In principle, their intent is similar to what we do by using patient data to build the DTMC that is then used to test the BSN. However, their approach targets unit testing and is conceived for testing in production, while we conduct ex-vivo testing at the system level.

SAMBA [63] is a Self-Adaptive Model-BAsed online testing framework addressing Service Oriented Architecture (SOA). When its monitor detects a change in the SOA under test, the Model Generator component is responsible for modifying the model that is referred for test generation. Then the newly generated test cases are executed. The overall scheme is partly similar to our approach functioning for what concerns our adaptation of test cases and/or test oracle. However, the types of system targeted (SOA vs. BSN) differ widely, and consequently, the implementation of test generation and execution modules are completely different. Moreover, they execute test cases in production and not ex-vivo.

Perhaps the most similar approach is the framework WS-REPAS [16], which combines a DTMC modeling approach with monitoring for in vivo testing of services. Precisely, they collect field data through monitoring and use them to continuously update the parameters of a DTMC model. When changes are observed, testing is triggered, which

executes an in-vivo testing session. Differently from *AdapTA*, though, WS-REPAS aims at non-functional testing.

Concerning approaches that are specifically conceived for testing BSNs, we found in the literature works conducting fault diagnosis to gather confidence in BSNs [114, 115]. The work in [115] proposes a method that utilizes Hidden Markov Models (HMMs) to detect faulty readings from individual sensors. Similarly, the study in [114] uses multi-sensor fusion approaches to identify faulty sensors by assuming correlations between sensorial measurements and diseases. However, neither of them is performing testing. Finally, our work described in Chapter 4 outlines a strategy for testing BSNs, the *GATE4BSN*. Indeed, *AdapTA*, described in Figure 5.1, is partly inspired by *GATE4BSN* approach. However, *GATE4BSN* lacks field data and does not adapt the testing strategy, unlike *AdapTA*.

Chapter 7

Discussion

In this chapter, we provide a discussion on how the proposed approach *AdapTA* relates not only to the taxonomy and the reference architecture for SATF provided in this work but also to the challenges found in the literature. Therefore, we first apply both the taxonomy and reference architecture to *AdapTA*. Then, we discuss the challenges in SATF that *AdapTA* addresses and highlight those that remain unresolved.

7.1 The Taxonomy and the Reference Architecture Applied to *AdapTA*

In Chapter 3, we presented our work, published in [32], which includes a comprehensive literature review. This review highlights that Self-Adaptive Testing in the Field (SATF) remains an underexplored area in research, as evidenced by the collection of only 19 relevant papers. Even though many of these works are employed to test adaptive systems (e.g., [15, 55, 56, 61, 63, 68]), none of them test BSNs.

As stated in Chapter 3, since there was no standard taxonomy when it comes to SATF approaches, we proposed a feature model, based on the collected studies, containing the features that a SATF approach may have. We here discuss how *AdapTA*, our proposed SATF approach, handles these features, and illustrate examples of other works, among the collected in the review, that also implement them.

7.1.1 The Taxonomy Applied to *AdapTA*

Monitoring is the continuous process of gathering, analyzing, and collecting data on the execution of the SUT. According to Chapter 3, the monitor is an optional component in SATF approaches. *AdapTA* has a monitor feature that continuously (e.g., [14–16, 55, 57–64, 66–68]) monitors the environment change or evolution, and the SUT adaptation and evolution (e.g., [16, 57–59]). As most of the works collected in Chapter 3, in *AdapTA* changes are foreseeable (e.g., [14–16, 18, 55, 57, 59–64, 66–68]) by the monitor ahead of time.

In addition to having test cases, which is a mandatory feature in SATF approaches, *AdapTA* also contains an oracle that is independent of them and does not rely on human involvement (e.g., [16, 18, 55, 57, 59–61, 63, 65]). The class of field testing of our test strategy is Ex-vivo (e.g., [16, 67]) and we test functional (e.g., [14, 15, 17, 18, 55, 57–63, 65, 66, 68]) requirements at the system level (e.g., [14–18, 56, 57, 61–67]). Moreover, our approach uses a Model-Based testing criteria (e.g., [16, 55, 58, 59, 61, 63, 66]) since we extract input data from DTMCs.

Regarding the adaptation feature, the object to adapted by *AdapTA* are test cases (adaptation of existing ones - e.g., [18, 57, 61, 62]) and test strategy (e.g., [14, 17, 18, 55, 58–61, 64, 67]). The activation of the trigger for adaptation is based on the SUT (e.g, [14, 16, 18, 55, 57, 59, 61–63, 66, 67]), what in their definition also includes its environment. In *AdapTA*, the trigger is reactive (e.g., [14–17, 55, 58, 59, 61, 63, 65–68]) which means that it "responds when a change has already occurred" [32]. In *AdapTA* the type of adaptation is "External" (e.g., [64, 68]) as the logic of the testing approach and adaptation mechanism are not merged.

We also consider that as all the works collected in Chapter 3, the decision-making process for the adaptation in *AdapTA* is *static*, since any changes require recompiling and redeploying the system or its components. With respect to the adaptation technique, *AdapTA* adapts by changing the *structure* (e.g., [14–18, 57, 59–63, 65, 68]) of the testing system, by changing test cases or the testing strategy. The *Openness* reflects the level of flexibility within the set of adaptive actions [11]. As most of the approaches collected in [32], we consider that *AdapTA* is a *close-adaptive* system (e.g., [14–18, 55, 57–65, 67, 68]), that is, it has a predetermined number of adaptive actions; hence, no additional action may be added at runtime. Finally, we also assess *AdapTA* according to the level of

decentralisation held by the adaptation logic [12], without considering the application logic. *AdapTA* has a centralised adaptation logic (e.g., [15–18, 55, 57, 59, 61–68], which means that a central unit has to control the adaptations.).

