The Journal of Systems and Software 226 (2025) 112408

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

IoT systems testing: Taxonomy, empirical findings, and recommendations ☆

Jean Baptiste Minani [a,*], Yahia El Fellah [b], Fatima Sabir [c], Naouel Moha [b], Yann-Gaël Guéhéneuc [a], Martin Kuradusenge [d], Tomoaki Masuda [e]

[a] Concordia University, Montréal, QC, Canada
[b] École de Technologie Supérieure, Montréal, QC, Canada
[c] University of the Punjab, Lahore, Punjab, Pakistan
[d] University of Rwanda (UR), Kigali City, Kigali, Rwanda
[e] NTT Communications, Tokyo, Japan

ARTICLE INFO

Keywords:
IoT testing taxonomy
IoT systems testing taxonomy
Software engineering taxonomy
Quality assurance taxonomy
Testing approaches
Testing techniques
Application testing
System testing

ABSTRACT

The Internet of Things (IoT) is reshaping our lives, increasing the need for thorough pre-deployment testing. However, traditional software testing may not address the testing requirements of IoT systems, leading to quality challenges. A specific testing taxonomy is crucial, yet no widely recognized taxonomy exists for IoT system testing. We introduced an IoT-specific testing taxonomy that categorizes aspects of IoT systems testing into seven distinct categories. We mined testing aspects from 83 primary studies in IoT systems testing and built an initial taxonomy. This taxonomy was refined and validated through two rounds of surveys involving 16 and then 204 IoT industry practitioners. We assessed its effectiveness by conducting an empirical evaluation on two separate IoT systems, each involving 12 testers. Our findings categorize seven testing aspects: (1) testing objectives, (2) testing tools and artifacts, (3) testers, (4) testing stage, (5) testing environment, (6) Object Under Test (OUT) and metrics, and (7) testing approaches. The evaluation showed that testers equipped with the taxonomy could more effectively identify diverse test cases and scenarios. Additionally, we recommend new research opportunities to enhance the testing of IoT systems.

1. Introduction

Leotta et al. (2017) describe the Internet of Things (IoT) refers to network systems of physical devices that are connected and exchange data through the Internet. Cisco[1] predicted that IoT systems will include 500 billion devices by 2030, making computing power ubiquitous across IoT systems. Ahmed et al. (2019) highlighted that ensuring the proper functioning of IoT systems is crucial due to their direct impact on personal lives and public safety. Without proper testing, IoT systems may risk loss of life and financial resources, especially in safety-critical domains. Pontes et al. (2018) and Jean Baptiste et al. (2023b, 2024c) reported that proper testing in IoT systems is still challenging. This is due to their distributed nature, dynamism, and heterogeneity, as well as the multiple layers. Cisco, IBM, and Intel proposed a reference model with seven layers has been proposed (Inc. Cisco Systems, 2014). However, AltexSoft (2020) claims that there is no universally accepted IoT system architecture, as it varies based on business needs. Nevertheless, many IoT systems have four layers (Burhan et al., 2018; Abdullah et al., 2020; Rao and Haq, 2018; Touqeer et al., 2021): device, network, cloud, and application layer. Unlike traditional software, IoT systems require testing at all layers (White et al., 2017).

Jean Baptiste et al. (2024c, 2023b) identified the lack of a testing guide as one of the testing challenges for IoT systems testing among many other challenges that affect the quality of these systems. Mubarakah et al. (2020), Villalón et al. (2015a) mentioned that to improve testers' understanding of different testing aspects of IoT systems, a taxonomy can be a valuable guide. Taxonomy helps IoT testers understand and apply various testing aspects systematically, ensuring they do not overlook any aspect. By providing structured guidance, a taxonomy guides the testers to better understand different aspects of IoT systems testing. To the best of our knowledge, no taxonomy exists

☆ Editor: Antonia Bertolino.

* Corresponding author.
E-mail addresses: jeanbaptiste.minani@concordia.ca (J.B. Minani), yahia.el-fellah.1@ens.etsmtl.ca (Y.E. Fellah), fatima.sabir@pucit.edu.pk (F. Sabir), naouel.moha@etsmtl.ca (N. Moha), yann-gael.gueheneuc@concordia.ca (Y.-G. Guéhéneuc), kuradusenge@yahoo.com (M. Kuradusenge), tomoaki.masuda@sloan.mit.edu (T. Masuda).

