Runtime Verification and Field-based Testing for ROS-based Robotic Systems

Ricardo Caldas, Juan Antonio Piñera García, Matei Schiopu, Patrizio Pelliccione, Genaína Rodrigues, and Thorsten Berger

Abstract—Robotic systems are becoming pervasive and adopted in increasingly many domains, such as manufacturing, healthcare, and space exploration. To this end, engineering software has emerged as a crucial discipline for building maintainable and reusable robotic systems. The field of robotics software engineering research has received increasing attention, fostering autonomy as a fundamental goal. However, robotics developers are still challenged trying to achieve this goal given that simulation is not able to deliver solutions to realistically emulate real-world phenomena. Robots also need to operate in unpredictable and uncontrollable environments, which require safe and trustworthy self-adaptation capabilities implemented in software. Typical techniques to address the challenges are runtime verification, field-based testing, and mitigation techniques that enable fail-safe solutions. However, there is no clear guidance to architect ROS-based systems to enable and facilitate runtime verification and field-based testing. This paper aims to fill in this gap by providing guidelines that can help developers and quality assurance (QA) teams when developing, verifying or testing their robots in the field. These guidelines are carefully tailored to address the challenges and requirements of testing robotics systems in real-world scenarios. We conducted (i) a literature review on studies addressing runtime verification and field-based testing for robotic systems, (ii) mined ROS-based applications repositories, and (iii) validated the applicability, clarity, and usefulness via two questionnaires with 55 answers overall. We contribute 20 guidelines: 8 for developers and 12 for QA teams formulated for researchers and practitioners in robotic software engineering. Finally, we map our guidelines to open challenges thus far in runtime verification and field-based testing for ROS-based systems and, we outline promising research directions in the field.
Guidelines website and replication package: https://ros-rvft.github.io

Index Terms—Field-based Testing, Runtime Verification, Robotic Systems, Robot Operating System (ROS), Uncertainty, Guidelines.

1 Introduction

Robotics has become fundamental to societal advancement, featuring applications that range from micromachinery for medicine and healthcare to space exploration and navigation [1]. As robots become increasingly pervasive, the role of software in robotic systems is rising significantly [2], [3]. For instance, considering service robotics, studies have shown that a significant amount of resources is spent on building software rather than hardware [4]. The importance of software can be explained by the need to deploy robots with high autonomy and self-adaptation capabilities, allowing robots to operate under uncertainty [5], [6].

In the light of the growing recognition of software's pivotal role in robotics, software engineering is crucial for mission specification [7], [8], [9], [10], system architecture definition [11], [12], component design, implementation [13], [14], [15] as well as verification and validation [16], [17], [18]. Achieving autonomy, which minimizes human intervention, is a fundamental goal in robotics. However, despite the progress made, robotic systems still face challenges in adapting to real-world scenarios [19]. There is a need for instruments to deal with the openness and uncertainty of robots' operational environments – requiring (self-) adaptiveness capabilities and specific quality assurance techniques.

The toughest challenges in engineering software for robotics are: achieving robustness, costly and slow quality assurance provision processes, and lack of means to design dynamic (self-)adaptation [3], [20]. Furthermore, current simulation solutions are not able to emulate real-world phenomena in a sufficiently realistic manner, so developers prefer real-world experimentation to simulation [3], [20]. The robotics domain specificity is that (i) robots act in the real world, which introduces all sorts of unexpected error cases that should be caught and managed, (ii) the robots can change the environment itself via actions, and (iii) due to uncertainty and high variability, it is difficult to model real-world environments where robots operate, especially those that involve humans [3], [20].

In this context, it is impractical or even impossible to have robot's correct behavior guarantees at design time. To collect guarantees in robotics, quality assurance techniques must anticipate faulty scenarios, in spite of the complexities of runtime. Complexities arise from large input spaces, unpredictable corner cases, and lack of an oracle. Typical techniques to mitigate such complexities include using heuristics (e.g., test coverage [21]), or exploiting information available only at runtime [22], [23]). It is unclear, however, how to systematically adopt such techniques in a real-world setting.

ROS (Robot Operating System) is the de facto standard for robot application development and it has revolutionized robotics software engineering. However, there are still lim-

R. Caldas and M. Schiopu are with Chalmers University of Technology, Gothenburg, Sweden, - e-mail: {{ricardo.caldas,schiopu}@chalmers.se}
J. A. Piñera García and P. Pelliccione are with Gran Sasso Science Institute (GSSI), L'Aquila, Italy - e-mail: {{antonio.pinera,patrizio.pelliccione}@gssi.it}
G. Rodrigues is with University of Brasilia, Brasília, Brazil - e-mail: {genaina@unb.br}
T. Berger is with Ruhr University Bochum, Germany and Chalmers and University of Gothenburg, Sweden - e-mail: {thorsten.berger@rub.de}

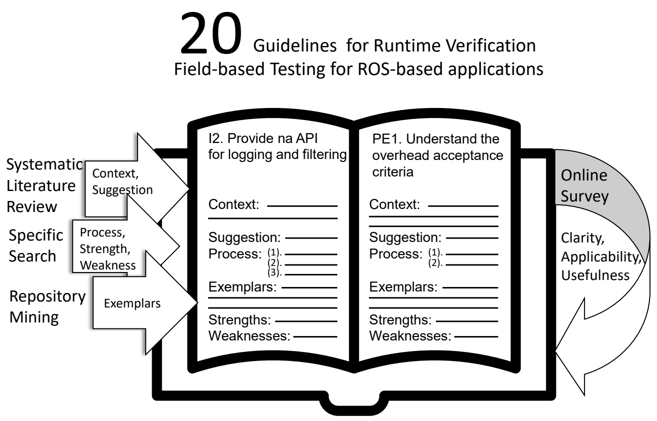
20 Guidelines for Runtime Verification
Field-based Testing for ROS-based applications
Systematic Literature Review
Context, Suggestion
Specific Search
Process, Strength, Weakness
Repository Mining
Exemplars
I2. Provide na API for logging and filtering
PE1. Understand the overhead acceptance criteria
Context:
Suggestion:
Process: (1). (2). (3).
Exemplars:
Strengths:
Weaknesses:
Online Survey
Clarity, Applicability, Usefulness

Fig. 1: Synthesis of 20 guidelines to runtime verification and field-based testing

ited advances in quality assurance methods for ROS-based systems. Alami et. al. [24] studied the effects of standards and practices of the ROS community contributing to quality assurance practices, highlighting what the influencers to successful adoption of quality assurance in the ROS community are. Moreover, tooling such as HAROS [25] promotes property-based testing of ROS applications by automatically deriving test cases from requirements specified in formal properties and the source code, which enables gaining confidence in ROS applications. However, there are no reported guidelines on available tooling and quality assurance techniques tackling real-world complexities for gaining confidence in ROS-based applications.

Our goal is to address this gap by establishing guidelines that can help developers and quality assurance (QA) teams when developing, verifying (runtime verification), or testing their robots in the field. In fact, despite recent efforts in providing guidance to architecting ROS-based systems [26], there is no clear guidance on how to design ROS-based systems to enable and facilitate quality assurance with verification and testing, especially at runtime. Our guidelines are tailored to ROS-based systems. They are defined not only for QA teams but also for developers. QA teams are responsible for the robotic systems' verification and validation. Developers are responsible for preparing software for field-based testing and runtime verification. Our research questions are:

RQ1. What guidelines should robotics developers follow to design ROS-based systems in order to enable and facilitate runtime verification and field-based testing?

RQ2. What guidelines should quality assurance teams follow to perform runtime verification and field-based testing for ROS-based applications?

Figure 1 illustrates our methodology. First, we systematically reviewed the literature on runtime verification and field-based testing of ROS-based systems. After we sketched the first drafts of the guidelines, we performed specific searches on search engines and scientific databases tailored to the specific guidelines to complement knowledge gaps. We also performed an in-depth analysis of repositories with open-source robotics applications that contain solutions to the challenges we identified. Thereafter, we synthesized actionable guidelines and enriched by concrete exemplars collected from the repositories through an online questionnaire. Finally, we validated the guidelines' applicability, clarity, and usefulness with robotics experts, including active researchers and practitioners.

Our study results in 20 guidelines: 8 guidelines for developers and 12 guidelines for QA teams. The catalogue of guidelines is specifically designed to instruct practitioners in the field-based testing of ROS-based systems, from both rigorous development and real-world testing and verification lenses. These guidelines are carefully crafted to address the challenges and requirements of testing and verification of robotic systems in real-world scenarios. Practitioners can identify best practices and recommendations to simplify, facilitate, and make more effective the verification and validation at runtime of their ROS-based robotic systems. We offer an overview of existing methods and tools for testing ROS-based systems, focusing on how they address the challenges of field-based testing and runtime verification.

Researchers can identify limitations, strengths, and gaps in the research landscape. We relate the guidelines with open challenges in runtime verification and field-based testing, and discuss promising research directions in the field.

2 Background

We introduce the Robot Operating System (ROS) (Sect. 2.1), runtime verification (Sect. 2.2), and field-based testing in robotics (Sect. 2.3).

2.1 Robotic Systems Development Based on ROS

Robot Operating System (ROS) is an ecosystem for developing robotic systems. Mainly, ROS comprises a middleware for interfacing fundamental robotic components (i.e., hardware, or software) with the ambition to eliminate re-inventing the wheel as a practice in robotics software engineering. ROS is used both in academia for teaching and research, and in industry, promoting commercial-friendly policies. ROS is prominently backed by a crescent and active open-source community. In the ROS ecosystem, the software is organized into basic building blocks called packages, that are meant to serve enough functionality to be useful. Over time, distributions became very large, surpassing 4000 packages in the ROS distribution Melodic Morenia. Furthermore, many open-source packages are not part of the official distributions.

A typical ROS-based system contains distributed resources interacting with each other. This network of resources is called the *ROS Computation Graph*, which is composed by four main elements:

- **Nodes** are the main resources in a ROS computation graph. They are processes that consume, process, and produce data. They communicate with each other via message passing. The other types of resources either hold shared data (parameters) or serve as message-passing channels (topics and services). Nodes should be specific and modular, rather than monolithic components. It is normal for a single robot to have a network of many nodes.
- **Topics** is the most common message-passing mechanism. It follows an asynchronous publish-subscribe

model with many-to-many connections. Publishers can send messages at any time, independently of the number of active subscribers. Subscribers, upon receiving a new message, are alerted through a callback function. Both publishers and subscribers utilize a message queue, the size of which is user-determined.

- **Services**, which are offered as the second message-passing option by ROS, facilitate synchronous one-to-one interactions through remote procedure calls. This model distinguishes between a server (node providing the service) and a client (node using the service).
- **Parameters** represent the last resource type, providing data sharing among nodes without relying on direct messaging or explicit communication. ROS maintains a local parameter server, a shared key-value store, which is accessible for reading and writing by any node or user, such as through a command line.

ROS is currently migrating to ROS 2, which introduces significant changes to managing the computation graph. This requires effort from developers to migrate their applications. As a result, some providers maintain their applications in previous ROS distributions while implementing new ones in ROS 2. For instance, MoveIt, a popular framework for robotics manipulation, maintains two versions. In our paper, we distinguish between ROS and ROS 2 when necessary. We believe that eventually ROS 2 will become the standard.

2.2 Runtime Verification in Robotics

Runtime Verification (or Runtime Monitoring[1]) comprises methods to analyze and check the dynamic behavior of computational systems [27], [28]. Runtime verification ensures that a property expressed with a formal language is not violated at runtime. It consists of a lightweight, yet rigorous, formal method that complements classical exhaustive verification techniques (e.g., model checking and theorem proving) with a more practical approach that analyses execution traces. At the price of limited execution coverage, RV can give precise information on the system's runtime behavior.

We follow the taxonomy of Falcone et al. [29] to discuss runtime verification in robotics. Runtime verification plays an important role in checking robotic software. Typically, robotic applications operate in uncertain environments, placing fundamental barriers to exhaustive verification. As such, runtime verification turns out to be a promising direction for gathering confidence in robotic systems.

In robotics, runtime verification may rely on reactive synthesis for generating monitors from specified system properties [30]. Monitors check properties at runtime. For example, monitors can be used to encode safety constraints that, upon violation, may trigger reactive behaviors that engage the system in a recovery mode [31]. Monitors check properties against traces of execution, i.e., a finite sequence of observations that represents the behavior of interest. The process of adding monitors to the system is called instrumentation. For instance, instrumentation may be needed to collect UAV data (e.g., GPS coordinates, attitude, and mission status), which is aggregated, analyzed, and persisted [32]. The aggregated data can either be checked on-the-fly (i.e., online monitoring) or in a post-mortem analysis (i.e., offline monitoring). The post-mortem analysis is an after-the-fact analysis technique that uses collected traces to validate the robotic system, e.g., Brunner et al. [33] proposes an architecture to robotics with an emphasis on post-mortem analysis by combining execution history logs (a.k.a. traces), Gantt charts, resource usage profiles, and task execution metrics and statistics.

2.3 Field-Based Testing in Robotics

Field-based testing[2] is a testing technique that uses (but is not constrained to) information from the field (a.k.a. the real world). We follow the definitions from Bertolino et al. [34]. As shown in Fig. 2, field-based testing includes *in-vivo* testing, i.e., tests that are executed in the production environment, and *in-vitro* testing, i.e., tests that are executed in the development environment but that are using data from the field.

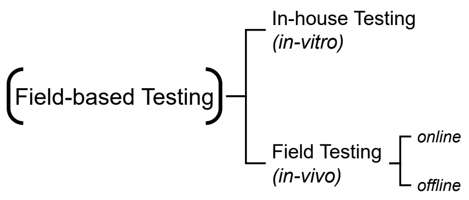

Fig. 2: A classification of Field-based testing strategies by Bertolino et al. [34]

In-house Testing (in-vitro) uses information from the field in tests conducted in the development environment. A typical scenario of in-vitro testing is when developers collect an abnormal behavior from runtime and want to better understand it by experimenting with such a scenario in simulation.

Field Testing (in-vivo) refers to tests that are executed in the production environment. Field tests may be divided into two sub-categories: *offline testing* and *online testing*. Field testing, as opposed to in-house testing, is desirable when the system itself operates in environments subject to uncertainty, whereas modeling and simulating the production environment is error-prone and may lead to false results.

Offline Field Testing is performed in the production environment on an instance of the system under test (SUT) that is different from the operational one. Then, the testing team may understand and validate the system by applying the desired inputs in the replica. This approach minimizes interference in the nominal operation of the system and may be more cost-effective than online field testing.

Online Field Testing is performed in the production environment. Online field testing is closer to reality, reducing the effect of uncertainty in the tests substantially. Other implications, such as security, privacy, and safety may emerge from performing online field testing.

1. There is no agreement in the community about which terminology is correct.

2. Please notice that Field Testing is different from Field-based Testing. The former is part of the latter.

3 METHODOLOGY

Our study contributes guidelines for development and quality assurance involved with engineering ROS-based applications. We focus on runtime verification and field-based testing to validate ROS-based applications. Our target audience is mainly practitioners and researchers interested in understanding the state-of-the-art of testing and RV in ROS in the light of growing recognition of software pivotal role in robotic software engineering. We aim to deliver concrete scientific artifacts that can be used by the ROS community [24].

We follow *design science* [35], [36], [37] to answer our research questions. We perform three research cycles, as depicted in Fig. 3. Each research cycle focuses on creating awareness about the problem domain and possible solutions, synthesizing a solution, and validating whether the solution mitigates the problem. In the three cycles, we incrementally synthesized a set of guidelines that answer our research questions.