7.1.2 The Reference Architecture Applied to *AdapTA*

With respect to the architecture, as shown in Figure 5.1, we used the reference architecture shown in Figure 3.4 as a guide for the development of *AdapTA*. For this reason, *AdapTA* has many elements in common with the reference architecture. Among them, we could highlight the fact that we also define our adaptation logic based on the MAPE-K Loop, so we also share the Monitor, Analyze, Plan, and Execute components. In *AdapTA*, the Knowledge component is the Data Sensor Reading (*DSR*). In the reference architecture, the Knowledge component is the "Database of test cases and oracles, and of the model of the SUT". Another difference is that in *AdapTA* the Execute component contains only the SATF adaptation, while in the reference architecture, it is also composed of the test strategy, test execution, and oracle independent from test case components.

7.2 Challenges Addressed x Still Open

Dealing with challenges in SATF is a complex and demanding task that requires addressing a wide range of unpredictable factors. Unlike traditional software testing, which operates in controlled environments with predefined conditions, SATF deals with real-world systems that continuously evolve. These systems, often embedded in dynamic and diverse contexts, need to adapt their testing strategies at runtime, which introduces a layer of complexity that is difficult to manage.

Among the challenges found and explored in Section 3.5, we believe that in this work we contribute to the advancement of SATF by addressing Uncertainty with *AdapTA*. In a way, we try to anticipate and predict the possible changes that may occur in the SUT or in the environment (the patient). Thus, whenever these changes happen, based on the type of change, we perform a self-adaptation in the testing process.

Regarding the Overhead and Isolation challenges, since our approach is ex-vivo, meaning that we use field data to test in-house, we do not face this issue. However, concerning human intervention, this could indeed be a problem even in ex-vivo approaches. Nevertheless, up to this point, with the possible adaptations considered, there has been no need for human involvement, which is an advantage of the proposed approach *AdapTA*.

Chapter 8

Conclusions and Future Work

This thesis offers a comprehensive exploration of Self-Adaptive Testing in the Field, providing valuable insights and in-depth discussions on its key concepts, challenges, and potential applications.

First, we presented a new category of testing called self-adaptive testing in the field. The need for 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 thesis, we offered a taxonomy of self-adaptive testing in the field together with a reference architecture of SATF 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 variety of challenges to be solved in SATF approaches, which, when compared to the number 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.

Second, to gain insights into how the testing of BSNs, the type of system we address here, should be conducted, we proposed *GATE4BSN*, a novel approach to test BSN applications. *GATE4BSN* is a generic and abstract approach that is instantiated in three

approaches, namely: (i) *PASTA*, which simulates patients by considering a set of sensors and by mimicking the trend of each sensor via a DTMC; (ii) *ValComb*, which makes use of all possible combinations of sensor risk levels to explore different behaviors of a patient; and (iii) *TransCov*, which employs a dummy patient in which all the transitions in the DTMCs are labeled with equal probability. Then, their DTMCs are executed until all the transitions are covered.

We evaluated the three instances of *GATE4BSN*, by using them to test a self-adaptive BSN system from the literature, and they were able to detect some unknown failures. Through experiments, we compared the three approaches among themselves and with a random baseline approach. We could then conclude that our approaches outperform the random baseline in terms of effectiveness. Also, statistical analysis showed that *PASTA* is the most effective instance of *GATE4BSN*. *ValComb* is in second place with effectiveness value close to *PASTA* but with ET approximately 76 times smaller than for *PASTA*. By clustering non-passing test cases from *PASTA* and *ValComb* we noticed that there exist many test cases containing a combination of risk labels that are failing in *PASTA* but are passing in *ValComb*. This indicates that *PASTA* is effective in finding faults that are not reported by *ValComb*. Finally, *TransCov* is also fast but due to more than one non-deterministic element in its implementation, its PTCR is very unstable, presenting large variations from one execution to another.

Third, we finally proposed a SATF approach for testing BSNs, the *AdapTA*. *AdapTA* employs an ex-vivo approach, using real-world data collected from the field to simulate patient behavior in in-house experiments. We derive DTMCs to simulate different patient profiles and use them to generate the test input data for the BSN. The outputs of the BSN are compared with a proposed oracle to assess the test results. *AdapTA* adaptive logic continuously observes the SUT and the simulated patient, making testing adaptations when needed. We also presented three adaptation scenarios, detailing the trigger conditions and the adaptation policies for each.

We assessed *AdapTA* in the three adaptation scenarios using a self-adaptive BSN system and a clinical database from the literature and identified several previously unknown failures. Through experiments, we compared *AdapTA* against a baseline, which is a non-adaptive version of the proposed approach for each of the scenarios. The results,

confirmed by statistical analysis, demonstrated that AdapTA achieves greater effectiveness compared to the baseline across all three adaptation scenarios, emphasizing the value of its adaptive logic. Notably, scenario 3, which focused the testing on the patient's critical situation, showed the best improvement. This scenario likely represents the context where the SA-BSN should perform optimally, as missing an alarm could have serious consequences, hence it is crucial to give this scenario appropriate attention. Through the analysis of the rate of passing tests by patient profile, we noticed that for Scenarios 2 and 3, which take into consideration the patient criticality, the adaptation provided the greatest benefit to the most critical patient profiles. On the other hand, for Scenario 1, which does not consider the patient's conditions to perform the adaptation, we could not see a clear impact of the patient profile on the improvement brought by the adaptation.