[1] https://www.globenewswire.com/news-release/2014/10/14/1271271/0/en/The-Internet-of-Things-World-Forum-Unites-Industry-Leaders-in-Chicago-to-Accelerate-the-Adoption-of-IoT-Business-Models.HTML

https://doi.org/10.1016/j.jss.2025.112408
Received 19 August 2024; Received in revised form 2 January 2025; Accepted 24 February 2025
Available online 8 March 2025

for testing IoT systems. We want to propose an IoT-specific testing taxonomy that categorizes different aspects of IoT systems testing to guide IoT systems testers.

To achieve our objective, we reviewed 83 IoT testing-related primary studies (PSs) selected from 8 digital libraries. We followed existing guidelines for developing taxonomies by Kundisch et al. (2021), Ralph (2018), Usman et al. (2017). We improved the taxonomy by conducting surveys with IoT practitioners. We assessed its effectiveness by conducting an empirical evaluation using two systems as a case study with 12 testers each. We defined the following research questions (RQs):

- RQ1: What Are Testing Objectives?
- RQ2: What Are Testing Tools and Artefacts?
- RQ3: Who Is Responsible for Testing?
- RQ4: What Are Testing Stages?
- RQ5: What Are Testing Environments?
- RQ6: What Are Testing Approaches?
- RQ7: What Are Objects Under Test and Metrics?
- RQ8: How Does The Taxonomy Improve Testing?

The purpose of this taxonomy is twofold: 1. To provide a structured framework for organizing and understanding key dimensions of IoT systems testing to guide the practitioners. 2. To serve as a reference to relevant testing concepts for the IoT systems testers, helping the testing teams to work more efficiently and effectively.

This testing taxonomy categorizes and organizes testing aspects for improved clarity, benefiting researchers, practitioners, and future developments. This article extends our earlier workshop paper, which was accepted for publication in the proceedings of SERP4IoT'24 (Jean Baptiste et al., 2024b, 2023a). The abstract of this paper is also available as a preprint (Jean Baptiste et al., 2024d). [R3C3] The improvements to the previous work include a revised taxonomy that incorporates feedback from a second-round survey with 204 practitioners, detailed navigation guidelines, validation through experiments with testers, additional recommendations, and an expanded discussion. Building upon the contributions of our previous work, this paper provides the following new contributions: [R2C1]

- We validated and refined the taxonomy from our previous study (Jean Baptiste et al., 2024b) through collaboration with industry practitioners, incorporating valuable feedback from practitioners involved in IoT systems testing. Thus, the taxonomy aligns with real-world testing practices and effectively captures the nuances of testing IoT systems according to the 6Ws and 1H framework.
- We conducted an empirical evaluation with two case studies and 12 practitioners to assess the effectiveness of the developed taxonomy. This study aimed to evaluate the taxonomy's practical impact on testing IoT systems by gathering insights from testers directly involved in the process, validating its relevance and applicability in real-world scenarios.
- We provided recommendations on prioritized testing types tailored to each layer of IoT Systems, informed by the insights of industry practitioners.
- We set up two public access points for professionals to continuously access and stay updated with our IoT systems testing taxonomy. The first is hosted on the Ptidej website,[2] while the second is available in a GitHub repository,[3] ensuring that the latest version, incorporating newly identified aspects, is always accessible.

The rest of this article is organized as follows: Section 2 provides the motivational background and related work. Section 3 describes the research methodology. Section 4 presents practitioners' feedback. Section 5 discusses the taxonomy and answers our RQs, while Section 6 outlines the findings of the empirical evaluation. Section 7 summarizes the recommendations. Section 8 provides the discussion. Section 9 highlights potential threats to the validity of our study. Finally, Section 10 concludes the article and outlines future work.

[2] https://www.ptidej.net/Members/minanijb/Taxonomy/

[3] https://baptiste2k8.github.io/taxonomy4IoTTesting/

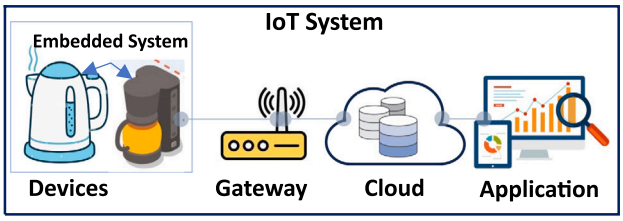


Fig. 1. IoT system key components.