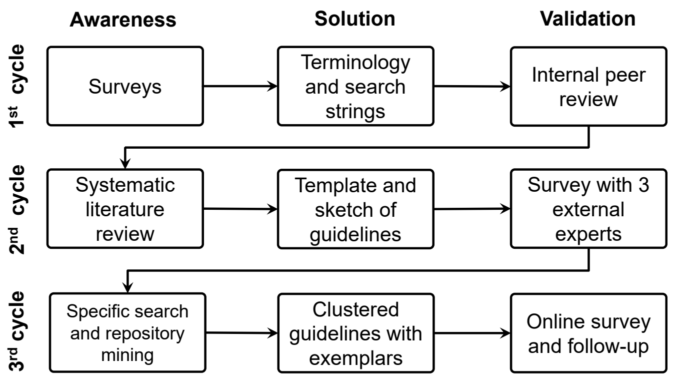
Awareness
Solution
Validation
1st cycle
Surveys
Terminology and search strings
Internal peer review
2nd cycle
Systematic literature review
Template and sketch of guidelines
Survey with 3 external experts
3rd cycle
Specific search and repository mining
Clustered guidelines with exemplars
Online survey and follow-up

Fig. 3: Overview of the activities to synthesizing guidelines according to design science

First Cycle. The first cycle is devoted to establishing awareness of the state-of-the-art of field-based testing and runtime verification in robotics through existing surveys. Through a literature review in Scopus, IEEE Xplore, and ACM DL, we collected surveys that discuss runtime verification and field-based testing. We, then, reviewed the surveys to define the terminology to delimit the scope of the study and the search strings that we would use in the next iteration. These surveys also played a role in the identification of sketches of guidelines (which should be considered as input for the second cycle) and in the identification of challenges, which are discussed in Sect. 6. The terminology and the search strings were internally validated through peer reviews and discussions with co-authors to consolidate the terminology and scope.

Second Cycle. We, then, used the search strings defined in the previous cycle to perform a systematic literature review. Then, we defined a template for describing the guidelines and a first draft of the guidelines. More specifically, the data extracted from surveyed papers has been used to identify an initial set of guidelines. In this step, we identified 8 guidelines, 4 concerning the developers' activities and 4 concerning the quality assurance team activities (as reported in the replication package [38]). As presented later in the paper, the final set of guidelines is much larger (20 guidelines) since some of this initial set of guidelines have been split into more guidelines and new ones have been added. As a preliminary validation step, we presented a subset of the guidelines' sketches at the Robotics Software Engineering workshop[3]. We received feedback on the guidelines from three experts in robotics through a questionnaire and subsequent discussion.

Third Cycle. The third cycle was devoted to consolidating the set of guidelines and validating them with experts from industry and academia. The comments from the expert evaluation in cycle 2 suggested to further extend the set of guidelines due to potential missing guidelines and to complement the guidelines with concrete examples, e.g., *"Can you provide code examples?"* or *"Without examples I wonder how such a specification pattern could look like."*. Thus, guided by such comments, we decided to perform a specific search and to mine online repositories. The rationale is that the SLR should be complemented by specific searches and repositories mining focused on finding additional papers, information, and concrete examples in a specific topic or argument. For example, in the initial set of 8 guidelines, we had only one guideline concerning the specification of properties. The feedback from one of the three external experts highlighted that there was the need of more precision and standardized terms. Then, thanks to specific searches and mining of repositories for property specification in ROS-based systems, the single guideline on the specification of properties has been refined into three guidelines, one focused on logic-based languages, one on domain specific languages and the third on scenario-based specification (SDB1, SBD2, and SBD3). It is important to highlight that these specific searches cannot be substituted by a refined SLR including more terms (i.e., re-performing the second cycle of the methodology), since the granularity of specific searches is different from the granularity of the SLR.

To perform the specific searches and repositories mining, we used the terminology from the second cycle and the feedback from the experts. Information on the specific searches we made is available in the replication package [38]; specifically, within the protocol for repositories mining, we provide tables with the keywords used for the mining activity performed for each guideline. We searched for repositories mentioned in the documents in complement to repositories that touched upon the theme of each guideline sketch. In the solution phase, we then developed the final version of the guidelines, provided exemplars for each of them, and clustered the guidelines as they are presented in this paper. Finally, we validated the final set of guidelines with practitioners and researchers. We designed and distributed an online questionnaire to experts from industry and academia, through the ROS community forum (namely ROS Discourse), LinkedIn, X (Twitter), and to experts we met at robotics and software engineering conferences. Before releasing the questionnaire to the wider community, we performed a pre-study with an experienced roboticist and active member of the ROS community and two members with widely recognized contributions to robotic software engineering from academia. The results from the third cycle confirmed the relevance of the guidelines and helped us to fine-tune them. This will be further detailed in Sect. 3.3.

3. https://rsemeeting.github.io/rse2022/

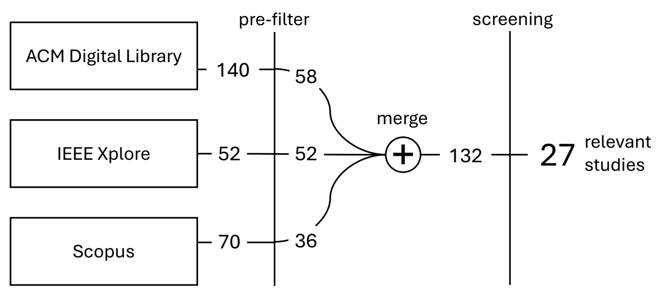

Fig. 4: Literature review and quantities of studies

In the reminder, we detail the steps that require further explanation: the Systematic Literature Review of the second cycle is described in Sect. 3.1; the specific search and repository mining step is described in Sect. 3.2, and the online survey and follow-up step of the third cycle is described in Sect. 3.3. All generated artifacts and information are available in the replication package [38].

3.1 Systematic Literature Review

Given the lack of comprehensive scientific reports on runtime verification and field-based testing in robotics, we used a systematic literature review to identify the available and relevant research [39]. To ensure a clear and thorough methodology, we established a protocol for our literature review, outlining our research goals, the process to be followed, data extraction, and measures to mitigate threats to the validity of our results. Detailed information is provided in the replication package [38].

Research Goal. Characterize runtime verification (RV) and field-based testing (FT) for building confidence in ROS-based applications from the researchers' perspective.

Process. Our search process started by defining relevant research questions, which informed the creation of a search string to be used in automatic search engines (i.e., IEEE Xplore, ACM Digital Library, and Scopus). The search string was composed of terms elicited from previous studies on runtime verification tools [29] (lines 3-4) and field-based testing [34] (lines 6-8).

```
(ros  OR robot* operating system) AND
(runtime verification  OR run−time verification OR
 runtime assurance OR run−time assurance OR
 online verification   on−line verification   OR
 runtime monitoring OR run−time monitoring OR
 runtime testing OR run−time testing OR
 online testing  OR on−line testing  OR
 field −based testing OR field testing  OR in−vivo testing)
```

As shown in Figure 4, the search resulted in 262 papers. We applied pre-filters in the search engine to only include papers from Computer Science and Engineering and similar areas, reducing the number of studies to 132 merged and unique papers. Then, we used inclusion and exclusion criteria for screening for relevance, resulting in 27 relevant studies.

The inclusion criterion tailors the study to ROS, independently of the version or distribution, and the exclusion criteria determine that studies should not be short papers, secondary studies, or duplicated. We also exclude papers that are not from Computer Science or Engineering.

Data Extraction. The screening phase and the research goals were the basis for defining a classification scheme as a set of categories that should be used for extracting data from the resulting studies. The categories used in the scheme define the meta-information of the publications, the nature of the assurance technique employed (i.e. runtime verification or field-based testing), a classification of the applied techniques according to the taxonomies for runtime verification [29] and field-based testing [34], the software quality attributes assessed in the studies, use cases, motivation, and rationale. Finally, if the study needed to be more precise about how they used RV or FT, we would query about the used technique, whether it is static or dynamic, and whether it was performed offline or online. With the classification scheme and the studies in hand, data extraction is a matter of carefully reading and collecting information from the studies. The extraction activity resulted in 9 studies reporting on field-based testing, 16 studies reporting on runtime verification, 1 study discussing both, and 2 studies discussing none of them, but, instead, providing valuable insights into runtime verification and field-based testing in ROS.

3.2 Specific search and repository mining

Guided by the comments from experts (2nd cycle), we (three authors of this paper) searched for additional information to complement the guidelines. As defined in the description of the third cycle, we focused on finding additional papers and concrete code examples to make guidelines more useful. We performed both a specific search[4] and mined online repositories. First, for the specific search, we manually selected results searched from Google Scholar and the ACM digital library. Then, we followed the mining approach by Malavolta [26] to collect new online repositories. We collected and cloned 1088 repositories, searched for patterns in the files within the repositories, performed a manual context analysis. The results went through a peer review with the other authors of our paper. The peer review aimed at a refinement of the search, or at potentially finding new exemplars. Further information and the scripts described in this section are available in our online replication package [38].

To collect and clone the 1088 repositories we relied on rosmap,[5] a tool designed for dependency mapping, which allowed the gathering of an exhaustive list of all public ROS projects available on Github, as well as Bitbucket and Gitlab. Then, to filter out unwanted repositories and download respositories to a local server, we modified the Python script from Malavolta's replication package.[6]

- *explorer.py* removes repositories that are duplicate, forked, lower than 100 commits, have a low star count (≤ 2), and are demo projects; such repositories have a low relevance to our study, which looks into exemplars to complement the guidelines.
- *cloner.py* downloads the repositories to a local server. The local setup allows for quick targeted searches inside the source files of mature projects, with a high degree of search-term flexibility.

4. The search was performed in December 2023
5. https://github.com/jr-robotics/rosmap
6. https://github.com/S2-group/jss-2021-replication-package

We curated the downloaded repositories by manual inspection. We used the UNIX 'grep' utility to efficiently scan large numbers of files. Particularly for guidelines with no straightforward exemplars, we searched for exemplars by looking up keywords relevant to each of the guidelines and examining code comments in the files (e.g., LLVM, LibFuzzer, which we found while looking for fuzzing mechanisms, used by PX4 Autopilot, a large open source project [40]). Moreover, for guidelines that already contained exemplars from the literature, we performed dependency analysis by looking into source files of the other repositories, specifically in their *import* statements. Whenever we found matching files, either by keyword search or dependency analysis, we analyzed peripheral code which helped us to discard irrelevant exemplars. Finally, the remaining repositories went through peer-review sessions in which other authors of this paper determined whether the repositories were viable exemplars discovered for the corresponding guidelines.

3.3 Online Survey and Follow-up

Although our guidelines are grounded on data, provenient from the scientific literature and mined repositories, we made an additional validation step with experts in robotics particularly in runtime verification, or field-based testing to validate the adherence of the guidelines with the target groups. The validation aims at collecting the practitioner's sentiment on whether each guideline is useful, clear, and applicable. The instrument of this validation was online questionnaires with follow-up emails. Our validation strategy comprised (i) recruitment, (ii) questionnaire design, (iii) results analysis, and (iv) follow-ups and reporting data.

3.3.1 Recruitment: Collecting Names and Contacts of Robotics Experts

The guidelines intend to contribute practical suggestions on how (i) developers should design robotic systems to enable runtime verification and field-based testing, and how (ii) quality assurance teams can verify and test ROS-based applications. We targeted practitioners, i.e., developers and QA teams with experience in robotics (ROS specifically), runtime verification [29], field-based testing [34], and researchers with experience in robotics. We used three data sources to collect names and contacts: authorship and email addresses in the papers and repositories used as sources of information for the synthesis of the guidelines, advertisement in social media and forums, and renowned experts in software robotic research.

Overall, our survey received 55 answers to the questionnaire. We manually collected names and emails from papers, which resulted in 306 contacts. Our mining algorithm provided 772 emails from the authors from mined repositories. In addition, we shared the questionnaire in the ROS discourse forum.[7] By March of 2024, the post was accessed by 1.6k ROS discourse users. Moreover, the link to the questionnaire targeting developers was accessed 60 times and the link to the questionnaire targeting quality assurance teams was accessed 8 times. OpenRobotics also shared the discourse post on LinkedIn[8] and X (Twitter).[9]

We also participated in conferences, meetings, workshops, and summer schools during the study. We invited a list of 67 robotics experts to participate in this survey. To avoid bias, before inviting candidate respondents, we did not discuss the subject matter of any of the guidelines but discussed the idea of synthesizing guidelines to support runtime verification and field-based testing of ROS-based applications.

Our recruitment campaign started on November 29th 2023 and ended on the 5th of January of 2024, when we were not receiving any more answers from the candidate respondents.

3.3.2 Questionnaire Design

The survey aims to capture the target group's opinion and experience in runtime verification and field-based testing ROS-based systems. Since we decided to separate the guidelines targeting to two different audiences, developers and QA teams, we also decided to use two separate questionnaires. First, the introduction sets the context and gives instructions on how to answer the questions. Then, the main part for each guideline consisted of three questions eliciting usefulness, clarity, and applicability. We also included open-ended questions that enabled respondents to express their opinions about one or more guidelines. Third, the profiling section asked about the respondents' background and previous experience with software engineering for robotics, especially ROS-based applications.

We piloted the questionnaire design with a member of the ROS community, who was responsible for maintaining the ROS Discourse Forum,[10] and with an additional expert from academia. In addition, we contacted a professor with over ten years of research on robotics software engineering, and a postdoc that has recognizable contributions to the robotics software engineering field. The main comments from the pre-study highlighted that (i) the term "field-based testing" was not clear, (ii) some guidelines were not guidelines, and (iii) the guidelines were written too formally, with *academic jargon*. We addressed their comments and rectified the questionnaire with the same member who assisted in publishing the post.

3.3.3 Result Analysis

We mainly relied on quantitative data analysis combined with statistical tests to help us build confidence in our results and steer follow-up analyses. To analyze the Likert questions we used (i) Likert plot visualizations of responses and (ii) quantitative statistical analyses, i.e., Boxplot, significance test. We used the analysis to follow-up with the respondents to better understand variance in the responses on specific guidelines. To perform the statistical analyses (ii), we assume that the Likert items can be interpreted as intervals, and we use consistent confidence levels ($\alpha = 0.05$) and the Cohen's r parameters to analyse the effect size.