We believe that our contributions in this thesis can help shape the research in the future of this important category of testing. The proposed taxonomy and reference architecture might also help practitioners to anticipate a need they will have in the near future or to select among the available techniques those that better match their needs. Also, the reference architecture and the feature model may be employed as a guide to the development of novel SATF 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, researchers can make wise decisions when the development is still in the initial phase. In future work, we plan to further explain how engineers and researchers can use the reference architecture and the feature model to build their own SATF approach.

For *GATE4BSN* and *AdapTA*, in future works, we plan to test other BSNs in different domains. Also, we will explore how *GATE4BSN* and *AdapTA* can be used for testing other systems that would benefit from our idea of simulating the input (our patients' simulation), such as state-based systems (e.g., CPSs and autonomous systems), which often depend on a specific order of inputs that eventually lead to failures [124]. Specifically, the adaptations in *AdapTA* may provide significant improvements in the effectiveness when testing evolving systems. An example of these systems is autonomous drone navigation

systems [125] that may benefit from risk-based testing since it focuses on high-risk situations, such as flying near obstacles or in inclement weather, which could lead to loss of control or collision. Industrial robotics for manufacturing [126] is another example of an application that could take advantage of dynamic testing with a time window, since real-time adjustments are necessary for robots to respond to moving parts, changing positions, or unexpected obstacles. Testing within a time window ensures that robots can make adjustments quickly to avoid collisions or errors. Finally, in *AdapTA*, since our adaptation logic is external to the testing components, it may also be easily adaptable to include new scenarios, or even the proposed scenarios may be expanded to encompass other patient profiles, potentially more accurate oracles (defined for example for an expert in the domain), and other implementations of testing strategies.

8.1 Publications

During the development of this thesis, the following papers were accepted for publication at software engineering conferences or journals. We note that [P2] is not related to the topic of this thesis, but was a collaborative effort with other colleagues. [P1] is mostly related to the work seen in Chapter 3, while [P3] is the journal version of our systematic literature review, and relates to Chapters 3, 6, 7 of this thesis. [P4] contains part of our in-house approach, the *GATE4BSN*, and is mostly related to Chapters 4 and 6. Finally, [P5] is mainly related to Chapters 4 and 6 since it describes in details our in-house approach, *GATE4BSN*, and [P6] contains the description of *AdapTA* and thus is predominantly related to Chapters 5, 6 and 7.

Published:

[P1] Samira Silva, Antonia Bertolino, and Patrizio Pelliccione. 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*, pages 58–69, 2022.

[P2] Samira Silva, Adiel Tuyishime, Tiziano Santilli, Patrizio Pelliccione, and Ludovico Iovino. Quality Metrics in Software Architecture. In *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, pages 58–69. IEEE, 2023.

[P3] Samira Silva. Reviewing and Proposing Approaches for Self-Adaptive Testing in the Field. In *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 13–16. IEEE, 2023.

[P4] Samira Silva, Patrizio Pelliccione, and Antonia Bertolino. Self-Adaptive Testing in the Field. *ACM Transactions on Autonomous and Adaptive Systems*, 19(1):1–37, 2024.

[P5] Samira Silva, Ricardo Caldas, Patrizio Pelliccione, and Antonia Bertolino. Different Approaches for Testing Body Sensor Network Applications. *Journal of Systems and Software.* 112336, 2025.

[P6] Samira Silva, Ricardo Caldas, Patrizio Pelliccione, and Antonia Bertolino. An Adaptive Testing Approach Based on Field Data. In *6th ACM/IEEE International Conference on Automation of Software Test (AST), 2025.*

Appendix A

Additional Material

A.1 E-mail Template Sent to Authors of Collected Papers

Dear {{Name}},

I am Samira Silva, a PhD student at the Gran Sasso Science Institute, under the supervision of Professors Antonia Bertolino and Patrizio Pelliccione. Currently, we are performing a scoping review on the topic of self-adaptive testing in the field, that is an extension of a previous work published in the proceedings of SEAMS 2022 that you can check hereLink. The following paper(s) of your authorship has(have) been selected for our study:

{{Paper 1}}
{{Paper 2}}
{{Paper 3}}

Since you are one of the authors of the works we collected, and given your expertise in self-adaptive testing in the field research, we would like to invite you to contribute to this research by answering a simple questionnaire of 6 questions. As a result of our study, we aim to offer insights into the state-of-the-art of SATF.

Please answer the following form: https://forms.gle/aYXYT4rzSvexY5to8

Also, contact us if you want to discuss the topic further.

We kindly ask you to answer the questionnaire by October 17, or to let us know by then if more time is needed.

Best regards,

Samira Silva (GSSI)
Antonia Bertolino (ISTI-CNR)
Patrizio Pelliccione (GSSI)

A.2 Questionnaire Sent to Authors of Collected Papers

Self-adaptive Testing in the Field (SATF)

In our paper "*Self-Adaptive Testing in the Field: Are We There Yet?*"(SEAMS 2022), we conducted a scoping review on Self-Adaptive Testing in the Field (SATF). Based on the papers we reviewed, we derived a definition for SATF, a taxonomy, and identified key challenges mentioned across the works. We are now extending this research by updating our set of primary studies through further literature searches and snowballing.