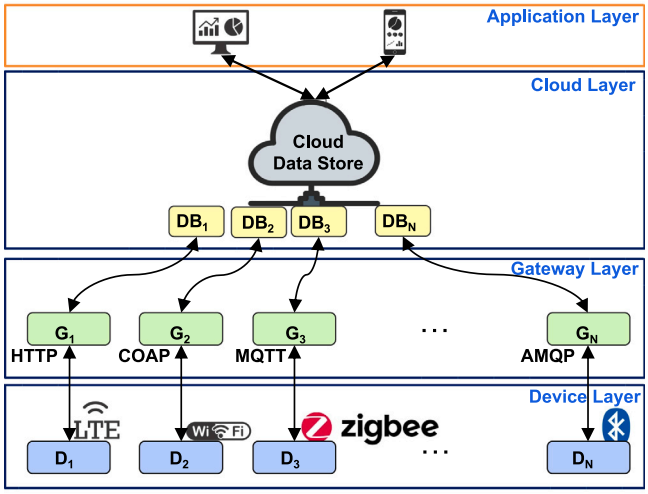


Fig. 2. Example of IoT system architecture.

2. Motivational background and related work

2.1. Motivational background

This section provides background on IoT system testing. First, we introduce the concept of IoT systems and their complexity. Then, we discuss how testing these systems necessitates a dedicated taxonomy that extends beyond existing taxonomies for testing traditional software.

2.1.1. Key concepts

IoT Systems. IoT systems are composed of several core components that contribute to their functionality and complexity. These components include a multitude of devices, gateways, cloud services, and applications. Fig. 1 shows the key components of the IoT system.

Each component plays a role in the system's overall operation. The diversity of devices involved, ranging from simple sensors to complex processors, and the variety of protocols they use, adds layers of complexity to the system architecture. Many papers refer to an IoT system as a network of devices, often emphasizing the devices and network layers while overlooking other components. To avoid potential confusion, in this paper, the term IoT system specifically refers to what is commonly understood as an **IoT application**. We define an IoT application as a software system designed to manage, process, and use data collected from various IoT devices as shown in Fig. 2.

Embedded Systems. Embedded systems are designed to perform specific, standalone tasks within hardware devices, typically without native internet connectivity. An example of this is a microwave oven's control system, which operates independently to manage cooking times

and power levels based on user input. In the context of IoT systems, all IoT devices run on embedded systems. By adding a communication layer, which enables an IoT device to interact with other devices, a cloud layer for storage, and an application layer, we transition from an isolated embedded system to an IoT system. The application layer in an IoT system can take the form of a web application, mobile application, or desktop application, providing a user interface to manage and interact with the IoT devices.

Traditional Software Systems. Traditional software systems are characterized by a centralized architecture in which client applications—ranging from desktop software to mobile and web-based applications—interact with central servers responsible for processing requests, managing data storage, and executing critical operations.

Unlike traditional systems, where communication primarily occurs between client applications and central servers, IoT systems can interconnect multiple devices over the Internet. This connectivity enables them to collect, process, and transfer data amongst themselves or to cloud storage or backend servers as part of a distributed system. While not all embedded systems are part of IoT, those that are connected and networked can be considered a subset of IoT systems.

In this study, the term "System Under Test (SUT)" refers to the entire IoT system, including application layer, device layer, communication layer, and cloud layer. "Object Under Test (OUT)" denotes a specific component of the IoT system being tested such as device layer, application layer, cloud layer, or communication layer. In the next section, we discuss related work, primarily focusing on testing traditional systems or addressing limited aspects of IoT systems.

2.2. Related work

Table 1 summarizes the related work and compares them with our study based on the testing aspects covered. [R2C7] We specifically considered studies that explicitly mention taxonomies or classify aspects of IoT system testing, such as testing types, approaches, tools, or other dimensions closely aligned with the scope of our study, while excluding general studies on IoT system testing. In this paper, we use **testing aspects** to refer to any concept that can be considered when testing a given system. It encompasses the rationale behind testing, the environments for conducting tests, approaches used, appropriate timing for testing, the individuals responsible, tools and metrics used, and the artifacts generated.