7. https://discourse.ros.org/t/we-need-participants-for-a-survey-on-field-based-testing-of-ros-applications/34879

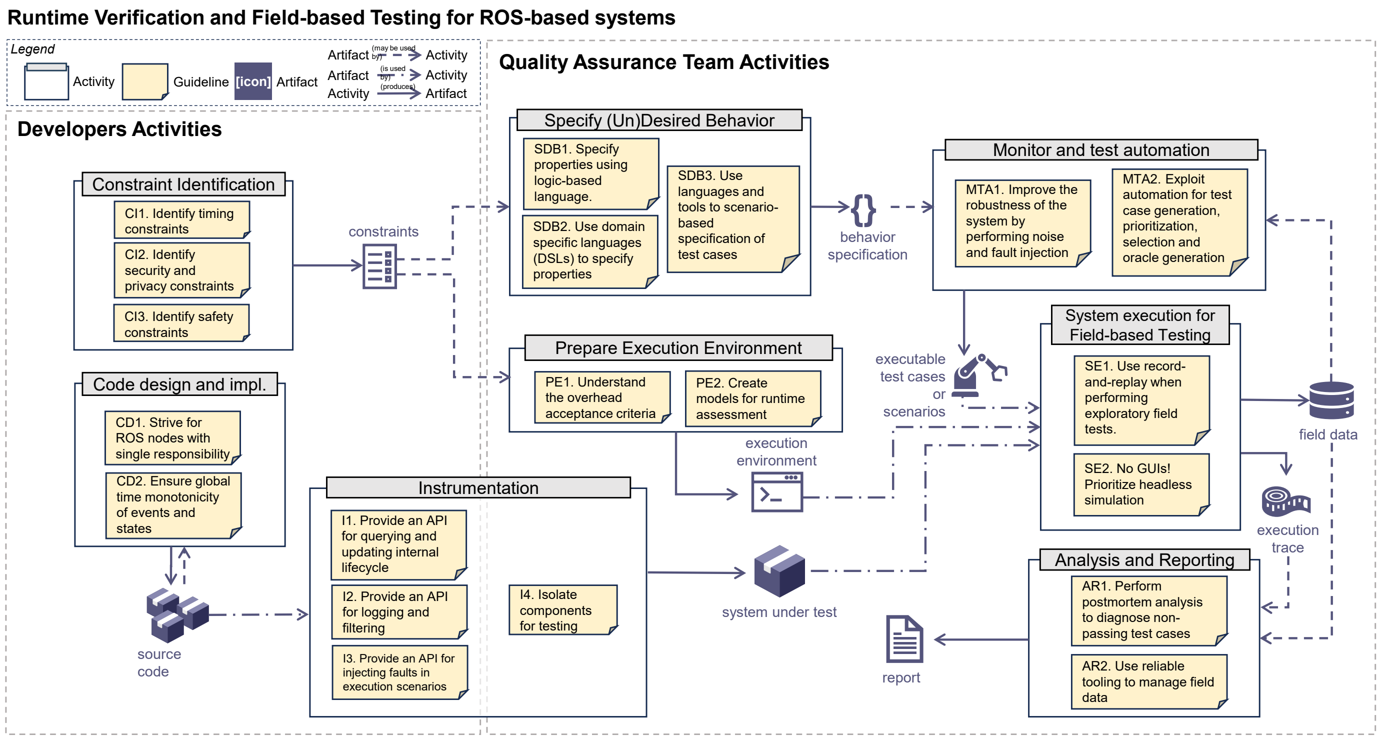
Runtime Verification and Field-based Testing for ROS-based systems
Legend
Activity
Guideline
[icon] Artifact
Artifact (may be used by) Activity
Artifact (is used by) Activity
Activity (produces) Artifact
Developers Activities
Quality Assurance Team Activities
Constraint Identification
CI1. Identify timing constraints
CI2. Identify security and privacy constraints
CI3. Identify safety constraints
constraints
Code design and impl.
CD1. Strive for ROS nodes with single responsibility
CD2. Ensure global time monotonicity of events and states
source code
Instrumentation
I1. Provide an API for querying and updating internal lifecycle
I2. Provide an API for logging and filtering
I3. Provide an API for injecting faults in execution scenarios
I4. Isolate components for testing
system under test
Specify (Un)Desired Behavior
SDB1. Specify properties using logic-based language.
SDB2. Use domain specific languages (DSLs) to specify properties
SDB3. Use languages and tools to scenario-based specification of test cases
behavior specification
Monitor and test automation
MTA1. Improve the robustness of the system by performing noise and fault injection
MTA2. Exploit automation for test case generation, prioritization, selection and oracle generation
Prepare Execution Environment
PE1. Understand the overhead acceptance criteria
PE2. Create models for runtime assessment
execution environment
executable test cases or scenarios
System execution for Field-based Testing
SE1. Use record-and-replay when performing exploratory field tests.
SE2. No GUIs! Prioritize headless simulation
field data
execution trace
Analysis and Reporting
AR1. Perform postmortem analysis to diagnose non-passing test cases
AR2. Use reliable tooling to manage field data
report

Fig. 5: Overview of the guidelines organized according to the engineering activities of developers and the QA teams when engineering ROS-based software and who strive to establish field-based testing and runtime verification

4 Guidelines

We now describe the guidelines we synthesized in our study. Sect. 4.1 provides an overview of the guidelines for preparing and performing runtime verification and field-based testing for ROS-based systems. Then, before describing the guidelines in detail, Sect. 4.2 describes the template used to specify guidelines. Finally, Sect. 4.3 and 4.4 highlight two representative guidelines. The complete list of guidelines can be found on our dedicated website [38].

4.1 Guidelines Overview

Figure 5 provides an overview of the guidelines. The guidelines are organized into two groups: (i) guidelines concerning activities that are performed to prepare ROS-based systems for field-based testing and to enable runtime verification, and (ii) guidelines concerning the actual runtime verification and field-based testing of ROS-based systems. The first set of guidelines is mostly defined for developers, and the second set is for quality assurance (QA) teams. Then, guidelines are organized into eight activities, each focused on specific activities:

- *Constraint Identification (CI)* guidelines concerns the elicitation of non-functional requirements that may be critical or non-negotiable to the system under design and test. Preparing or running test scenarios to attest correct behavior of the system should not violate the identified constraints. These constraints might be useful for the *Specify (Un)Desired behavior* SDB group of guidelines (explained shortly); however, the specification of properties or test cases can use other sources of information, e.g., a requirement document.
- *Code Design and Implementation (CD)* includes guidelines that may facilitate testing and verification by modifying the source code of the robotics software under scrutiny [41], [42].
- *Instrumentation for runtime verification and field-based testing(I)* includes guidelines to support either the developers or the testers in instrumenting the code resulting in a system ready to test. Instrumentation in this context refers to modifying the source code to expose internal variables to monitors or oracles [29], [43], [28].
- *Specify (Un)Desired behavior (SDB)* takes into consideration the elicited constraints and provides the means to abstract the system behavior in terms of states and events (a.k.a. state changes) and how specification languages can be used to describe properties of a set of such states or events [29], [28].
- *Monitor and Test Automation (MTA)* concerns guidelines to automate the generation of monitors or test, including the synthesis of test cases or test scenarios that may accept or reject an observed execution trace [29], [43], [28], [34].
- *Prepare Execution Environment (PE)* concerns guidelines to prepare the environment by setting up supporting devices such as stubs and mocks to runtime verification and field-based testing [41], [42], [44]. The environment preparation may take into considera-

8. https://www.linkedin.com/feed/update/urn:li:activity:7138595420783484929/
9. https://twitter.com/OpenRoboticsOrg/status/1732832183507951929?t=_bBFf7RptL7QM3wrEQ0KnA&s=19
10. https://discourse.ros.org/

tion testing constraints previously elicited to avoid damages to the test or lead to skewed results.

- *System Execution for Field-based Testing (SE)* includes guidelines concerning the activity of running one or more test cases or test scenarios in a given execution environment to exercise a system under test, which results in field data [29], [34], [43].
- *Analysis and Reporting (AR)* gathers the field data generated during test execution and derives evidence, conclusions, and finally a report on the observed behavior [29], [34], [43]. In the case of inconclusive results, the report may be used by the developers to further refine the constraints or code design. Otherwise, the report may be used as a collection of assurance arguments to leverage confidence in the tested ROS-based system.

It is important to highlight that the overview offered in Fig. 5 does not define a new development process for ROS-based systems. The guidelines are agnostic to the development process used. Moreover, the guidelines are independent of each other, and each of them can be used in isolation. Also, it is the responsibility of the developers or the QA team to select and use those guidelines that are more beneficial for the system they are developing and/or analysing. However, as shown Fig. 5 groups of guidelines are connected by arrows, since the artifact produced by some guidelines might be useful for other guidelines. Specifically, the relationships among guidelines are of three different types: (i) an artifact *might be used* by an activity, (ii) an artifact *is used by* an activity, and (iii) an activity *produces* an artifact. In the legend of the figure, we explain the graphical representation used for these three types of relationships.

The guidelines are detailed in Sect. 4.3 and 4.4 after the description of the template used to document them (in Sect. 4.2). Our dedicated website [38] contains the full list of guidelines for enabling and facilitating runtime verification and field-based testing of ROS-based systems.

4.2 Template for Guideline Specification

In the following, we describe the fields that compose the template. The template is used in Sect. 4.3 and 4.4, as well as, to document all guidelines in our dedicated website [38].

ID: Identifier used to facilitate tracing guidelines between groups.
Title: Title summarizes an action that practitioners should follow to mitigate or avoid a recurring problem.
Context (WHEN): The Context is a paragraph placing the guideline among a known set of conditions. This paragraph should delimit the scope in which the guideline is applicable. It should also introduce the conceptual terminology used in the guideline, which is defined by the conditions under which the guideline is valid.
Reason (WHY): Reason introduces the recurring problem faced by practitioners. It intends to leverage the relevance of the guideline to practitioners.
Suggestion (WHAT): Suggestion is a sentence or two introducing WHAT should the practitioners do to mitigate the recurring problem.
Process (HOW): Process is a paragraph that carefully guides the practitioners through HOW they can practice the guideline. In this paragraph, there should be references to tools that may help, concrete examples, or references to precise explanations of researchers or practitioners who have done something similar.
Exemplars: Exemplars are concrete descriptions of papers/artifacts that follow the guidelines. An exemplar can be generic such as an artifact or rather specific such as a model problem in testing ROS-based systems in the field.
Strengths: Strengths is a list of benefits that the practitioners should consider when applying the guideline.
Weaknesses: Weaknesses is a list of technological/theoretical barriers that may slow the actual implementation of the guideline, either by undesired side-effects or scenarios in which applying the guideline might not lead to the desired effect.

4.3 Guidelines for Development to Support Runtime Verification and Field-Based Testing

We synthesized the guidelines in Tab. 1 relevant for enabling and facilitating runtime verification and field-based testing through *development* activities, except I4 that concerns QA activities.The guidelines are organized in groups, as explained in Sect. 4.1 and summarized in Fig. 5. For each guideline we summarize the name of the guideline, and according to the main activities described in the research methodology (Sect. 3), we provide the links to (i) the relevant initial surveys (column Initial), (ii) the papers identified during the systematic literature review (column Lit. Review), (iii) papers identified during the specific search phase (column Specific Search), and (iv) exemplars identified with the repositories mining (column exemplars). The group of Constraint Identification (CI) guidelines contains three guidelines, CI1, CI2, and CI3, devoted to identifying constraints of different nature, i.e., timing constraints, security and privacy constraints, and safety constraints. The group of Code Design and Implementation (CD) guidelines contains two guidelines, CD1 recommending to design ROS nodes with single responsibility and CD2 guiding towards ensuring global time monotonicity of events and states. The last group contains four Instrumentation (I) guidelines. I1, I2, and I3 focus on providing APIs for querying and updating an internal lifecycle (I1), for logging and filtering (I2), and for injecting faults in execution scenarios (I3). The last one (I4) focuses on isolating components for testing. We elaborate on the guideline I2 (provision of an API for logging and filtering) in depth because we believe I2 can help understand each field of the template as well as the group of guidelines to facilitating field-based testing and runtime verification since it contains several exemplars.

Guideline I2: Provide an API for Logging and Filtering

The guideline I2 recommends developers provide an API for inline or outline logging and filtering to gather relevant data from runtime or the field given that the quality assurance team might not have access to the system under test. Using I2 renders effortless observability of inner states and events, but warns that the misuse of I2 may turn into performance issues, adding noise to data, false positives, or open security breaches.

TABLE 1: Guidelines to facilitating field-based testing and runtime verification of ROS-based systems

	Guideline	Initial	Lit.Review	Specific Search	Exemplars
Constraint Identification	CI1. Identify timing constraints	[34], [45]	[46], [47]	[48], [49]	Autoware_Perf[48]
	CI2. Identify security and privacy constraints	[26], [29] [34], [41]	[50], [51]	[52], [53]	ROSRV[51] SROS[52]
	CI3. Identify safety constraints	[54], [34]	[51]	[55], [56], [19]	ROMoSu[56] Kobuki
Code design and impl.	CD1. Strive for ROS nodes with single responsibility	[26], [34]	[57], [58]	[59], [60], [61] [62], [63]	bsn[63] skiros2[61] hmrs_mission_control[12]
	CD2. Ensure global time monotonicity of events and states	[26]	[64]	[65], [66], [67]	ros2-picas [66] mavros
Instrumentation	I1. Provide an API for querying and updating internal lifecycle	[26], [29], [34]	[68], [58]	[69], [70], [71]	Micro-ROS[69] system_modes[70] rclc_lifecycle [71] lifecycle_service_client [70]
	I2. Provide an API for logging and filtering	[54], [29], [34]	[72]	[73], [74]	rosout cloudwatchlogs-ros2 ROSMonitoring[72] fault-tolerant-ros-master black-box[74]
	I3. Provide an API for injecting faults in execution scenarios	[54], [29], [34]	[58]	[75], [76], [77]	imfit[78] ros1_fuzzer ros2_fuzzer
	I4. Isolate components for testing	[29], [34]	[51], [72]	[79], [80]	ROSRV [51] ROSMonitoring[72]

ID I2
Title Provide an API for Logging and Filtering
Context (WHEN) Dynamically gathering information is a fundamental step for gaining confidence in ROS-based systems. Logging (and playback) is, in fact, one of the most used techniques for testing ROS-based systems [54]. Often named *monitoring* [29] or *logging* [34], the process of recording textual or numerical information about events of interest may be a valuable input to the testing team. With such data at hand, the testing team will process the data and transform it into useful information to challenge their hypotheses about how the system should work.
Reason (WHY) Logging important events depends on instrumenting the code (with 'hooks') that enables the monitoring or logging. It is unrealistic to assume that the testing team will have access to the source code or that the testing team knows what events to log or how to do so.
Suggestion (WHAT) The development team should provide an API for logging and filtering data to enable access to valuable runtime data used for runtime verification and field-based testing. The standard for logging and filtering is rosbag (wiki: rosbag). Though, in addition, AWS CloudWatch (git: aws-robotics/cloudwatchlogs-ros2) collects data from the rosout topic and provides a filter for eliminating noise from the logged events. Another example is the Robotic Black Box (git: ropod-project/black-box) which allows for listening to data traffic from distinct sources and logging the messages using MongoDB.
Process (HOW) We divide, according to Falcone et al. [29], techniques for gathering information in two: inline, and outline. Inline logging and filtering stand for techniques that ask for actually inserting snippets of code in the system under scrutiny as a means to provide an API for logging, e.g., [134]. Outline logging and filtering stands for techniques that enable an external means to gather and filter information that does not require changing the source code, e.g., [72], [73], [74]. Developers may choose one or another given their domain of application.
Exemplars Exemplar implementations for providing an API for logging and filtering may be inline or outline.
Inline Logging and Filtering. ROS, by standard, contains a system-wide logging mechanism, namely ROS logging (wiki: logging). ROS logging works with macros for instrumenting the ROS nodes with information hooks. In the background, the macros send messages with the information to be logged through a standard topic called rosout. On the other side of the topic, an extra node, within the roscore package, persists the data in a textual format (git: noetic-devel/ros/../rosout). Developers can use the macros to log information that might be used for testing in a later stage, given the application-specific requirements. For instance, the Amazon AWS service for robotics provides AWS CloudWatch Logs (git: aws-robotics/cloudwatchlogs-ros2) interfaces directly with rosout to monitor applications using the standard ROS logging interface. In addition, the standard library provides logging macros with embedded filtering capabilities, which enables eliminating noise from the logged events and can render a useful tool for testers. ROS Rescue [134] is another example of inline logging. The tool aims to solve the ROSMaster problem as a single point of failure. The authors approach check-pointing and restoring state by logging changes in the metadata stored in the master node. Such metadata contains URIs from various nodes, port numbers, published or subscribed topics, services, and parameters from the parameter server. Kaveti et al. create an API for the ROS master node (git: master_api.py) using the official logging library from Pythonto persist metadata in YAML format. Their technique opens space for further inspection of ROS applications without access to the source code.