Through this questionnaire, which is being sent to the authors of the newly included papers, we aim to deepen our understanding of SATF and validate our interpretations regarding its features. Your input will be invaluable to this process. If you are interested in discussing this topic further, please indicate your availability for a conversation at the end of the questionnaire.

Rest assured that all data collected will be anonymized and used solely for the purposes of this study. Only the authors of the study will have access to the data.

Involved Researchers:

- Samira Silva (E-mail: samira.silva@gssi.it)
- Antonia Bertolino
- Patrizio Pelliccione

Table A.1: Introduction of the Questionnaire about Taxonomy and Reference Architecture for SATF.

Name and Surname:_____	
Q#	**Question**
1	**In our review of the literature, we observed that not all the SATF approaches have a monitor component. Thus, in our reference model we included a monitor component as optional. Do you agree?** ☐ Agree ☐ Neither agree nor disagree ☐ Disagree
2	**Please, explain/comment your answer to the previous question.** _____
3	**In a SATF approach, we noticed that the events that a monitor could expect in order to activate testing are: SUT adaptation, SUT evolution and/or configuration change made by the user, and in our reference model, we considered that more than one of these events together could activate the testing. Do you agree?** ☐ Agree ☐ Neither agree nor disagree ☐ Disagree
4	**Please, explain/comment your answer to the previous question.** _____
5	**We noticed the objects to be adapted in a SATF approach are: test cases, oracle, monitor, or the testing approach, and in our reference model, we considered that an approach could adapt more than one of these objects. Do you agree?** ☐ Agree ☐ Neither agree nor disagree ☐ Disagree
6	**Please, explain/comment your answer to the previous question.** _____

Table A.2: Questionnaire about the Taxonomy and Reference Architecture for SATF - Part 1.

Q#	Question
7	**In a SATF approach, we noticed the adaptation could be triggered: periodically, by changes in the SUT, through a policy or on-demand, and in our reference model we considered that an adaptation could be triggered by more than one of these strategies. Do you agree?** ☐ Agree ☐ Neither agree nor disagree ☐ Disagree
8	**Please, explain/comment your answer to the previous question.**_____
9	**In a SATF, we noticed the testing adaptation could be performed by making changes to: test parameters, the test structure or the test context, and in our reference model we considered that more than one of these changes could be performed by an approach. Do you agree?** ☐ Agree ☐ Neither agree nor disagree ☐ Disagree
10	**Please, explain/comment your answer to the previous question.**_____
11	**If you would like to discuss further with us about this study, please, leave here your e-mail address and we will contact you soon to arrange a call.** _____

Table A.3: Questionnaire about the Taxonomy and Reference Architecture for SATF - Part 2.

A.3 Replication Packages

We provide a comprehensive replication package for each contribution presented in this thesis. The process of conducting the literature review, along with the development of a definition, taxonomy, and reference architecture for SATF, is documented and available at https://github.com/samirasilva/samira_phd_gssi/tree/main/taas_2023_supp. Additionally, all scripts and data used in the experiments with *GATE4BSN* and *AdapTA* can be accessed at the following links: https://github.com/samirasilva/Paper_JSS and https://github.com/samirasilva/AdapTA_Paper_AST, respectively.

Bibliography

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

[2] Jeffrey O Kephart and David M Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[3] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[4] Christian Murphy, Gail Kaiser, Ian Vo, and Matt Chu. Quality Assurance of Software Applications Using the In-Vivo Testing Approach. In *2009 International Conference on Software Testing Verification and Validation*, pages 111–120. IEEE, 2009.

[5] Luciano Baresi and Carlo Ghezzi. The Disappearing Boundary Between Development-Time and Run-Time. In *Proceedings of the Workshop on Future of Software Engineering Research (FoSER)*, 2010.

[6] Antonia Bertolino, Guglielmo De Angelis, Sampo Kellomaki, and Andrea Polini. Enhancing Service Federation Trustworthiness Through Online Testing. *Computer*, 45(1):66–72, 2012.

[7] Brian Fitzgerald and Klaas-Jan Stol. Continuous Software Engineering: A Roadmap and Agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[8] Luca Gazzola, Leonardo Mariani, Fabrizio Pastore, and Mauro Pezze. An Exploratory Study of Field Failures. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 67–77. IEEE, 2017.

[9] Kai-Yuan Cai, João W Cangussu, Raymond A DeCarlo, and Aditya P Mathur. An Overview of Software Cybernetics. In *Eleventh Annual International Workshop on Software Technology and Engineering Practice*, pages 77–86. IEEE, 2003.

[10] Kai-Yuan Cai. Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics. *Information and Software Technology*, 44(14): 841–855, 2002. ISSN 0950-5849.

[11] Mazeiar Salehie and Ladan Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):1–42, 2009.

[12] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A Survey on Engineering Approaches for Self-Adaptive Systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.

[13] Betty H Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer Berlin Heidelberg, 2009.

[14] Mariano Ceccato, Davide Corradini, Luca Gazzola, Fitsum Meshesha Kifetew, Leonardo Mariani, Matteo Orrù, and Paolo Tonella. A Framework for In-Vivo Testing of Mobile Applications. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 286–296. IEEE, 2020.