Several studies proposed taxonomies for software testing to guide the testing teams. Villalón et al. (2015a) discussed a taxonomy featuring 9 overarching categories and 27 subcategories tailored for traditional software. This taxonomy underwent validation via a comprehensive survey involving IT managers and industry professionals, cementing its relevance and utility within the field. However, this taxonomy does not cover any aspect related to IoT systems testing. Vegas et al. (2009) presented a taxonomy for unit testing of conventional software systems. It provides 13 testing techniques, including methodologies such as random testing, boundary value analysis, statement testing, branch testing, path testing, thread testing, and mutation testing. This taxonomy may not suffice for testing IoT systems since it discusses only testing techniques and does not cover other aspects such as testing environment, testing tools, and items to be tested. Unterkalmsteiner et al. (2014) introduced taxonomy for requirements engineering and software testing (REST). It focuses on aligning software requirements and resultant software. The taxonomy was validated through an industry survey. This validation highlighted its potential to enhance both requirement engineering and software testing processes. However, its focus is limited to a few aspects of IoT systems and may not fully serve the needs of testing IoT systems. A distinct perspective emerged in the work of Cheverda et al. (2022), where the authors proposed a taxonomy to evaluate software quality, focusing on metrics. This taxonomy effectively addressed eight attributes (compatibility, portability, functional sustainability, security, usability, performance, maintainability, and reliability) fundamental to traditional software systems. However, Khezemi et al. (2024) proposed different quality attributes for IoT systems. Nevertheless, this taxonomy is missing many aspects of IoT systems such as guiding practitioners to know what to test, how to test, where to conduct the test, and when to test. Mubarakah et al. (2020) introduced a taxonomy based on SWEBOK analysis, emphasizing ten knowledge areas. Although the study emphasized software testing, it lacked details on test levels and techniques. Makhshari and Mesbah (2021b) focused on taxonomy for categorizing IoT bugs but did not address actual IoT system testing. Yet Raibulet (2018) focused on taxonomy for software evaluation. This study attempted to address "How" and "What" aspects of software testing and did not consider other aspects such as *stage of testing*, *environment for testing*, and *objective of testing*.

Ladisa et al. (2023) provided a taxonomy for evaluating open-source software. However, this study did not address any aspect of IoT systems testing. Zander and Schieferdecker (2011), Felderer et al. (2016) focused on taxonomy for model-based testing. These studies concentrated on traditional software and did not consider the unique characteristics of IoT systems. Costa et al. (2020), Roggio et al. (2014), Kiran Bhagnani (2014) presented taxonomies related to performance testing tools, testing terminologies, and testing techniques. While these taxonomies can be crucial in testing IoT systems, their coverage is limited to a few aspects of software projects in general, without considering the specific needs of testing IoT systems such as testing devices and connectivity. Coppola and Alégroth (2022) discussed the taxonomy of software metrics, while Mubarakah et al. (2020) provided the taxonomy for software engineering tools and methods. However, none of these taxonomies can address fully the need for testing IoT systems. The ISO-29119 (ISO Standards, 2021) provides techniques for testing traditional software systems. However, testing IoT systems demands a broader approach that accounts for their distinctive attributes. It is essential to adapt and expand these standards to address the testing needs specific to IoT systems. Firesmith (2015) explored the taxonomy of testing types for a software project. While this taxonomy is detailed, it falls short in addressing IoT system aspects. Despite this limitation, our study draws inspiration from Firesmith (2015) for its comprehensive coverage, even though it does not explicitly consider the testing needs of IoT systems. We draw inspiration from this previous work and explain in the following section why a taxonomy dedicated to testing IoT systems is necessary. [R3C7] Yaqoob et al. (2017) provided a comprehensive taxonomy of IoT architectures and devices, which complements our focus on testing by offering insights into the structural aspects of IoT systems.

[R3C8]Finally, Usman et al. (2017) proposed various approaches to develop and validate the taxonomy, but did not propose any taxonomy. Table 2 presents a comparison of our study with related works that proposed and validated taxonomies.

In addition to validation approaches, related studies have also introduced various validation metrics. Table 3 highlights the metrics we used in our validation and compares them with those used in other studies. We notice that many studies used perception-based metrics, such as usefulness, completeness, understandability, and clarity, to assess the quality of taxonomies (Villalón et al., 2015b).

2.3. Definition of validation metrics

[R2C2][R3C2] To validate this taxonomy, we used several metrics, as presented in Table 3. This section provides definitions for these metrics.

- Completeness: The degree to which a taxonomy covers all relevant concepts and subtopics.
- Coverage: The extent to which a taxonomy addresses the breadth of topics and subtopics within its scope

Table 1
[R2C7] Related works focusing on taxonomy in software engineering.