TABLE 2: Guidelines for performing runtime verification and field-based testing on ROS-based applications

	Guideline	Initial	Lit.Review	Specific search	Exemplars
Prepare Execution Environment	PE1. Understand the overhead acceptance criteria	[26], [29], [34]	[81], [72]	[82], [48], [83], [84]	ROS-Immunity [81] ROSMonitoring [72] ros2_tracing[80] RTF4ADS [83] and Secure Channels in ROS [84]
	PE2. Create models for runtime assessment	[34]	[85], [47], [86] [87], [88], [89]	[90], [91], [92], [93] [94], [95], [96], [97] [98]	ROS_Tecnomatix [93] cpsaml[94] dtron + adapter [95], [96] ros-model [97], [98]
Specify (Un)-Desired Behavior	SDB1. Define properties using a logic-based language	[41]	[25], [50], [99]	[100], [9], [101], [102]	MTL [25], [50] rtamt4ros [103] LTL [101] Past-Time LTL [102]
	SDB2. Use Domain Specific Languages (DSLs) to specify properties	[41]	[51], [72], [56] [104], [105]	[106]	ROSRV [51] ROSMonitoring [72] RuBaSS [106]
	SDB3. Use languages and tools to scenario-based specification of test cases	[54], [41], [34]	[107], [108], [109]	[110], [111], [112]	Geoscenario [110] SCENIC [111] PerceMon [107] pedsim_ros [109]
Monitor and Test Automation	MTA1. Improve the robustness of the system by performing noise and fault injection	[34]	[58], [113], [114]	[78], [115], [116]	ros1_fuzzer ros2_fuzzer RoboFuzz [116] imfit [78] camfitool ROSPenTo[117]
	MTA2. Exploit automation for test case generation, test case prioritization and selection, oracle and monitor generation	[34], [54]	[58]	[118]	Equiv. Part.[118] Mithra[119] Trajectory Generator[120] Hypothesis [121]
System Execution	SE1. Use record-and-replay when performing exploratory field tests	[34], [54]	[46]	[122], [118], [123] [124], [63]	rosbag rosbag2 Rerun.io NUbots [123] BSN [124], [63]
	SE2. No GUIs! Prioritize headless simulation	[34], [54]	[58]	[125], [126]	CI [127] OpenDaVinci [128]
Analysis & Reporting	AR1. Perform postmortem analysis to diagnose non-passing test cases.	[29]	[129]	[130], [131], [132]	Overseer[129] RoSHA [130] CARE [131] Rason[132]
	AR2. Use reliable tooling in order to manage field data	[34], [26]	[83], [57], [129]	[118], [133]	wstool[11] [118] Overseer [129] FTT [133] warehouse_ros

Outline Logging and Filtering. ROSMonitoring [72] employs monitors to persist events in textual format. The monitors contain filtering capabilities to eliminate entries that are inconsistent with the specification (requires an oracle). In this context, the launch files to configure ROSMonitoring can be seen as an API for logging and filtering (git: ROSMonitoring/../online_config.yaml). Monitors in ROSMonitoring are nodes, so the API is a set of known topics and message formats. The tester, in that case, only needs to specify what topics ROSMonitoring will listen to and the type of message to be recorded. Logging and filtering happens within a separate service. On a similar stance, and inspired by aircraft black box (and software black box [73]), Mitrevski et al. [74] proposes the concept of Robotic Black Box (git: ropod-project/black-box). The black box operates as an isolated component responsible for listening to data traffic from distinct sources and logging in an easily retrievable medium. Similarly to ROSMonitoring, Mitrevski's black box approach to logging (git: ropod-project/black-box/../logger_main.*) inspects topics and message types that are configured in advance. Different from ROSMonitoring, Robotic Black Box offers logging not only in textual format but also in a MongoDB database, which supports data processing, filtering, and retrieval. Robotic Black Box stands out when it comes to its filtering and retrieval capabilities. The approach builds a customized query interface over the MongoDB database using names of collections, timestamps and metadata to filter the results (git: black_box_tools/db_utils.py).

Strengths An API for retrieval and filtering facilitates access to valuable information resulting in effortless observability of inner states and events.

Weaknesses Overuse of logging may result in performance issues. Incorrectly implemented logging and filtering capabilities may lead to noise in the data, impacting the reliability of the tests. Insufficient logging may result in an incomplete assessment of the system behavior and might generate false positives. Finally, it may also lead to security breaches if not done with caution.

4.4 Guidelines for Quality Assurance through Runtime Verification and Field-Based Testing

We synthesized the guidelines in Tab. 2 relevant to providing runtime verification and field-based testing quality assurance through *testing* activities. Analogous to the guidelines presented in Sect. 4.3, the guidelines for provisioning quality assurance through runtime verification and field-based testing are organized in groups and summarized in Fig. 5). We trace each guideline to the sources of information, namely, the relevant initial surveys (column Initial), the systematic literature review (column Lit. Review), the specific search phase (column Specific Search), and the repositories mining (column Exemplars). The Prepare Execution Environment (PE) group contains two guidelines: PE1 to warn about the overhead acceptance criteria, and PE2 to create models for runtime assessment. The Specify (Un)-Desired Behavior group (SDB) contains three guidelines concerning the specification of desired and/or undesired behaviors. SDB1 and SDB2 concern the specification of properties through the use of logic-based languages (SDB1) or Domain Specific Languages (DSLs) (SDB2). SDB3, in turn, focuses on scenario-based specifications of test cases. The Generate Monitors and Test Cases group (MTA) contains two guidelines. MTA1 focuses on how to improve the robustness of the system by performing noise and fault injection. MTA2 discusses how to exploit automation for monitoring and testing, e.g., generation and prioritization of test cases. The System Execution group (SE) contains two guidelines focusing on the importance of using record-and-replay when performing exploratory field tests (SE1) and the importance of headless simulation (without GUI) for optimization and/or automation (SE2). Finally, the Analysis & Reporting group (AR) contains two guidelines. AR1 focuses on performing postmortem analysis to diagnose non-passing test cases. While AR2 focuses on the use of reliable tooling to manage field data. In the remainder of this section, we show in detail the guideline PE1 on understanding the overhead acceptance criteria.

We provide an in-depth view of a representative guideline (PE1) for understanding the overhead acceptance criteria in the preparation of the execution environment.

PE1. Understand the overhead acceptance criteria

The guideline PE1 recommends quality assurance teams understand the monitoring, isolation, or security and privacy overhead acceptance criteria as a basis for deciding the runtime assurance strategy given that performance is fundamental in robotics applications. Using PE1 helps to avoid unexpected interruptions in the robotic mission due to the testing apparatus. However, the overhead calculation process may turn laborious and conflict with time-to-market requirements.

ID PE1
Title Understand the overhead acceptance criteria
Context (WHEN) According to practitioners, performance is among the three most important quality attributes when designing a robotic application in ROS [26], intended as the degree to which a robotic application performs its functions within specified time and is efficient in the use of resources. Performance is often important in robotics because many computations performed by robots tend to be data-intensive, e.g., computer vision, planning, and navigation [26]. Gaining confidence in, e.g, correctness, reliability, robustness, of robotic applications, then, should not interfere with the nominal performance of the robot under scrutiny. Less (or no) impact on performance is especially desirable when the assurance gathering process interacts with the running system, for instance, in-the-field testing and runtime verification. We name the extra load available for gaining confidence on ROS-based applications as *overhead acceptance criteria*. The overhead acceptance criteria can be allocated to monitoring, isolation, or maintaining the security of privacy during the runtime assessment session [34].
Reason (WHY) A rule of thumb says that more observations tend to enable more precise assurance arguments within a limit. Observing, however, is never free from side effects, incurring in overhead [29]. Therefore, the quality assurance team must contrast the precision of the assurance arguments against overhead introduced by the observation medium. The overhead acceptance criteria are the basis for deciding the runtime assurance strategy.
Suggestion (WHAT) The use of runtime verification or field-based techniques might add computational overhead. The QA team should understand how much overhead is acceptable; this is important to decide on a test strategy that will not severely impact the performance of the running system. Such overhead may be due to monitoring with ros_tracing (git: ros2/ros2_tracing), component isolation, or security and privacy maintenance overhead with ROSploit (git: seanrivera/rosploit).
Process (HOW) Typically, understanding the overhead acceptance criteria follows from contrasting the required performance for delivering the required service, aka nominal performance (e.g., time to reaction, latency, speed), against available resources (e.g., computing power). The QA team may use off-the-shelf ROS tooling, like git: ApexAI/Performance_test , to understand the nominal performance of the ROS-based application. The difference between nominal performance and expected performance may be allocated to the overhead acceptance criteria. If the nominal performance is close enough to the expected performance, there is not enough space for implementing runtime assurance techniques. We categorize three types of overhead that may affect the runtime assurance process: Runtime Monitoring [72], [80], Isolation overhead [83], [135], and Security and Privacy overhead [81], [84].
Exemplars Exemplar overhead analyses may be due to monitoring, isolation, and security and privacy. *Monitoring Overhead* is all extra load put on the system-under-test due to gathering, interpreting, and elaborating data about the execution. For example, ROSMonitoring (git: autonomy-and-verification-uol/ROSMonitoring) [72] determines the monitoring overhead by calculating the delay introduced in the message delivery time between ROS nodes. The authors analyze the overhead by varying the size of the system under monitoring, message passing frequency, and number of monitor nodes. However, their overhead analysis is not trans-

parent for gathering, interpreting, or elaborating on data, the analysis looks at the monitoring overhead as a black box. Another example is ros2 tracing (git: ros2/ros2_tracing) [80] that provides a tool for tracing ROS2 systems with low latency overhead. The authors measure monitoring overhead by collecting the time between publishing a message and when it is handled by the subscription callback. In short, monitoring and tracing tools add some overhead that typically affects the message passing latency, the QA team must understand and define a precise time allowance for this overhead.

Isolation Overhead is all extra load put on the system-under-test to guarantee that the runtime assessment will not interfere with the normal operation or produce undesired side effects. As an example, Lahami et al. [83] propose a safe and resource-aware approach to test dynamic and distributed systems. They achieve safety by employing testing isolation techniques such as BIT-based, tagging-based, aspect-based, cloning-based, and blocking-based and resource awareness by setting resource monitors such as processor load, main memory, and network bandwidth. The authors show that the overall overhead is relatively low and tolerable, mainly if dynamic adaptations are not commonly requested. However, there is no fine-grained evaluation of the overhead introduced by isolation techniques. In fact, isolation overhead is rarely reported [135].

Security and Privacy Overhead is all extra load put on the system-under-test for maintaining security and privacy constraints while testing. For example, ROS-Immunity is a security tool for defending ROS-based applications from malicious attackers. Rivera et al. [81], determine overhead for maintaining the ROS-based system secure, while operating, in terms of power consumption (in Watts) by comparing the system's power draw with and without their tool. From another stance, Breiling et al [84] present a secure communication channel to enable communication between ROS nodes using protocols such as Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) per each ROS topic in the application. The protocols follow three steps: an initial handshake with mutual authentication, using symmetric encryption (AES-256), and using Message Authentication Codes (MAC) for data integrity. Importantly, they evaluate the overhead introduced for each step, which amounted to a few percentage points (2%-5%) of increase in average CPU load.

Strengths Understanding the overhead acceptance criteria in advance may avoid unexpected interruptions in the ROS application functionality due to runtime quality assessment procedures. For example, when testing procedures are mistakenly scheduled whenever the ROS-based application is operating under a high load. Learning about the overhead in advance criteria may also ask for re-design due to a lack of verifiability during runtime. For example, when the nominal performance of the ROS system is close to the expected performance, in value.

Weaknesses Although there is tooling to support the assessment of the overhead acceptance criteria, the process for understanding involves executing the system, collecting performance data, and analysis. This may conflict with time-to-market requirements, asking for further negotiation with the business goals of the ROS-based system.

5 Validation

This section discusses the outcomes of the validation with experts from industry and academia made through questionnaires (available in the replication package section of the website [38]). Our goal is to check whether the guidelines synthesized from literature and insights extracted from open-source ROS repositories are considered to be useful, clear, and applicable by developers and QA teams testing and verifying ROS code. To this end, we elicit three hypotheses per guideline:

- H_1. Overall, the guideline is useful. *(Usefulness)*
- H_2. The formulation of the guideline is clear. *(Clarity)*
- H_3. The guideline is applicable to ROS-based systems. *(Applicability)*

Differently from usefulness, we note that applicability refers to the extent to which the respondents consider that the guideline could be directly applied to ROS-based systems that *they have worked with* at the moment of the survey or in the past. In turn, usefulness refers to ROS-based systems in general.

It is important to note that applicability, clarity, and usefulness are not independent aspects. In Sect. 6 we will investigate discrepancies among them to identify research gaps and research challenges for future work.

5.1 Participants

We received 55 answers to the questionnaire, of which 33 were from developers and 22 from quality assurance (QA) teams. To obtain those answers, we sent 1032 emails out of which: 306 targeted paper authors, 772 targeted developers (extracted from repositories), and 68 targeted experts we met personally. We excluded 114 emails due to intersections, e.g. paper authors that were also targeted developers. Additionally, we shared posts in the ROS discourse forum. Duplicated emails were removed. The 39-day campaign stopped when we did not receive any more answers from the candidate respondents. The questionnaires were tailored to check the three hypotheses. In addition, we asked each respondent to answer four other questions: (i) for how long they have worked in robotics, (ii) what their experience with ROS was, (iii) what kinds of organizations they have worked in, and (iv) what robotics domains they have worked on. Based on their answers, we created a summary of five fictitious but representative profiles, as described in Table 3.

5.2 Questionnaire Results

The Likert plot in Fig. 6 provides an overview of the answers from the questionnaire. While the majority of votes lean to the 'Agree' side, some guideline descriptions (i.e., CD2, CI1, CI2, CI3, I1, I2, I3, I4, MTA1, MTA2, PE2, SDB1, SDB2, SDB3, SE2) received 'Strongly Disagree' votes. This subsection mostly analyses disagreements in contrast to agreements since they might shed light on new research opportunities. To clarify the disagreements, we collected comments left in free text from the online questionnaire[12]. The questionnaire targeted per group comments, in which respondents could direct their

12. We make available the anonymized transcript of all comments in the replication package [38].

ID	Time	Experience	Organization	Domain	Role	# Individuals
P1	>10y	Contributed to ROS packages	Academia and Industry	Service Robotics	Developer	18
P2	3–5y	Worked on app. using ROS	Industry	Industrial automation	Developer	17
P3	1–3y	Contributed to ROS packages	Academia	General-purpose	Developer	6
P4	1–3y	Worked on app. using ROS	Academia and Industry	General-purpose	QA	3
P5	3–5y	Worked on app. using ROS	Academia and Independent Groups	Marine Robotics	QA	11

TABLE 3: Summary of the representative profiles for the 55 questionnaire respondents. Note: app. stands for applications

suggestion either to the whole group or to specific guidelines, usually indicating their identifier.

Guideline CD2 concerns ensuring global time monotonicity of events and states to address the potential non-determinism in the scheduling of events in ROS-based applications. In general, comments left to CD2 emphasize the need to ensure the monotonicity of events, *"While typically small, occasionally you will see a large time jump if NTP does something unexpected. With enough robots and enough usage, these (and surely other) odd events will occur."*. While some comments question whether monotonicity of events should be an explicit or implicit property of the design phase, there is a suggestion on how to address the time synchronization problem using the NTP/PTP time protocols, i.e.,*"NTP/PTP status and verifying that these time protocols have converged to a stable solution with small offsets"*.

Guideline CI2 concerns security and privacy, which are often under-explored in the design of ROS-based applications. Threats to privacy and security may be catastrophic and may be an unacceptable side-effect of runtime assessment of such robotic applications. During the validation of this guideline, the respondents were mostly unsatisfied with the wording. They claim that the guideline uses *"complicated language"* or *"worded confusingly"*. Ultimately asking for a guideline refinement with a better understanding of the SROS tool and the Alias Robotics company, and leveraging separate criteria to form security constraints: *"(1) software BOM and patch process, (2) application security, and (3) user data privacy controls"*.

Guideline I3 concerns providing an API for stimulating unexpected scenarios for testing the robustness of the target system; this can help addressing challenges in finding both fault and error that are representative of real software faults. During the validation of this guideline, respondents claim that they agree with the guideline's intention but warn that implementing such a solution might lead to little return on investment, they assert that *"[...] it will result in a lot of development to implement fake faults, and you're still going to miss so many real faults"*. While they say that *"Building tools to simulate faults can also be very difficult and time-consuming."*, they also highlight that such approaches also introduce safety risks when injecting faults in field tests: *"Injecting faults as part of the codebase in safety systems [...] would never be approved by certification entities. [...]."*.