[15] Mark B Cooray, James H Hamlyn-Harris, and Robert G Merkel. Dynamic Test Reconfiguration for Composite Web Services. *IEEE Transactions on Services Computing*, 8(4):576–585, 2014.

[16] Antonio Guerriero, Raffaela Mirandola, Roberto Pietrantuono, and Stefano Russo. A Hybrid Framework for Web Services Reliability and Performance Assessment. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 185–192. IEEE, 2019.

[17] Majid Makki, Dimitri Van Landuyt, and Wouter Joosen. 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*, pages 70–73, 2016.

[18] Erik M Fredericks and Betty HC Cheng. 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*, pages 157–167. IEEE, 2015.

[19] Giancarlo Fortino, Raffaele Gravina, and Stefano Galzarano. *Body Sensor Networks*, pages 1–23. John Wiley & Sons, 2018.

[20] Mayda Alrige and Samir Chatterjee. Toward a Taxonomy of Wearable Technologies in Healthcare. In *New Horizons in Design Science: Broadening the Research Agenda: 10th International Conference, DESRIST 2015, Dublin, Ireland, May 20-22, 2015, Proceedings 10*, pages 496–504. Springer, 2015.

[21] Omer Aziz, Benny Lo, Rachel King, Ara Darzi, and Guang-Zhong Yang. Pervasive Body Sensor Network: An Approach to Monitoring the Post-Operative Surgical Patient. In *International Workshop on Wearable and Implantable Body Sensor Networks (BSN'06)*. IEEE, 2006.

[22] Tae-Gyu Lee and Seong-Hoon Lee. Design of Wearable Bio-Patch System Platform in Human Healthcare Environment. *Indian Journal of Science and Technology*, 8 (17), 2015.

[23] Nesime Tatbul, Mark Buller, Reed Hoyt, Steve Mullen, and Stan Zdonik. Confidence-Based Data Management for Personal Area Sensor Networks. In *1st International Workshop on Data Management for Sensor Networks: in Conjunction with VLDB*, pages 24–31, 2004.

[24] Luke Conroy, Ciarán Ó Conaire, Shirley Coyle, Graham Healy, Philip Kelly, Damien Connaghan, Noel E O'Connor, Alan F Smeaton, Brian Caulfield, and Paddy Nixon. Tennissense: A Multi-Sensory Approach to Performance Analysis in Tennis. In *27th International Society of Biomechanics in Sports Conference, 17-21,Limerick, Ireland*, 2009.

[25] Julien Pansiot, Benny Lo, and Guang-Zhong Yang. Swimming Stroke Kinematic Analysis with BSN. In *2010 International Conference on Body Sensor Networks*, pages 153–158. IEEE, 2010.

[26] Ryan Burchfield and S Venkatesan. A Framework for Golf Training Using Low-Cost Inertial Sensors. In *2010 International Conference on Body Sensor Networks*, pages 267–272. IEEE, 2010.

[27] Venet Osmani, Sasitharan Balasubramaniam, and Dmitri Botvich. Self-Organising Object Networks Using Context Zones for Distributed Activity Recognition. In *2nd International ICST Conference on Body Area Networks*, 2007.

[28] Aung Aung Phyo Wai, Foo Siang Fook, Maniyeri Jayachandran, Jit Biswas, Jer-En Lee, and Philip Yap. Implementation of Context-Aware Distributed Sensor Network System for Managing Incontinence Among Patients with Dementia. In *2010 International Conference on Body Sensor Networks*, pages 102–105. IEEE, 2010.

[29] Changhai Nie and Hareton Leung. A Survey of Combinatorial Testing. *ACM Computing Surveys (CSUR)*, 43(2):1–29, 2011.

[30] Eric Bernd Gil, Ricardo Caldas, Arthur Rodrigues, Gabriel Levi Gomes da Silva, Genaína Nunes Rodrigues, and Patrizio Pelliccione. Body Sensor Network: A Self-Adaptive System Exemplar in the Healthcare Domain. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 224–230. IEEE, 2021.

[31] Arthur Rodrigues, Ricardo Diniz Caldas, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. A Learning Approach to Enhance Assurances for Real-Time Self-Adaptive Systems. In *13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, pages 206–216, 2018.

[32] Samira Silva, Patrizio Pelliccione, and Antonia Bertolino. Self-Adaptive Testing in the Field. *ACM Transactions on Autonomous and Adaptive Systems*, 19(1):1–37, 2024.

[33] Samira Silva, Antonia Bertolino, and Patrizio Pelliccione. 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*, pages 58–69, 2022.

[34] Samira Silva. Reviewing and Proposing Approaches for Self-Adaptive Testing in the Field. In *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 13–16. IEEE, 2023.

[35] Samira Silva, Ricardo Caldas, Patrizio Pelliccione, and Antonia Bertolino. Different Approaches for Testing Body Sensor Network Applications. *Journal of Systems and Software*, page 112336, 2025.

[36] Samira Silva, Ricardo Caldas, Patrizio Pelliccione, and Antonia Bertolino. An Adaptive Testing Approach Based on Field Data. In *Proceedings of the 6th ACM/IEEE International Conference on Automation of Software Test (AST)*, 2025.

[37] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. John Wiley and Sons, New York, second edition, 2016. ISBN 9780471460817.

[38] James A Whittaker and Michael G Thomason. A Markov Chain Model for Statistical Software Testing. *IEEE Transactions on Software engineering*, 20(10):812–824, 1994.

[39] D Richard Kuhn, Raghu N Kacker, Yu Lei, et al. Practical Combinatorial Testing. *NIST Special Publication*, 800(142):142, 2010.

[40] D Richard Kuhn, Dolores R Wallace, and Albert M Gallo. Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, 2004.

[41] Linbin Yu, Yu Lei, Raghu N Kacker, and D Richard Kuhn. ACTS: A Combinatorial Test Generation Tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 370–375. IEEE, 2013.

[42] Emil Vassev, Mike Hinchey, Dharini Balasubramaniam, and Simon Dobson. An ASSL Approach to Handling Uncertainty in Self-Adaptive Systems. In *2011 IEEE 34th Software Engineering Workshop*, pages 11–18. IEEE, 2011.

[43] Autonomic Computing et al. An Architectural Blueprint for Autonomic Computing. *IBM White Paper*, 31(2006):1–6, 2006.

[44] Javier Cámara, David Garlan, Won Gu Kang, Wenxin Peng, and Bradley Schmerl. Uncertainty in Self-Adaptive Systems Categories, Management, and Perspectives. *Institute for Software Research. Carnegie Mellon University*, 2017.

[45] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless Sensor Network Survey. *Computer Networks*, 52(12):2292–2330, 2008.

[46] Raffaele Gravina and Giancarlo Fortino. Wearable Body Sensor Networks: State-of-the-Art and Research Directions. *IEEE Sensors Journal*, 21(11):12511–12522, 2020.

[47] Kyeonghye Guk, Gaon Han, Jaewoo Lim, Keunwon Jeong, Taejoon Kang, Eun-Kyung Lim, and Juyeon Jung. Evolution of Wearable Devices with Real-Time Disease Monitoring for Personalized Healthcare. *Nanomaterials*, 9(6):813, 2019.

[48] Xiaochen Lai, Quanli Liu, Xin Wei, Wei Wang, Guoqiao Zhou, and Guangyi Han. A Survey of Body Sensor Networks. *Sensors*, 13(5):5406–5447, 2013.

[49] Allan Collins, Diana Joseph, and Katerine Bielaczyc. Design Research: Theoretical and Methodological Issues. *Journal of the Learning Sciences*, 13(1):15–42, 2004.

[50] Zachary Munn, Micah DJ Peters, Cindy Stern, Catalin Tufanaru, Alexa McArthur, and Edoardo Aromataris. Systematic Review or Scoping Review? Guidance for Authors When Choosing Between a Systematic or Scoping Review Approach. *BMC Medical Research Methodology*, 18(1):1–7, 2018.

[51] Micah Peters, Christina Godfrey, Hanan Khalil, Patricia McInerney, Deborah Parker, and Cassia Baldini Soares. Guidance for Conducting Systematic Scoping Reviews. *International Journal of Evidence-Based Healthcare*, 13(3):141 – 146, 2015. ISSN 1744-1595.

[52] Heather L. Colquhoun, Danielle Levac, Kelly K. O'Brien, Sharon Straus, Andrea C. Tricco, Laure Perrier, Monika Kastner, and David Moher. Scoping Reviews: Time for Clarity in Definition, Methods, and Reporting. *Journal of Clinical Epidemiology*, 67(12):1291–1294, 2014. ISSN 0895-4356.

[53] Morena Barboni, Antonia Bertolino, and Guglielmo De Angelis. What We Talk About When We Talk About Software Test Flakiness. In Ana C. R. Paiva, Ana Rosa Cavalli, Paula Ventura Martins, and Ricardo Pérez-Castillo, editors,

Quality of Information and Communications Technology, pages 29–39, Cham, 2021. Springer International Publishing. ISBN 978-3-030-85347-1.

[54] Claes Wohlin. 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*, pages 1–10, 2014.

[55] Andreas Metzger, Eric Schmieders, Osama Sammodi, and Klaus Pohl. 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)*, pages 49–50. IEEE, 2012.

[56] Erik M Fredericks, Andres J Ramirez, and Betty HC Cheng. Towards Run-Time Testing of Dynamic Adaptive Systems. In *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 169–174. IEEE, 2013.

[57] Erik M Fredericks, Byron DeVries, and Betty HC Cheng. 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*, pages 17–26, 2014.

[58] Gerald Schermann, Dominik Schöni, Philipp Leitner, and Harald C Gall. Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-phase Live Testing Strategies. In *Proceedings of the 17th International Middleware Conference*, pages 1–14, 2016.

[59] Mariam Lahami, Moez Krichen, and Mohamed Jmaiel. Safe and Efficient Runtime Testing Framework Applied in Dynamic and Distributed Systems. *Science of Computer Programming*, 122:1–28, 2016.

[60] Henner Heck, Stefan Rudolph, Christian Gruhl, Arno Wacker, Jörg Hähner, Bernhard Sick, and Sven Tomforde. Towards Autonomous Self-Tests at Runtime. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 98–99. IEEE, 2016.

[61] Y Mohana Roopa and M Ramesh Babu. Self-Test Framework for Self-Adaptive Software Architecture. In *2017 International Conference of Electronics, Communication and Aerospace Technology (ICECA)*, volume 2, pages 669–674. IEEE, 2017.

[62] Joachim Hänsel and Holger Giese. 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)*, pages 9–12. IEEE, 2017.