Guideline SDB1 concerns the use of a logic-based language for specifying properties that describe observable actions, outputs, how they relate to each other and when they should manifest. To simplify the specification of properties in temporal logic, some of the tools refer to existent property specification patterns [136], which also enable the formulation of properties in structured and unambiguous English. Guideline SDB2 proposes to address the specification problem with a code-like alternative, a built-in ROS-tailored language allowing quality assurance teams to specify and validate the correct behaviour of the system. Focusing on testing, SDB3 proposes to use scenario-based test case generation approaches for the systematic exploration of different situations and conditions that the robotic system may encounter in the real world. For the SDB group, the respondents discussed that they believe that behavior specification is an activity that is targeted to senior developers rather than QA experts. They say *"I wish QA teams could do such advanced tasks. Those are senior developers, not QA"*.

In light of this discussion, we have some understanding concerning the applicability of the SDB group of guidelines, as well as of CD2, CI2, and I3.

Guideline I1 recommends the use of ROS nodes with lifecycle management to provide a structured way to manage the state of the nodes and the interactions between them. This structure helps (i) ensuring that nodes are in the right state for testing, (ii) guaranteeing that the interactions between nodes are predictable, and (iii) mitigating dangling references to nodes that are no longer in use. Concerning I1, this guideline recommends an API for querying and updating internal lifecycle. This is considered to be, overall a useful guideline, but we understand that it requires special conditions to realize it, e.g., full control of the robotic system. It is interesting to highlight that this guideline is very similar to guideline B1 in [26], where the authors suggest that the behavior of each node should follow a well-defined life cycle, which should be queryable and updatable at runtime.

We do not have enlightening comments for the remaining guidelines CI1, CI3, I2, I4, MTA1, MTA2, PE2, and SE2. However, the overall evaluation of many of them is quite positive and no further investigation is needed. Instead, we feel that it would be worth better investigating the usefulness, clarity, and applicability of guidelines MTA2, PE2, and CI2, which are those with scores of applicability lower than 65% and for which we have no further information from the questionnaires. To better analyze the validation results we performed statistical and practical significance analysis.

5.3 Statistical Analysis

In complement to the Likert plot analysis, we back up the responses using statistical analysis. First, we assume that the answers from the Likert items are symmetric and this allows us to use boxplots and visualize whether the hypotheses ($H_1 - H_3$) hold for each guideline from a bird's eye view. We map the Likert items to integer values according to the following rule: *Strongly Disagree*=1, *Disagree*=2, *Agree*=3, *Strongly Agree*=4. The resulting Fig. 7 compares the boxplots for each guideline (y-axis) for the attributes we tested against, i.e., applicability, clarity, and usefulness; while the x-axis determines the Likert Items, i.e., Strongly Disagree, Disagree, Agree, and Strongly Agree.

At first glance at the plot from Fig. 7, we observe that the *medians* revolve around *Agree* and *Strongly Agree* for the three dimensions (i.e., applicability, clarity, and usefulness). Therefore, overall developers and QA teams agree that the synthesized guidelines are applicable, useful, and clear.

In light of such results, we further analyzed the statistical and practical significance regarding the agreement for these guidelines (see Sect. 5.3.1). For those guidelines where the significance test failed or practical significance had a small effect, we conducted a follow-up discussion with the questionnaire respondents to understand the causes for disagreement (see Sect. 5.4).

5.3.1 Significance Analysis

In light of the significance analysis, we can claim that all guidelines are *applicable*, *clearly* formulated, and *useful* according to the 55 responses from the questionnaire. We conclude also that guidelines CI2, MTA2, and PE2 ask for more information to draw solid conclusions on whether they are *applicable* or not. We reached this conclusion thanks to the analysis detailed in the following.

In preparation for the significance analysis tests, we transform the input data from categorical (i.e., *Strongly Disagree*, *Disagree*, *Agree*, *Strongly Agree*), to numeric (*Strongly Disagree & Disagree* = -1 and *Strongly Agree & Agree* = 1). The mapping separates the answers in a format such that both sides entail an equal Euclidean distance from the center, 0. This format allows us to sum the points for each respondent and evaluate whether the aggregate rejects ($sum < 0$) or accepts ($sum > 0$) the hypotheses $H_1 - H_3$. In addition, we applied Shapiro Wilk's normality test to understand whether the data is normal (available in the validation section of the guidelines website [38]) and the test reveals that it is not. Thus, we employed the non-parametric one-sample Wilcoxon test for statistical and practical significance.

Statistical Significance: We test our hypotheses for statistical significance by employing the one-sample Wilcoxon test. The test checks whether the sum of all the respondents for each guideline, e.g. AR1, and each dimension, e.g., applicability, is greater than 0 with a 5% significance level. The resulting plot (Fig. 8) summarises all the tests, combinations of guidelines, and dimensions.

Importantly, Fig. 8 highlights that guidelines CI2, MTA2, and PE2 resulted in p-values larger than 0.05, specifically for *applicability*. Precisely, CI2$_{p-value}$ $= 0.1948536$, MTA2$_{p-value}$ $= 0.3416874$, and PE2$_{p-value}$ $= 0.1037148$. The result suggests that the data collected does not confirm that the aforementioned guidelines are applicable to ROS-based systems, as pointed out by the medians from Fig. 7. In complement, the tests confirm that the remaining correlation between guideline (below the red dashed line in Fig. 8) and attributes *applicability*, *clarity*, and *usefulness* is statistically significant.

Practical Significance: We test our hypotheses for practical significance by calculating the effect size given by the one-sample Wilcoxon signed-rank test. Similarly to the statistical significance test, the test checks whether the effect size for one sample Wilcoxon test is greater than zero for the sum of all respondent's answers to the combinations of guidelines and dimensions. The resulting plot (Fig. 9) summarises the effect size for all combinations of guidelines and dimensions.

According to the definitions from Cohen J. [137] limiting values for effect size are small effect < 0.3, moderate effect > 0.3 and < 0.5, and large effect > 0.5. When it comes to applicability, guidelines CI2, MTA2, and PE2 fall in the small effect category, confirming that we need further data to strengthen our conclusions. Guidelines AR2, CI2, I1, I3, SDB1, and SDB3 are ranked with moderate size effect, which is enough in our case to justify their practical significance. Finally, the large-size effect guidelines are the rest, which confirms their practical significance .

5.4 Follow-up With Respondents

Given the lack of statistical and practical significance to guidelines CI2, MTA2, and PE2, we followed up with respondents who willingly left their contact in the questionnaire and opted for disagree or strongly disagree. Then, we sent to the respondents an email asking for further details and reasoning.

According to their availability to provide further feedback in the follow-up process, we contacted 15 individuals, out of which 10 were academics and 5 were practitioners in the industry. From the 10 responses we received, 7 were academics, and more specifically, 3 were professors, 3 were PhD students and 1 was a technician. Meanwhile, on the practitioner side, out of the 3 responses, two were research engineers and one was a developer/software architect.

In the remainder of this section, we summarize the points made by the researchers and practitioners concerning the applicability of guidelines CI2, MTA2, and PE2 in their line of work.

For MTA2, which refers to exploiting automation tools for test case generation, test case selection, and oracle generation, the main points made were:

- As one researcher suggested, the project needs to be large enough to justify developing models for exploiting test and monitor automation:

  "If the project is too small, the additional effort required to develop some sort of digital twin is too high compared to the project."
- For complex scenarios, the automatic generation of test environments could be computationally too demanding.

For PE2, which refers to exploiting system models system or its environment, as well as the creation of digital twins, the main points were:

- The amount of work surpasses the project' scale.
- It is a difficult challenge to create realistic data for large complex scenarios, even if the model is accurate, given that the computational load can be so high that the real-time factor is so low that it compromises the testing reliability testing. As one expert put it:

  "I am working now in a system which the goal is identifying cracks in tunnels, we could not create digital twins (at least yet) because [if] even we have the perfect model, super detailed, of the real world,

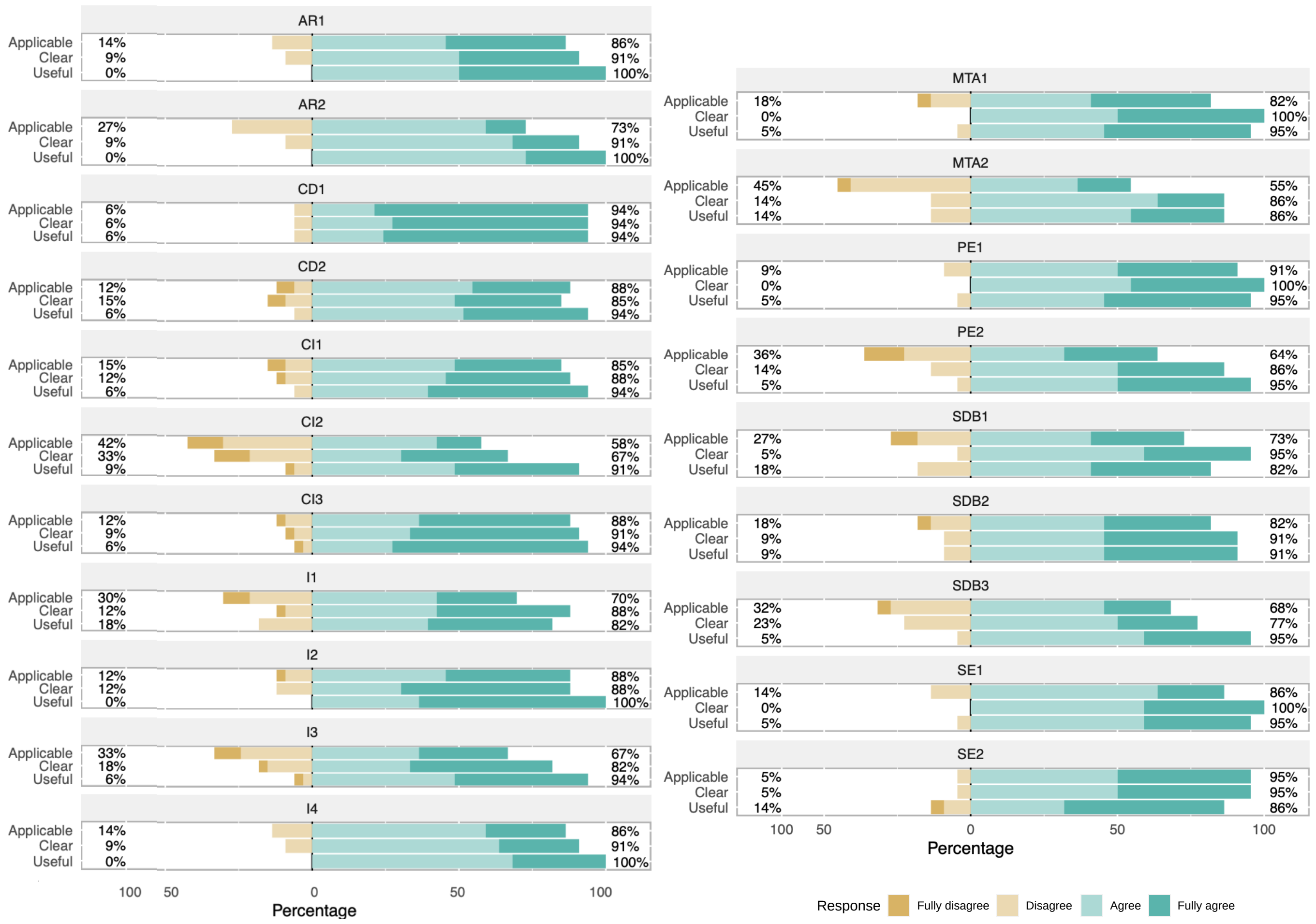

Fig. 6: Likert plot for the questionnaire answers

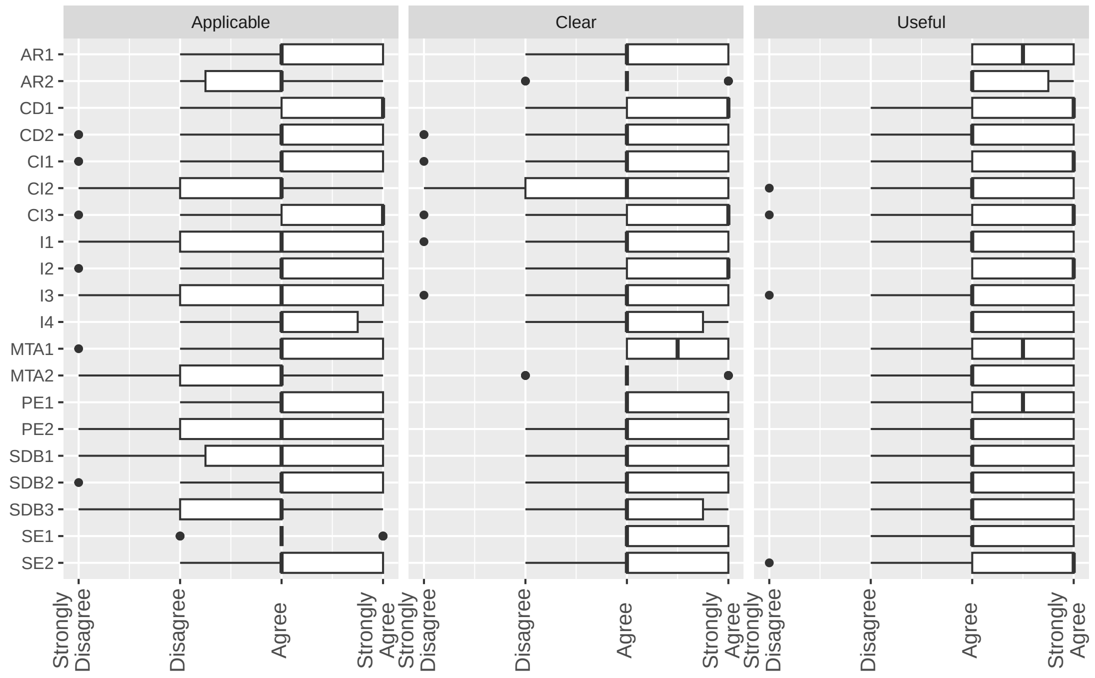

Fig. 7: Boxplots for the results bird's eye view

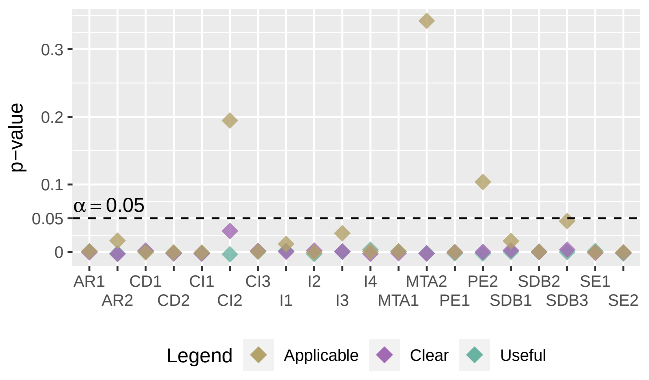

Fig. 8: Wilcoxon one-sample test for statistical significance (hypothesis: $\mu \geq 0$)

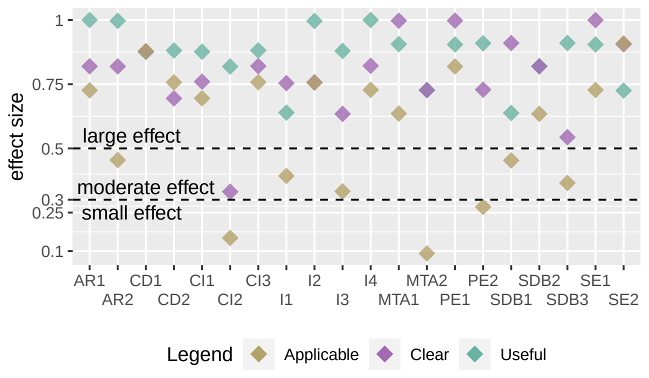

Fig. 9: Effect size for one sample Wilcoxon test for practical significance

it is not reliable to run in simulators, because the model is too big, and then the real time factor of the simulation is smaller than 0.3, even with a really good computer."