[63] Lucas Leal, Andrea Ceccarelli, and Eliane Martins. The SAMBA Approach for Self-Adaptive Model-Based Online Testing of Services Orchestrations. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 495–500. IEEE, 2019.

[64] Roberto Pietrantuono, Stefano Russo, and Antonio Guerriero. Testing Microservice Architectures for Operational Reliability. *Software Testing, Verification and Reliability*, 30(2):e1725, 2020.

[65] Elaheh Habibi and Seyed-Hassan Mirian-Hosseinabadi. On-Demand Test as a Web Service Process (OTaaWS Process). In *2021 7th International Conference on Web Research (ICWR)*, pages 16–23. IEEE, 2021.

[66] Erick Barros dos Santos, Rossana MC Andrade, and Ismayle de Sousa Santos. Runtime Testing of Context-Aware Variability in Adaptive Systems. *Information and Software Technology*, 131:106482, 2021.

[67] Matteo Camilli, Antonio Guerriero, Andrea Janes, Barbara Russo, and Stefano Russo. Microservices Integrated Performance and Reliability Testing. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test*, pages 29–39, 2022.

[68] Luca Gazzola, Leonardo Mariani, Matteo Orrú, Mauro Pezze, and Martin Tappler. Testing Software in Production Environments with Data from the Field. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 58–69. IEEE, 2022.

[69] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[70] 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. Dynamically Adaptable Software Is All About Modeling Contextual Variability and Avoiding Failures. *IEEE Software*, 34(6):72–77, 2017.

[71] Alessio Bucaioni, Amleto Di Salle, Ludovico Iovino, Ivano Malavolta, and Patrizio Pelliccione. Reference Architectures Modelling and Compliance Checking. *Software and Systems Modeling*, 2022.

[72] Patrizio Pelliccione, Eric Knauss, S. Magnus Ågren, Rogardt Heldal, Carl Bergenhem, Alexey Vinel, and Oliver Brunnegård. Beyond Connected Cars: A Systems of Systems Perspective. *Science of Computer Programming*, 191:102414, 2020. ISSN 0167-6423.

[73] ISO/IEC/IEEE Systems and Software Engineering – Architecture Description, 2011.

[74] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[75] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23, 2015.

[76] Antonia Bertolino. Software Testing Research: Achievements, Challenges, Dreams. In *Future of Software Engineering (FOSE'07)*, pages 85–103. IEEE, 2007.

[77] Nianyu Li, Javier Cámara, David Garlan, Bradley Schmerl, and Zhi Jin. Hey! Preparing Humans to Do Tasks in Self-Adaptive Systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 48–58, 2021.

[78] Alberto González, Eric Piel, and Hans-Gerhard Gross. A Model for the Measurement of the Runtime Testability of Component-Based Systems. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 19–28. IEEE, 2009.

[79] Barbara Kitchenham and Pearl Brereton. A Systematic Review of Systematic Review Process Research in Software Engineering. *Information and Software Technology*, 2013.

[80] Mark Kasunic. Designing an Effective Survey. Technical report, DTIC Document, 2005.

[81] Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to Advanced Empirical Software Engineering*, volume 93. London: Springer, 2008.

[82] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Computer Science. Springer, 2012.

[83] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, Carlo A Furia, and Ziwei Huang. Evolution of Statistical Analysis in Empirical Software Engineering Research: Current State and Steps Forward. *Journal of Systems and Software*, 156:246–267, 2019.

[84] Andrea Arcuri and Lionel Briand. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[85] William H Kruskal and W Allen Wallis. Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[86] András Vargha and Harold D Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[87] Stuart Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[88] Peter J Rousseeuw. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[89] David L Davies and Donald W Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[90] Per Runeson and Martin Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14:131–164, 2009.

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

[92] Paul C Jorgensen. *Software Testing: A Craftsman's Approach*. Auerbach Publications, 2013.

[93] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[94] M Saeed, M Villarroel, AT Reisner, et al. Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II). *A Public-Access Intensive Care Unit Database*, 2011:39, 2011.

[95] Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting In-hospital Mortality of ICU Patients: The Physionet/Computing in Cardiology Challenge 2012. In *2012 Computing in Cardiology*, pages 245–248. IEEE, 2012.

[96] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, Physiotoolkit, and Physionet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*, 101(23): e215–e220, 2000.

[97] Adolphe Quetelet. *A Treatise on Man and the Development of His Faculties*. W. and R. Chambers, 1842.

[98] Gabriela Félix Solano, Ricardo Diniz Caldas, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. Taming Uncertainty in the Assurance Process of Self-Adaptive Systems: A Goal-Oriented Approach. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 89–99. IEEE, 2019.

[99] Marie Therese Cooney, Erkki Vartiainen, Tinna Laakitainen, Anne Juolevi, Alexandra Dudina, and Ian M Graham. Elevated Resting Heart Rate Is an Independent Risk Factor for Cardiovascular Disease in Healthy Men and Women. *American Heart Journal*, 159(4):612–619, 2010.

[100] Lingxia Song, Jiajin Li, Sen Yu, Yunjia Cai, Huan He, Jiayi Lun, Li Zheng, and Jufeng Ye. Body Mass Index Is Associated with Blood Pressure and Vital Capacity in Medical Students. *Lipids in Health and Disease*, 22(1):174, 2023.

[101] Onyesom Innocent, Oweh O ThankGod, Etumah O Sandra, and Ifie E Josiah. Correlation Between Body Mass Index and Blood Glucose Levels Among Some Nigerian Undergraduates. *Hoaj Biology*, 2(4):1–4, 2013.

[102] Suman Dua, Monika Bhuker, Pankhuri Sharma, Meenal Dhall, and Satwanti Kapoor. Body Mass Index Relates to Blood Pressure Among Adults. *North American Journal of Medical Sciences*, 6(2):89, 2014.

[103] Kerstyn C Zalesin, Barry A Franklin, Wendy M Miller, Eric D Peterson, and Peter A McCullough. Impact of Obesity on Cardiovascular Disease. *Endocrinology and Metabolism Clinics of North America*, 37(3):663–684, 2008.

[104] Hyuktae Kwon, Jae Moon Yun, Jin Ho Park, Be Long Cho, Kyungdo Han, Hee-Kyung Joh, Ki Young Son, and Su Hwan Cho. Incidence of Cardiovascular Disease and Mortality in Underweight Individuals. *Journal of Cachexia, Sarcopenia and Muscle*, 12(2):331–338, 2021.

[105] Qing Wu, Ming Yu, Jianfei Fu, and Meizhen Liu. Prevalence, Risk Factors, and Clinical Correlations of Underweight in Chinese Newly Diagnosed and Drug-naïve Patients with Parkinson's disease. *Neurological Sciences*, 42:1097–1102, 2021.

[106] Henry B Mann and Donald R Whitney. On a Test of Whether One of Two Random Variables Is Stochastically Larger Than the Other. *The Annals of Mathematical Statistics*, pages 50–60, 1947.

[107] Bento Rafael Siqueira, Fabiano Cutigi Ferrari, Marcel Akira Serikawa, Ricardo Menotti, and Valter Vieira de Camargo. Characterisation of Challenges for Testing of Adaptive Systems. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*, pages 1–10, 2016.

[108] Liliana Marie Prikler and Franz Wotawa. Challenges of Testing Self-Adaptive Systems. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B*, SPLC '22, page 224–228, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392068.

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

[110] Mariam Lahami and Moez Krichen. A Survey on Runtime Testing of Dynamically Adaptable and Distributed Systems. *Software Quality Journal*, pages 1–39, 2021.

[111] Bento R Siqueira, Fabiano C Ferrari, Kathiani E Souza, Daniel SM Santibáñez, and Valter V Camargo. 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)*, pages 284–293. IEEE, 2020.

[112] Gerson Barbosa, Érica Ferreira de Souza, Luciana Brasil Rebelo dos Santos, Marlon da Silva, Juliana Marino Balera, and Nandamudi Lankalapalli Vijaykumar. A Systematic Literature Review on Prioritizing Software Test Cases Using Markov Chains. *Information and Software Technology*, 147:106902, 2022.

[113] Kai-Yuan Cai. Optimal Test Profile in the Context of Software Cybernetics. In *2nd Asia-Pacific Conference on Quality Software*, pages 157–166. IEEE, 2001.

[114] Arunanshu Mahapatro and Pabitra Mohan Khilar. Fault Diagnosis in Body Sensor Networks. *International Journal of Computer Information Systems and Industrial Management Applications*, 5, 2012.

[115] Haibin Zhang, Jiajia Liu, Rong Li, and Hua Le. Fault Diagnosis of Body Sensor Networks Using Hidden Markov Model. *Peer-to-Peer Networking and Applications*, 10:1285–1298, 2017.

[116] Mark Utting, Alexander Pretschner, and Bruno Legeard. A Taxonomy of Model-Based Testing Approaches. *Software Testing, Verification and Reliability*, 22(5): 297–312, 2012.

[117] Stacy J Prowell. JUMBL: A Tool for Model-Based Statistical Testing. In *36th Annual Hawaii International Conference on System Sciences*, pages 9–pp. IEEE, 2003.

[118] Stacy J Prowell. Using Markov Chain Usage Models to Test Complex Systems. In *38th Annual Hawaii International Conference on System Sciences*, pages 318c–318c. IEEE, 2005.

[119] Kai-Yuan Cai. Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics. *Information and Software Technology*, 44(14): 841–855, 2002.

[120] Cu D Nguyen, Alessandro Marchetto, and Paolo Tonella. Combining Model-Based and Combinatorial Testing for Effective Test Case Generation. In *International Symposium on Software Testing and Analysis*, pages 100–110, 2012.

[121] Aiman Gannous and Anneliese Andrews. Integrating Safety Certification Into Model-Based Testing of Safety-Critical Systems. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 250–260. IEEE, 2019.

[122] Fitash Ul Haq, Donghwan Shin, and Lionel C Briand. Many-objective Reinforcement Learning for Online Testing of DNN-Enabled Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1814–1826. IEEE, 2023.

[123] Jubril Gbolahan Adigun, Tom Philip Huck, Matteo Camilli, and Michael Felderer. Risk-driven Online Testing and Test Case Diversity Analysis for ML-Enabled Critical Systems. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 344–354. IEEE, 2023.

[124] D Richard Kuhn, MS Raunak, and Raghu N Kacker. Ordered T-Way Combinations for Testing State-Based Systems. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 246–254. IEEE, 2023.

[125] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. Metamorphic Model-Based Testing of Autonomous Systems. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pages 35–41. IEEE, 2017.

[126] Arnaud Gotlieb, Dusica Marijan, and Helge Spieker. Testing Industrial Robotic Systems: A New Battlefield!. *Software Engineering for Robotics*, pages 109–137, 2021.