For CI2, which refers to identifying security or privacy vulnerabilities, the major issues raised were:

- It is not applicable or relevant to their projects in the current state.
- Academic projects usually see no need for such measures as there is not an abundance of sensitive information present.
- There are alternative ways to address some of the more frequent vulnerabilities e.g., making the system unreachable from the Internet. As one expert mentions:

  "We've built a new version of the Gateway around the super-secure Turris Omnia router, the only extra security feature we use from this router is the dynamic firewall Sentinel it offers out of the box."

As a result of our follow-up, we revised the guidelines CI2, MTA2, and PE2 for further clarification following the respondents' latest feedback. Acknowledging the experts' comments, we added statements to the *Weaknesses* field of the guidelines addressing their concern and better delimiting the scope of applicability each guideline. The modifications are both represented in our online guidelines repository in a pull request[13] and represented in the guidelines in the appendix – additions are marked with blue highlight and deletions with red.

6 Discussion

We now discuss open challenges in field-based testing [34] and runtime verification [29], [138], how our guidelines mitigate them, and what opportunities for future research exist. We selected challenges from the main papers on field-based testing and runtime verification used as initial surveys in our paper. We merged two challenges and excluded two challenges that are not relevant either for the robotics domain or are not focused on software. It is important to highlight that these survey papers do not focus on ROS systems. This explains why some of the challenges are not connected to any guideline, as summarized inTable 4, which provides a mapping between open challenges and guidelines. We will also exploit identified discrepancies among the applicability, clarity, and usefulness validation to identify research gaps and challenges.

6.1 Lack of (Formal) Specification

Formal specifications are used in test case generation and oracle or monitor definition. However, formal specifications are typically not found in real systems [34]. This poses a threat to using automated processes to indulging in generating test artifacts. Guidelines on *specification of (un)desired behavior* (SDB1, SDB2, SDB3) propose means to discuss and present languages, e.g., LTL, MTL, STL, ROSRV, RuBaSS, Geoscenario, SCENIC, that may assist the specification of behavior. Towards this direction, works from Hammoudeh [97], [98] take a step forward and propose a model-driven engineering approach by creating meta-models for ROS, further detailed in PE2. What is not defined by their work is the extent to which data from the field may be used in the specifications to shorten the gap between high-level models and concrete real-world scenarios.

According to the evaluation in Sect. 5, these guidelines (SDB ones) have all a discrepancy between usefulness, generally considered as high, and applicability to projects in which respondents have been working, where the evaluation shows less support. This could imply that, even though the respondents see value in this set of guidelines, as supported by the usefulness, the specific conditions of the project in which they have been working were not ideal for these guidelines. This interpretation is supported by the results in Sect. 5. In fact, respondents reported that behavior specification is an activity that is targeted to senior developers rather than QA teams, which do not have the competencies to do such advanced tasks. The use of languages for the specification of (un)desired behaviors more accessible and easy to use for senior developers (rather than quality assurance experts) would potentially mitigate this discrepancy.

Concerning specification, we can identify a gap between academics and practitioners. In fact, stimulated by research results, we attempted to define a guideline concerning

13. https://github.com/ros-rvft/ros-rvft.github.io/pull/1/files

TABLE 4: Open challenges for runtime verification and field-based testingof ROS-based systems

Open Challenge	Guidelines	FT or RV?
Lack of (Formal) Specifications [34]	SDB1, SDB2, SDB3, PE2	FT
Generating and implementing field test cases [34]	MTA2, SEI1, PE2	FT
Isolation Strategies [34]	I4, CD1, PE1	FT
Oracle Definition [34]	MTA2	FT
Security and Privacy [34], [138]	CI2, PE1	RV&FT
Distributed monitoring [29], [138]	-	RV
Monitoring states [29]	I1	RV
Richer reactions [29]	-	RV
Support to imprecise traces [29]	CD2, AR2	RV

the application of Contract Programming or Design by Contract (DbC), a software design technique that enables the software designer to declare what the code is supposed to do [139], [140], to ROS, where the software components are represented by ROS nodes.

On the one hand, we found some interesting academic works. Contracts_lite [14] is a ROS package from the safety working group that enables the explicit definition of contracts and enforcement of rules. As a different alternative, Luckcuck et al. [141] take a stance on contract-based verification for ROS systems. In their work, the contracts are manually written using the ROS Contract Language (RCL) and the ROS-based system needs to be abstracted into a model before writing the contracts. On the other hand, we did not find much use of DbC for robotics in industrial settings. We found an informal discussion on this topic on ROS2[15], however, we believe that there are not enough available solutions and examples to formulate it as a guideline.This could be an interesting research direction to investigate.

Still concerning specification, some approaches in space ROS[16] use properties implicitly specified. Implicit specification happens when there is a general understanding of a particular desired behavior. For instance, Cobra and IKOS provide support for two static analysis tools from NASA with embedded or implicit properties.

We conclude this subsection by highlighting the opportunity to make formal specifications more accessible and usable in practice. A first step in this direction can be found in patterns for properties specification [142], [9], [136], which we mention in SDB1. This would make it easier for practitioners to use a set of tools for runtime verification and field-based testing that require a formal specification as input.

6.2 Generating, Implementing, Orchestrating, and Governing Field Test Cases

We proposed various guidelines on field-based testing, as summarized in the following:

- MTA2 aims at exploiting automation for field-based testing. In this guideline, we describe approaches concerning automatic test case generation and selection.
- Guideline SE1 focuses on the use of record-and-playback when performing exploratory field tests. Among other recommendations, this guideline suggests using exploratory field testing to find corner cases, and then field endurance tests [118] to test these corner cases.
- Guideline PE2 recommends the use of model-based runtime assessment to help manage complexity and ensure the safe operation of ROS-based applications. However, maintaining models always adds overhead.

However, we believe that generating, implementing, orchestrating, and governing field test cases includes many other aspects, and test automation in field-based testing is still limited and relies heavily on human contributions as already highlighted in [34]. In the following we discuss promising future research directions.

- Because of environment uncertainty, field test cases shall adapt to the operational environment. Also, testers should rely on rules and policies to establish when, how, by whom, and in which order, a selected set of tests can be executed [34]. This is made clear by the category of testing called self-adaptive testing in the field (SATF) [143], which focuses on field-based testing approaches that change over time to follow and adapt to the changes and evolution of systems, environments, or users' behaviors. We do not have a specific guideline focusing on this challenge.
- One of our respondents recommends using state machines when needed. State machines are useful to the extent that the robot knows precisely what state it is in and there is a discrete set of transitions to choose from. However, the robot may not have a single state (e.g., multimodal distribution of pose estimations). Furthermore, the vast majority of important control code pertains to the processes running within states, not to the transition logic between states. So, the recommendation is to not make a state-based architecture and quality assurance system if the majority of bugs do not have anything to do with state transitions.
- Another respondent would like to see guidelines concerning CI/CD pipelines in the quality assurance process. This can ensure that the code is automatically tested at every stage of development, thus identifying issues early and improving the overall quality of the software.
- Another of our respondents highlighted the wish for a guideline focusing on the identification of tests that should be re-executed at runtime to avoid the re-execution of all tests at runtime.

Because of environmental uncertainty, field test cases shall adapt to the operational environment. Also, testers

14. https://github.com/ros-safety/contracts_lite
15. https://discourse.ros.org/t/design-by-contract/2405
16. https://github.com/space-ros

should rely on rules and policies to establish when, how, by whom, and in which order, a selected set of tests can be executed [34].

6.3 Isolation Strategies

Field test cases must be non-intrusive. They should not intervene with the running processes or their data. However, it can be difficult and expensive to apply strategies to guarantee the isolation of the field test cases in practice [34]. Guideline I4 focuses on isolating components for testing. Similarly to what was recommended in [34], we found that Man-in-the-Middle (MITM) can be a good strategy to provide isolation of computing nodes for field test case isolation. However, this can be difficult for robotics. In fact, the two-way interaction between the software and the physical environment makes it impossible to assess the behaviour of one in isolation from the other. This can be mitigated, e.g., with model or software in-the-loop testing approaches.

I4 identifies two means to enact MITM in ROS, through a proxy design pattern, e.g., ROSRV [51], or through topic remapping, e.g., ROSMonitoring [72]. Guideline CD1 recommends that developers strive for ROS nodes with a single responsibility. It should reduce the cost of observing and controlling modules, or skills [60], to facilitate isolation. Guideline PE1 focuses on understanding the overhead acceptance criteria. It concerns the extra load put on the system-under-test for guaranteeing that the runtime assessment will not interfere with normal operation or produce undesired side effects.

6.4 Oracle Definition

Field testing oracles need to adapt to the uncertainty and the unknown execution conditions of the environment [34]. We do not have a specific guideline focusing on this challenge. MTA2 aims at exploiting automation for field-based testing, including the oracle generation. The guideline highlights *Mithra* [119], a novel and unsupervised oracle learning technique for Cyber-Physical Systems (CPS). Further work in this direction would help make field-based testing more popular in robotics. One of the respondents highlights the importance of incorporating guidelines for the use of CI/CD pipelines in the quality assurance process. This would include the use of autonomous oracles to automate the testing phase.

6.5 Security and Privacy

Executing test cases in the field can challenge security and privacy [34]. This is indeed an important challenge. Guideline CI2 concerns the identification of security and privacy constraints. These constraints can help mitigate this challenge. Guideline PE1 concerns with understanding the overhead acceptance criteria and it focuses also on the overhead due to maintaining security and privacy constraints while testing. Besides security and privacy, one of our respondents would like to see guidelines covering broader aspects of quality assurance, including resource management, adaptability, data-driven approaches, system resilience, energy, and ethical considerations. This can be an interesting direction for future research. Integrating other aspects of quality assurance can help in building a more robust, efficient, and reliable quality assurance framework for ROS-based systems, ensuring they are well-prepared for real-world deployment and operation.

6.6 Distributed Monitoring and Monitoring States

Decentralization of the monitoring architecture is identified as an area that has not received much attention [29], probably due to its inherent complexity. We did not identify guidelines related to this aspect.

Concerning monitoring states, it is not very common to find tools able to monitor the states of a program directly [29]. Guideline I1 focuses on APIs for querying and updating the internal lifecycle. The internal states within ROS nodes are typically hidden and not easily accessible. This limits the ability to diagnose and understand unexpected behavior. Exposing these hidden states is crucial for providing granular information on the system's behavior, rendering increased observability. Furthermore, managing these hidden states allows for actions such as starting, stopping, and rolling back to a specific state, in other words, increased controllability.

Most monitoring tools do not support active reactions, like enforcement, recovery, and explanations for declarative specifications [29]. Our respondents would like to have guidelines to design APIs for extra-node enforcement and for bringing a node for a specific state (e.g., rollback). We did not find guidelines related to this aspect.

6.7 Support to Imprecise Traces

Runtime verification tools rarely support imprecision in their input traces, e.g., imprecise timestamps, traces with incomplete events or inconsistencies in event sources, or due to purposefully omitted events when sampling [29]. Our guidelines do not provide recommendations or solutions on how to deal with imprecise traces. Instead, some guidelines try to explain how to mitigate imprecision. Guideline CD2 focuses on ensuring global time monotonicity of events and states. Ensuring global time monotonicity of events and states permits addressing the potential non-determinism in the scheduling of events in ROS-based applications. A way to achieve that is to annotate messages with timestamps and synchronization to guarantee ordering. Also, guideline AR2 promotes the use of reliable tooling to manage field data. The quality assurance team should use reliable tools for field data management to avoid problems with corrupted, unreliable, and/or incomplete data.

7 Threats to Validity

Internal validity focuses on the level of influence that extraneous variables may have on the design of the study [144]. Concerning the systematic literature review, we formalized the design of this study into a detailed research protocol and thoroughly discussed it before the actual execution of the study. Our research protocol is based on well-accepted guidelines for systematic literature reviews and mapping studies [145]. Concerning the repository mining, we strictly followed the approach in [26], which successfully gathered repositories from ROS. In addition, we make the protocols for the systematic literature review and repository mining available online [38]. The online material contains all the artifacts generated during the study, including the motivation behind our decisions.

External validity concerns the generalizability of the study's results [144]. The generalizability of our study depends strongly on whether the primary studies collected in the literature review represent runtime verification and field-based testing in robotic systems using the robot operating system. We mitigated the threat by carefully defining the terminology used for the search strings, and inclusion and exclusion criteria. The terminology was validated with researchers credited for their contributions in the domains of robotics (more specifically, in ROS), runtime verification, and field-based testing. Additionally, we use multiple data sources, i.e., ACM Digital Library, IEEE Xplore, Scopus, to increase the coverage of the state-of-the-art. Moreover, we performed an additional round of in-depth searches targeting specific facets of runtime verification and field-based testing in ROS. This covers studies that do not mention runtime verification or field-based testing explicitly, but that can be relevant for the study.

Construct validity concerns whether the constructs used are suited to answer the research questions [144]. To mitigate such threats, we performed a thorough quantitative and qualitative validation of each of the guidelines. The validation followed well-established advice surveys in software engineering [39]. Our study considered experts from different backgrounds within robotics domains and experience with ROS, including a large group of experts who claimed more than 10 years of experience. Whenever the quantitative results rendered inconclusive results, we discussed with experts and presented their comments *ipsis litteris* in our paper. The comments are also in the replication package [38]. The questionnaire asked the respondents about usefulness, applicability, and clarity, which might have led to the hypothesis guessing phenomenon [144]. We mitigate such threat by asking questions specific to each of the guidelines and asking open-ended questions. In light of this, we tried not to make bold conclusions but incorporated the comments, critiques, and suggestions for improvement in the guidelines.

Reproducibility The synthesis of guidelines from diverse sources of knowledge, in our case scientific literature and mined repositories, may turn out to be hard to replicate and reproduce. To mitigate such issues, we provided a guideline specification template that caters to how the information collected from the diverse knowledge source is composed to generate a guideline.

8 Related Work

This section compares our proposed guidelines to approaches for robotics systems, field-based testing, and other literature about devising guidelines for autonomous systems.

Testing Robotics Systems. Testing Autonomous and Robotics Systems is an area in expansion. However, it still faces barriers to support the verification and testing [19], [3]. Song et al. [146], for instance, elicit testing challenges in the industry: unpredictable environment, complexity stemming from the system, scenario, or modeling, data accessibility, and, finally, lack of support standards and guidelines to the adoption of testing.

Brito et al. [147] promotes integration testing in ROS by releasing a testing framework for checking functional behavior or reaching a high structural coverage thus revealing common faults in mobile robotic systems. Similarly, Laval et al. [148] introduces a methodology to support the definition of repeatable, reusable, and semi-automated testing for mobile robotic systems. Babić et al. [149] implements a testing framework targeting heterogeneous marine robots to benefit from simulations while also reflecting the complexity of the real world. Their approach contributes to developing a vehicle-in-the-loop system that can seamlessly be swapped during testing. Biagola and Tonella [150] proposed a test generator for online testing of autonomous driving systems (ADS), namely GENBO (GENerator of BOundary state pairs). GENBO works by mutating the driving conditions of the ego vehicle controlled by the DNN model, within a fixed driving condition scenario from an existing failure-free environment instance. Although the approach is not particularly tailored for ROS-based systems and is limited to the ADS domain, the methodology could shed some light on improving runtime models through exploratory test generation of ROS-based systems.

Field-based Testing. Uniquely, Ortega et al. [118] report the experience of three developers in performing exploratory and endurance field tests in the Kelo AD disinfection (mobile) robot. Differently from our approach, their work defines a necessary first step to exploring field testing practices in the company's particular context. We, instead, survey the literature and repositories and synthesize guidelines based on data.

Moreover, Bertolino et al. [34] and Gazzola et al. [151] discuss Field Testing as a technique for attaining confidence in software-intensive systems. Bertolino surveys the literature and classifies field-based testing techniques of ex-vivo, offline, and online field testing approaches according to their own taxonomy. Gazzola et al. [151] presented the concept of field-ready test cases, tests designed to run in production environments. They showcase the usefulness of testing JFreeChart and Apache Commons Lang libraries in the field by exposing faults that escape testing during development. Differently, our work focuses on understanding field-based testing and runtime verification for robotics systems.

Riedesel [152] defines software telemetry as a set of practices including centralized logging, distributed tracing, and security information event management to handle data from production until delivery to consumers such as software engineers, customer support, or compliance teams. Riedesel's book [152] offers a practical view of telemetry (i.e. monitoring) systems and dives into a pipeline for telemetry as a conceptual architecture for building telemetry systems. Differently from our work, the book does not consider issues that are specific to ROS-based systems and, although their suggestions are fruitful to the discussion at a higher level, they do not help to solve ROS particularities. Moreover, we present guidance targeting software development that should facilitate telemetry (in their terms), which is not discussed in the book.

Guidelines. We also discuss existing studies that propose guidelines to support the development of robotic systems. Related works on specific guidelines can be found in the description of every guideline, as summarized in Tables 1 and 2, especially columns *specific search* and *exemplars* and in the detailed description of the guidelines, where, for each guideline, we provide some related works and exemplars. The guidelines are described as a whole in a website destined for them [38].

Malavolta et al. [26] propose 47 guidelines for supporting roboticists in architecting ROS-based systems. The guidelines are mined from a dataset of 335 GitHub repositories containing open-source ROS-based systems. Our work is also focused on ROS-based systems, but instead of focusing on architectural aspects, we focus on field-based testing and runtime verification.

Weyns et al. [153] provide a set of guidelines for artifacts that support industry-relevant research on self-adaptation. Artifact providers can use these guidelines for aligning future artifacts with industry needs. The guidelines are based on data obtained from a survey of practitioners and have been defined during working sessions at the SEAMS 2022 (17^{th} International Symposium on Software Engineering for Adaptive and Self-Managing Systems) conference. In [154] Weyns et al. present six principles for engineering smarter CPS covering the main stages of the lifetime of CPS: domain engineering, design, operation, and evolution. In their work, they raise the need for both new toolchains and engineers with a deep-rooted understanding of how to develop software able to adapt and evolve under continuous change. None of these guidelines are particularly tailored to robotics, as we do instead; also, we focus on field-based testing and runtime verification.

In the healthcare domain, Bombarda et al. [155] build on their experience in certifying software produced under emergency. By "under emergency", the author refers to development under strict and pressing time constraints and with the difficulty of establishing a heterogeneous development team made of volunteers. The paper's contribution consists of lessons learned and guidelines, aiming to assist developers in producing safety-critical devices in similar emergencies. The authors discuss each guideline's benefits and risks with the guidelines. These guidelines focus on a specific domain different from robotics and focus on a specific setting, i.e., certifying software produced under emergency.

9 Conclusion

We proposed 20 guidelines for developers and QA teams that aim at testing and verifying ROS-based robotic systems in the field. Iteratively using design science, we identified and constructed them via a literature review of runtime verification and field-based testing studies on robotic systems, and mining ROS-based application repositories. Grounded on data, the guidelines were deemed actionable by robotics practitioners and researchers who answered a questionnaire that assessed whether the guidelines were clear, useful, and applicable in their work domain. Our catalogue of guidelines is extensible and the connected website provides instruments to collect suggestions and recommendations from other researchers and practitioners interested in the domain.

Our guidelines are relevant for researchers and practitioners. The former can benefit from promising future research directions highlighted in our mapping from known challenges to guidelines, e.g., lack of formal specifications, isolation strategies, richer monitoring reactions, and support to imprecise traces. The latter can find best practices and recommendations on designing systems to facilitate effective verification and validation of their ROS-based systems at runtime.

Acknowledgment

This work is supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The authors acknowledge the support of the PNRR MUR project VITALITY (ECS00000041), Spoke 2 ASTRA - "Advanced Space Technologies and Research Alliance", of the PNRR MUR project CHANGES (PE0000020), Spoke 5 "Science and Technologies for Sustainable Diagnostics of Cultural Heritage", the PRIN project P2022RSW5W - RoboChor: Robot Choreography, the PRIN project 2022JKA4SL - HALO: etHical-aware AdjustabLe autOnomous systems, the European Center AgriBioSERV (SERvices for AGRIfood and BIOmedicine market), and of the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI. The work of P. Pelliccione was also partially supported by the Centre of EXcellence on Connected, Geo-Localized and Cybersecure Vehicles (EX-Emerge), funded by the Italian Government under CIPE resolution n. 70/2017 (Aug. 7, 2017). This work was financed in part by FAPDF under Call 04/2021 and CNPq process 313215/2021-9.

References

[1] G.-Z. Yang, J. Bellingham *et al.*, "The grand challenges of science robotics," *Science Robotics*, vol. 3, no. 14, 2018.

[2] D. Brugali and E. Prassler, "Software engineering for robotics [from the guest editors]," *IEEE Robotics & Automation Magazine*, vol. 16, no. 1, 2009.

[3] S. García, D. Strüber *et al.*, "Robotics software engineering: A perspective from the service robotics domain," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.

[4] N. Blümlein, "Function-based cost estimation for service robot prototypes in early design phases," Ph.D. dissertation, University of Stuttgart, 2014.

[5] J. Cámara, B. Schmerl *et al.*, "Software architecture and task plan co-adaptation for mobile service robots," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020.

[6] M. Askarpour, C. Tsigkanos *et al.*, "Robomax: Robotic mission adaptation exemplars," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2021.

[7] S. Dragule, S. G. Gonzalo *et al.*, "Languages for specifying missions of robotic applications," *Software Engineering for Robotics*, 2021.

[8] C. Menghi, C. Tsigkanos *et al.*, "Mission specification patterns for mobile robots: Providing support for quantitative properties," *IEEE Transactions on Software Engineering*, 2022.

[9] ——, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, oct 2021.

[10] S. Dragule, T. Berger *et al.*, "A survey on the design space of end-user oriented languages for specifying robotic missions," *International Journal of Software and Systems Modeling*, vol. 20, no. 4, 2021.

[11] D. Kortenkamp, R. Simmons *et al.*, "Robotic systems architectures and programming," *Springer handbook of robotics*, 2016.

[12] G. Rodrigues, R. Caldas *et al.*, "An architecture for mission coordination of heterogeneous robots," *Journal of Systems and Software*, vol. 191, 2022.

[13] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i) [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 16, no. 4, 2009.

[14] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part ii)," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, 2010.

[15] R. Ghzouli, T. Berger *et al.*, "Behavior trees and state machines in robotics applications," *IEEE Transactions on Software Engineering*, 2023.

[16] A. Gotlieb, D. Marijan *et al.*, "Testing industrial robotic systems: A new battlefield!" *Software Engineering for Robotics*, 2021.

[17] A. Cavalcanti, W. Barnett *et al.*, "Robostar technology: A roboticist's toolbox for combined proof, simulation, and testing," *Software Engineering for Robotics*, 2021.

[18] F. Ingrand, "Verification of autonomous robots: A roboticist's bottom-up approach," *Software engineering for robotics*, 2021.

[19] D. Bozhinoski, D. Di Ruscio *et al.*, "Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective," *Journal of Systems and Software*, vol. 151, 2019.

[20] S. García, D. Strüber *et al.*, "Software variability in service robotics," *Empirical Softw. Engg.*, vol. 28, no. 2, dec 2022.

[21] C. Hildebrandt, M. von Stein, and S. Elbaum, "Physcov: Physical test coverage for autonomous vehicles," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023.

[22] M. Rizwan, R. Caldas, C. Reichenbach, and M. Mayr, "Ezskiros: A case study on embedded robotics dsls to catch bugs early," in *2023 IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 2023.

[23] C. Hildebrandt and S. Elbaum, "World-in-the-loop simulation for autonomous systems validation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[24] A. Alami, Y. Dittrich *et al.*, "Influencers of quality assurance in an open source community," in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2018.

[25] A. Santos, A. Cunha *et al.*, "The high-assurance ros framework," in *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 2021.

[26] I. Malavolta, G. A. Lewis *et al.*, "Mining guidelines for architecting robotics software," *Journal of Systems and Software*, vol. 178, 2021.

[27] K. Havelund and A. Goldberg, "Verify your runs," in *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 2005.

[28] E. Bartocci, Y. Falcone *et al.*, "Introduction to runtime verification," *Lectures on Runtime Verification: Introductory and Advanced Topics*, 2018.

[29] Y. Falcone, S. Krstić *et al.*, "A taxonomy for classifying runtime verification tools," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 2, 2021.

[30] S. Azzopardi, N. Piterman *et al.*, "Runtime verification meets controller synthesis," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2022.

[31] M. Rizwan, C. Reichenbach *et al.*, "Rossmarie: A domain-specific language to express dynamic safety rules and recovery strategies for autonomous robots," in *Second Workshop on Quality and Reliability Assessment of Robotic Software Architectures and Components*, 2023.

[32] M. Vierhauser, A. Garmendia *et al.*, "Grum—a flexible model-driven runtime monitoring framework and its application to automated aerial and ground vehicles," *Journal of Systems and Software*, vol. 203, 2023.

[33] S. G. Brunner, P. Lehner *et al.*, "Design, execution, and post-mortem analysis of prolonged autonomous robot operations," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, 2018.

[34] A. Bertolino, P. Braione *et al.*, "A survey of field-based testing techniques," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, 2021.

[35] R. Wieringa, "Design science as nested problem solving," in *Proceedings of the 4th international conference on design science research in information systems and technology*, 2009.

[36] R. Baskerville, A. Baiyere *et al.*, "Design science research contributions: Finding a balance between artifact and theory," *Journal of the Association for Information Systems*, vol. 19, no. 5, 2018.

[37] A. R. Hevner and V. C. Storey, *Externalities of Design Science Research: Preparation for Project Success*. Springer International Publishing, 2021, vol. 12807 LNCS.

[38] R. D. Caldas, J. A. P. García *et al.*, "Replication package," Jul. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.12736738

[39] B. Kitchenham, O. P. Brereton *et al.*, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, 2009.

[40] Dronecode, "PX4 Autopilot Project," 2023. [Online]. Available: https://px4.io/

[41] M. Luckcuck, M. Farrell *et al.*, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, 2019.

[42] M. Proetzsch, T. Luksch *et al.*, "The behaviour-based control architecture ib2c for complex robotic systems," in *KI 2007: Advances in Artificial Intelligence: 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007. Proceedings 30*. Springer, 2007.

[43] Y. Falcone, K. Havelund *et al.*, "A tutorial on runtime verification," *Engineering dependable software systems*, 2013.

[44] Y. Falcone, M. Jaber *et al.*, "Runtime verification of component-based systems," in *Software Engineering and Formal Methods: 9th International Conference, SEFM 2011, Montevideo, Uruguay, November 14-18, 2011. Proceedings 9*. Springer, 2011.

[45] W. Schütz, *The testability of distributed real-time systems*. Springer Science & Business Media, 2007, vol. 245.

[46] M. Beul, N. Krombach *et al.*, "Autonomous navigation in a warehouse with a cognitive micro aerial vehicle," in *Robot Operating System (ROS)*. Springer, 2017.

[47] T. Lewis and K. Bhaganagar, "Configurable simulation strategies for testing pollutant plume source localization algorithms using autonomous multisensor mobile robots," *International Journal of Advanced Robotic Systems*, vol. 19, no. 2, 2022.

[48] Z. Li, A. Hasegawa *et al.*, "Autoware_perf: A tracing and performance analysis framework for ros 2 applications," *Journal of Systems Architecture*, vol. 123, 2022.

[49] M. Albonico, M. DJević *et al.*, "Software engineering research on the robot operating system: A systematic mapping study," *Journal of Systems and Software*, vol. 197, 2023.

[50] C. Hu, W. Dong *et al.*, "Runtime verification on hierarchical properties of ros-based robot swarms," *IEEE Transactions on Reliability*, vol. 69, no. 2, 2019.

[51] J. Huang, C. Erdogan *et al.*, "Rosrv: Runtime verification for robots," in *International Conference on Runtime Verification*. Springer, 2014.

[52] V. Mayoral-Vilches, R. White *et al.*, "Sros2: Usable cyber security tools for ros 2," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[53] S.-Y. Jeong, I.-J. Choi *et al.*, "A study on ros vulnerabilities and countermeasure," in *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 2017.

[54] A. Afzal, C. Le Goues *et al.*, "A study on challenges of testing robotic systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020.

[55] S. Adam, M. Larsen *et al.*, "Towards rule-based dynamic safety monitoring for mobile robots," in *Simulation, Modeling, and Programming for Autonomous Robots: 4th International Conference, SIMPAR 2014, Bergamo, Italy, October 20-23, 2014. Proceedings 4*. Springer, 2014.

[56] M. Stadler and M. Vierhauser, "Romosu: Flexible runtime monitoring support for ros-based applications," in *RoSE'23: 5th International Workshop on Robotics Software Engineering Proceedings*, 2023.

[57] C. Hartsell, N. Mahadevan *et al.*, "Model-based design for cps with learning-enabled components," in *Proceedings of the Workshop on Design Automation for CPS and IoT*, 2019.

[58] C. Hutchison, M. Zizyte *et al.*, "Robustness testing of autonomy software," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2018.

[59] S. Limsoonthrakul, M. N. Dailey *et al.*, "A modular system architecture for autonomous robots based on blackboard and publish-subscribe mechanisms," in *2008 IEEE International conference on robotics and biomimetics*. IEEE, 2009.

[60] M. Efatmaneshnik and M. Ryan, "A study of the relationship between system testability and modularity," *INSIGHT*, vol. 20, no. 1, 2017.

[61] F. Rovida, M. Crosby *et al.*, "Skiros—a skill-based robot control platform on top of ros," *Robot Operating System (ROS) The Complete Reference (Volume 2)*, 2017.

[62] A. Albore, D. Doose *et al.*, "Skill-based design of dependable robotic architectures," *Robotics and Autonomous Systems*, vol. 160, 2023.

[63] E. B. Gil, R. Caldas *et al.*, "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). IEEE, 2021.

[64] R. Halder, J. Proença *et al.*, "Formal verification of ros-based robotic applications using timed-automata," in *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*. IEEE, 2017.

[65] K. Chaaban, "A new algorithm for real-time scheduling and resource mapping for robot operating systems (ros)," *Applied Sciences*, vol. 13, no. 3, 2023.

[66] H. Choi, Y. Xiang *et al.*, "Picas: New design of priority-driven chain-aware scheduling for ros2," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021.

[67] T. Blaß, D. Casini *et al.*, "A ros 2 response-time analysis exploiting starvation freedom and execution-time variance," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021.

[68] G. Conte, D. Scaradozzi *et al.*, "Development and experimental tests of a ros multi-agent structure for autonomous surface vehicles," *Journal of Intelligent & Robotic Systems*, vol. 92, no. 3, 2018.

[69] K. Belsare, A. C. Rodriguez *et al.*, "Micro-ros," in *Robot Operating System (ROS) The Complete Reference (Volume 7)*. Springer, 2023.

[70] A. Nordmann, R. Lange *et al.*, "System modes-digestible system (re-) configuration for robotics," in *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 2021.

[71] J. Staschulat, I. Lütkebohle *et al.*, "The rclc executor: Domain-specific deterministic scheduling mechanisms for ros applications on microcontrollers: work-in-progress," in *2020 International Conference on Embedded Software (EMSOFT)*. IEEE, 2020.

[72] A. Ferrando, R. C. Cardoso *et al.*, "Rosmonitoring: a runtime verification framework for ros," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2020.

[73] S. Elbaum and J. C. Munson, "Software black box: an alternative mechanism for failure analysis," in *Proceedings 11th International Symposium on Software Reliability Engineering. ISSRE 2000*. IEEE, 2000.

[74] A. Mitrevski, S. Thoduka *et al.*, "Deploying robots in everyday environments: Towards dependable and practical robotic systems," *arXiv preprint arXiv:2206.12719*, 2022.

[75] J. A. Duraes and H. S. Madeira, "Emulation of software faults: A field data study and a practical approach," *Ieee transactions on software engineering*, vol. 32, no. 11, 2006.

[76] R. Barbosa, J. Karlsson *et al.*, "Fault injection," *Resilience Assessment and Evaluation of Computing Systems*, 2012.

[77] R. K. Lenka, S. Padhi *et al.*, "Fault injection techniques-a brief review," in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE, 2018.

[78] U. Yayan and C. Baglum, "Tailored mutation-based software fault injection tool (im-fit)," *SoftwareX*, 2023.

[79] S. Macenski, A. Soragna *et al.*, "Impact of ros 2 node composition in robotic systems," *IEEE Robotics and Automation Letters*, 2023.

[80] C. Bédard, P.-Y. Lajoie *et al.*, "Message flow analysis with complex causal links for distributed ros 2 systems," *Robotics and Autonomous Systems*, vol. 161, 2023.

[81] S. Rivera and R. State, "Securing robots: An integrated approach for security challenges and monitoring for the robotic operating system (ros)," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021.

[82] A. Bertolino, G. De Angelis *et al.*, "Enhancing service federation trustworthiness through online testing," *Computer*, vol. 45, no. 1, 2011.

[83] M. Lahami, M. Krichen *et al.*, "Safe and efficient runtime testing framework applied in dynamic and distributed systems," *Science of Computer Programming*, vol. 122, 2016.

[84] B. Breiling, B. Dieber *et al.*, "Secure communication for the robot operating system," in *2017 annual IEEE international systems conference (SysCon)*. IEEE, 2017.

[85] E. Guerrero, F. Bonin-Font *et al.*, "Adaptive visual information gathering for autonomous exploration of underwater environments," *IEEE Access*, vol. 9, 2021.

[86] N. Grigoropoulos and S. Lalis, "Simulation and digital twin support for managed drone applications," in *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2020.

[87] T. Dürschmid, C. S. Timperley, D. Garlan, and C. Le Goues, "Rosinfer: Statically inferring behavioral component models for ros-based robotics systems," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024.

[88] B. H. Cheng, R. J. Clark, J. E. Fleck, M. A. Langford, and P. K. McKinley, "Ac-ros: assurance case driven adaptation for the robot operating system," in *Proceedings of the 23rd acm/ieee international conference on model driven engineering languages and systems*, 2020.

[89] B. Pelletier, C. Lesire, C. Grand, D. Doose, and M. Rognant, "Predictive runtime verification of skill-based robotic systems using petri nets," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[90] C. Schlegel, T. Haßler *et al.*, "Robotic software systems: From code-driven to model-driven designs," in *2009 International Conference on Advanced Robotics*. IEEE, 2009.

[91] J. C. Kirchhof, J. Michael *et al.*, "Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020.

[92] T. Sotiropoulos, H. Waeselynck *et al.*, "Can robot navigation bugs be found in simulation? an exploratory study," in *2017 IEEE International conference on software quality, reliability and security (QRS)*. IEEE, 2017.

[93] C. Saavedra Sueldo, I. Perez Colo *et al.*, "Ros-based architecture for fast digital twin development of smart manufacturing robotized systems," *Annals of Operations Research*, 2022.

[94] A. Fend and D. Bork, "Cpsaml: a language and code generation framework for digital twin based monitoring of mobile cyber-physical systems," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022.

[95] J. Ernits, E. Halling *et al.*, "Model-based integration testing of ros packages: A mobile robot case study," in *2015 European Conference on Mobile Robots (ECMR)*. IEEE, 2015.

[96] K. G. Larsen, M. Mikucionis *et al.*, "Testing real-time embedded software using uppaal-tron: an industrial case study," in *Proceedings of the 5th ACM international conference on Embedded software*, 2005.

[97] N. Hammoudeh Garcia, M. Lüdtke *et al.*, "Bootstrapping mde development from ros manual code - part 1: Metamodeling," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019.

[98] N. Hammoudeh García, H. Deshpande *et al.*, "Bootstrapping mde development from ros manual code: Part 2—model generation and leveraging models at runtime," *Software and Systems Modeling*, vol. 20, no. 6, 2021.

[99] T. Yamaguchi, B. Hoxha *et al.*, "Rtamt–runtime robustness monitors with application to cps and robotics," *International Journal on Software Tools for Technology Transfer*, 2023.

[100] A. F. F. Santos, "Safety verification for ros software," Ph.D. dissertation, Universidade do Minho, 2021.

[101] S. Aldegheri, N. Bombieri *et al.*, "A containerized ros-compliant verification environment for robotic systems," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021.

[102] C. Lesire, S. Roussel *et al.*, "Synthesis of real-time observers from past-time linear temporal logic and timed specification," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[103] D. Ničković and T. Yamaguchi, "Rtamt: Online robustness monitors from stl," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020.

[104] Y. S. Kirca, E. Degirmenci, Z. Demirci, A. Yazici, M. Ozkan, S. Ergun, and A. Kanak, "Runtime verification for anomaly detection of robotic systems security," *Machines*, vol. 11, no. 2, 2023.

[105] E. Degirmenci, Y. S. Kırca, Ö. Örnek, M. Bulut, S. Kahraman, M. Özkan, and A. Yazıcı, "Developing an integrated runtime verification for safety and security of industrial robot inspection system," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2023.

[106] S. Adam, M. Larsen *et al.*, "Rule-based dynamic safety monitoring for mobile robots," *Journal of Software Engineering for Robotics*, vol. 7, no. 1, 2016.

[107] A. Balakrishnan, J. Deshmukh *et al.*, "Percemon: Online monitoring for perception systems," in *International Conference on Runtime Verification*. Springer, 2021.

[108] X. Duan, Q. Wang *et al.*, "Implementing trajectory tracking control algorithm for autonomous vehicles," in *2021 IEEE International Conference on Unmanned Systems (ICUS)*. IEEE, 2021.

[109] S. B. Liu, H. Roehm *et al.*, "Provably safe motion of mobile robots in human environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[110] R. Queiroz, D. Sharma *et al.*, "A driver-vehicle model for ads scenario-based testing," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[111] D. J. Fremont, E. Kim *et al.*, "Scenic: A language for scenario specification and data generation," *Machine Learning*, 2022.

[112] C. Xu, H. Huang *et al.*, "Sil simulation test platform based on prescan and ros," in *2022 International Conference on Cyber-Physical Social Intelligence (ICCSI)*. IEEE, 2022.

[113] J.-J. Bai, H.-X. Song, and S.-M. Hu, "Multi-dimensional and message-guided fuzzing for robotic programs in robot operating system," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024.

[114] K.-T. Xie, J.-J. Bai, Y.-H. Zou, and Y.-P. Wang, "Rozz: property-based fuzzing for robotic programs in ros," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[115] Y.-S. Hsiao, Z. Wan *et al.*, "Mavfi: An end-to-end fault analysis framework with anomaly detection and recovery for micro aerial vehicles," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023.

[116] S. Kim and T. Kim, "Robofuzz: Fuzzing robotic systems over robot operating system (ros) for finding correctness bugs," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022.

[117] B. Dieber, R. White *et al.*, "Penetration testing ros," *Robot Operating System (ROS)*, 2020.

[118] A. Ortega, N. Hochgeschwender *et al.*, "Testing service robots in the field: An experience report," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[119] A. Afzal, C. Le Goues *et al.*, "Mithra: Anomaly detection as an oracle for cyberphysical systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, 2021.

[120] C. Hildebrandt, S. Elbaum *et al.*, "Feasible and stressful trajectory generation for mobile robots," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020.

[121] A. Santos, A. Cunha *et al.*, "Property-based testing for the robot operating system," in *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 2018.

[122] Z. Han, T. Williams, and H. Yanco, "Mixed-reality robot behavior replay: A system implementation," in *AAAI Fall Symposium on Artificial Intelligence for Human-Robot Interaction (AI-HRI)*, 2022.

[123] B. Zahn, J. Fountain *et al.*, "Optimization of robot movements using genetic algorithms and simulation," in *RoboCup 2019: Robot World Cup XXIII 23*. Springer, 2019.

[124] R. D. Caldas, A. Rodrigues *et al.*, "A hybrid approach combining control theory and ai for engineering self-adaptive systems," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020.

[125] A. Afzal, D. S. Katz *et al.*, "Simulation for robotics test automation: Developer perspectives," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021.

[126] C. S. Timperley, A. Afzal *et al.*, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early?" in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018.

[127] V. Estivill-Castro, R. Hexel *et al.*, "Continuous integration for testing full robotic behaviours in a gui-stripped simulation." in *MoDELS (Workshops)*, 2018.

[128] C. Berger, "An open continuous deployment infrastructure for a self-driving vehicle ecosystem," in *Open Source Systems: Integrating Communities: 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30-June 2, 2016, Proceedings 12*. Springer, 2016.

[129] F. Roman, R. G. Maidana *et al.*, "Overseer: A multi robot monitoring infrastructure," in *International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2018, Brasil.*, 2018.

[130] D. Kirchner, S. Niemczyk *et al.*, "Rosha: A multi-robot self-healing architecture," in *Robot Soccer World Cup*. Springer, 2013.

[131] M. A. Hossen, S. Kharade *et al.*, "Care: Finding root causes of configuration issues in highly-configurable robots," *IEEE Robotics and Automation Letters*, 2023.

[132] M. G. Morais, F. R. Meneguzzi *et al.*, "Distributed fault diagnosis for multiple mobile robots using an agent programming language," in *2015 International Conference on Advanced Robotics (ICAR)*. IEEE, 2015.

[133] C. Tampier, A. Tiderko *et al.*, "Field test tool: automatic reporting and reliability evaluation for autonomous ground vehicles," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2022.

[134] P. Kaveti and H. Singh, "Ros rescue: fault tolerance system for robot operating system," *Robot Operating System (ROS) The Complete Reference (Volume 5)*, 2021.

[135] S. Silva, A. Bertolino *et al.*, "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*, 2022.

[136] C. Menghi, C. Tsigkanos *et al.*, "Mission specification patterns for mobile robots: Providing support for quantitative properties," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, dec 2022.

[137] J. Cohen, "A power primer," *Psychological Bulletin*, vol. 112, 1992.

[138] C. Sánchez, G. Schneider *et al.*, "A survey of challenges for runtime verification from advanced application domains (beyond software)," *Formal Methods in System Design*, vol. 54, 2019.

[139] B. Meyer, *Eiffel: The Language*. USA: Prentice-Hall, Inc., 1992.

[140] G. Dos Reis, J. Garcia *et al.*, "A contract design," *C++ Standards Committee Working Group ISOCPP*, 2016.

[141] M. Luckcuck, M. Farrell *et al.*, "A compositional approach to verifying modular robotic systems," *arXiv preprint arXiv:2208.05507*, 2022.

[142] M. Autili, L. Grunske *et al.*, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *IEEE Transactions on Software Engineering*, vol. 41, no. 7, 2015.

[143] S. Silva, P. Pelliccione *et al.*, "Self-adaptive testing in the field," *ACM Trans. Auton. Adapt. Syst.*, vol. 19, no. 1, feb 2024.

[144] C. Wohlin, P. Runeson *et al.*, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[145] S. Keele *et al.*, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[146] Q. Song, E. Engström, and P. Runeson, "Concepts in testing of autonomous systems: Academic literature and industry practice," in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021.

[147] M. A. Brito, S. R. Souza, and P. S. Souza, "Integration testing for robotic systems," *Software Quality Journal*, vol. 30, no. 1, 2022.

[148] J. Laval, L. Fabresse, and N. Bouraqadi, "A methodology for testing mobile autonomous robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013.

[149] A. Babić, G. Vasiljević, and N. Mišković, "Vehicle-in-the-loop framework for testing long-term autonomy in a heterogeneous marine robot swarm," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, 2020.

[150] M. Biagiola and P. Tonella, "Boundary state generation for testing and improvement of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2024.

[151] L. Gazzola, L. Mariani, M. Orrú, M. Pezzè, and M. Tappler, "Testing software in production environments with data from the field," in *IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2022.

[152] J. Riedesel, *Software Telemetry*. Manning Publications, 2021.

[153] D. Weyns, I. Gerostathopoulos *et al.*, "Guidelines for artifacts to support industry-relevant research on self-adaptation," *SIGSOFT Softw. Eng. Notes*, vol. 47, no. 4, sep 2022.

[154] D. Weyns, T. Bures, R. Calinescu, B. Craggs, J. Fitzgerald, D. Garlan, B. Nuseibeh, L. Pasquale, A. Rashid, I. Ruchkin, and B. Schmerl, "Six software engineering principles for smarter cyber-physical systems," in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, 2021.

[155] A. Bombarda, S. Bonfanti *et al.*, "Guidelines for the development of a critical software under emergency," *Information and Software Technology*, vol. 152, 2022.

Ricardo Caldas is a Ph.D. candidate at the Chalmers University of Technology in Sweden. His research topic is software design and assurance provision for resilient autonomous systems. He focuses on resilience as a key enabler to long-living autonomous systems by contributing to software architecture and testing for multi-agent heterogeneous applications, control-based self-adaptation, and automated explanation of violated properties. His contributions extend to software applications implemented in the robot operating system (ROS), in several domains such as robotics, autonomous driving, and healthcare. He received his master's degree in Computer Science from the University of Brasília in Brazil in 2019. More information is available at https://ricardocaldas.me.

Juan Antonio Piñera García is currently enrolled in the Ph.D. of Computer Science, at the Gran Sasso Science Institute (GSSI), in L'Aquila, Italy. His research topics focus in multi-robot mission coordination and self-adaptation, as well as Imitation Learning techniques based on Deep Learning. Previously, he was a Training Professor at the Computer Engineering Department, Technological University of Havana (Cuba) and a Research Fellow at the Group of Robotics and Mechatronics, Technological University of Havana (Cuba), where he co-led multiple projects to apply robotics to help solve everyday problems and was awarded multiple prizes at international robotics conventions, like the Robotic People Fest, in Colombia.

Matei Schiopu is a Ph.D. student at the Chalmers University of Technology in Sweden. His research interests include software engineering for cyber-physical and autonomous systems as well as applied machine learning. He received his Master's degree at the Chalmers University of Technology in 2024, working at Volvo Trucks on a machine learning research thesis together with the Battery Management System team.

Patrizio Pelliccione is a Professor in Computer Science at Gran Sasso Science Institute (GSSI, Italy) and Director of the Computer Science area. Patrizio is also adjunct professor at the University of Bergen, Norway. His research topics are mainly in software engineering, software architecture modeling and verification, autonomous systems, and formal methods. He received his PhD in computer science from the University of L'Aquila (Italy). Thereafter, he worked as a senior researcher at the University of Luxembourg in Luxembourg, then assistant professor at the University of L'Aquila in Italy, then Associate Professor at both Chalmers | University of Gothenburg in Sweden and University of L'Aquila. He has been on the organization and program committees for several top conferences and he is a reviewer for top journals in the software engineering domain. He is very active in European and National projects. In his research activity, he has collaborated with several companies. More information is available at http://patriziopelliccione.com.

Genaína Rodrigues is an associate professor with the Department of Computer Science at the University of Brasília. Her research interests include the mutual collaboration between smart autonomous systems engineering, in particular robotic systems, and software engineering mainly through goal-oriented requirements engineering and verification both at design and at runtime. She received her Ph.D. in the Computer Science from University College London. In 2020, she was awarded a fellowship as an experienced researcher through CAPES/Alexander von Humboldt Programme to conduct research at the Humboldt Universität zu Berlin in Germany. She has also served highly ranked conferences and journals in Software Engineering worldwide.

Thorsten Berger is a Professor in Computer Science at Ruhr University Bochum in Germany. His research focuses on automating software engineering for the next generation of intelligent, autonomous, and variant-rich software systems, exploring new ways of software creation, analysis, and evolution. He received the PhD degree in computer science from the University of Leipzig in Germany in 2013. Thereafter, he worked as a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark, and as an Associate Professor at Chalmers | University of Gothenburg in Sweden. He received a fellowship from the Royal Swedish Academy of Sciences and the Wallenberg Foundation, one of the highest recognitions for researchers in Sweden. He received two best-paper and two most influential paper awards, as well as his service was recognized with distinguished reviewer awards at A* conferences.