Study	Aim of the Study	Year	Aspects of the Study							
			What	How	When	Where	Which	Who	Why	IoT
Ladisa et al. (2023)	Taxonomy of attacks	2023	-	✝	-	-	-	-	-	No
Lonetti et al. (2023)	MB Security Testing	2023	-	✝	-	-	-	-	-	Yes
Fadhil and Sarhan (2022)	IoT Testing Survey	2022	-	✝	-	-	-	-	-	Yes
Coppola and Alégroth (2022)	Testing metrics	2022	†	-	-	-	-	-	-	No
Cheverda et al. (2022)	Taxonomy for quality assessment review	2022	-	-	-	-	-	-	-	No
Makhshari and Mesbah (2021b)	Taxonomy of bugs	2021	-	-	-	-	-	-	-	Yes
Costa et al. (2020)	Performance Testing Tools	2020	-	†	-	-	†	-	-	No
Mubarakah et al. (2020)	Taxonomy reviews	2020	-	✝	-	-	†	-	-	No
Raibulet (2018)	Taxonomy of software evaluation approaches	2018	-	†	-	-	-	-	-	No
Garousi et al. (2018)	Testing taxonomy for embedded systems	2015	†	†	-	-	†	-	-	No
Firesmith (2015)	Testing Types	2015	✚	✚	†	✚	✚	✓	✓	No
Villalón et al. (2015a)	Software testing	2015	†	†	-	-	†	-	-	No
Engström and Petersen (2015)	Testing interventions and practical challenges in SE	2015	-	-	-	-	-	-	-	No
Felderer et al. (2016)	Classification of Model-Based Security Testing	2015	†	-	-	-	†	-	-	No
Roggio et al. (2014)	Software testing Terminologies	2014	-	†	-	-	-	-	-	No
Kiran Bhagnani (2014)	Taxonomy for testing techniques	2014	-	†	-	-	-	-	-	No
Unterkalmsteiner et al. (2014)	Taxonomy for RE and software testing	2014	-	-	-	-	-	-	-	No
Utting et al. (2012)	Taxonomy for MBT approaches and tools	2012	-	†	-	-	†	-	-	No
Zander and Schieferdecker (2011)	MBT for embedded systems	2011	-	-	-	-	†	-	-	No
Zander et al. (2011)	Test generation, execution, and evaluation	2011	†	†	-	-	-	-	-	No
Vegas et al. (2009)	Testing Techniques	2009	-	†	-	-	-	-	-	No
This	Testing IoT systems	2024	✓	✓	✓	✓	✓	✓	✓	Yes

✹-:Not Covered[0%]; ✝:Limited Coverage[25%]; †: Partially Covered[50%];✚:Mostly Covered[75%];✓:Comprehensively Covered.
✹ What - Item Under Test and Metrics; How - How to Test; When - When to Test; Where - Testing Environment; Which - Which Tools and Artefacts; Who - Who Does Test; Why - Objective of Testing.

Table 2
Taxonomy validation approach.

Validation Approach	Studies					
	This	Mountrouidou et al. (2019)	Coppola and Alégroth (2022)	Costa et al. (2020)	Felderer et al. (2016)	Villalón et al. (2015b)
Comparision (Benchmarking)	- -	++	- -	- -	- -	- -
Expert Opinion (Survey)	++	- -	- -	- -	- -	++
Case Study	++	- -	- -	- -	- -	- -
Experiment	++	++	- -	- -	- -	- -
Classifying Existing Literature	++	- -	++	++	++	++

✹ ++: Validation Mentioned. –: Not Mentioned.

Table 3
Taxonomy validation metrics.

Validation Metrics	Studies					
	This	Mountrouidou et al. (2019)	Coppola and Alégroth (2022)	Costa et al. (2020)	Felderer et al. (2016)	Villalón et al. (2015b)
Completeness	++	++	- -	- -	- -	++
Coverage	++	- -	- -	- -	- -	- -
Effectiveness	++	- -	- -	- -	- -	- -
Expectation-Matching	++	- -	- -	- -	- -	- -
Helpfulness	++	- -	- -	- -	- -	- -
Importance	++	- -	- -	- -	- -	- -
Level of Details	++	- -	- -	- -	- -	- -
Understandability	++	- -	- -	- -	- -	++
Usefulness	++	- -	- -	- -	- -	++
Precision	- -	++	- -	- -	- -	- -
Timelessness	- -	++	- -	- -	- -	- -
Cohesion	- -	- -	- -	- -	- -	- -
Clarity	- -	- -	- -	- -	- -	++

✹ ++: The metric was used. –: Metric was not used.

- Effectiveness: The degree to which a taxonomy helps users achieve their goals or complete tasks efficiently.
- Expectation-Matching: The extent to which a taxonomy meets users' preconceived notions and expectations about its structure and content.
- Helpfulness: The degree to which a taxonomy provides useful guidance, clarification, or insights to users.
- Importance: The perceived significance and relevance of a taxonomy to users' needs and goals.
- Level of Details: The perceived granularity and specificity of information provided by a taxonomy.
- Understandability: The ease with which users can comprehend and interpret the concepts, relationships, and terminology presented in a taxonomy.

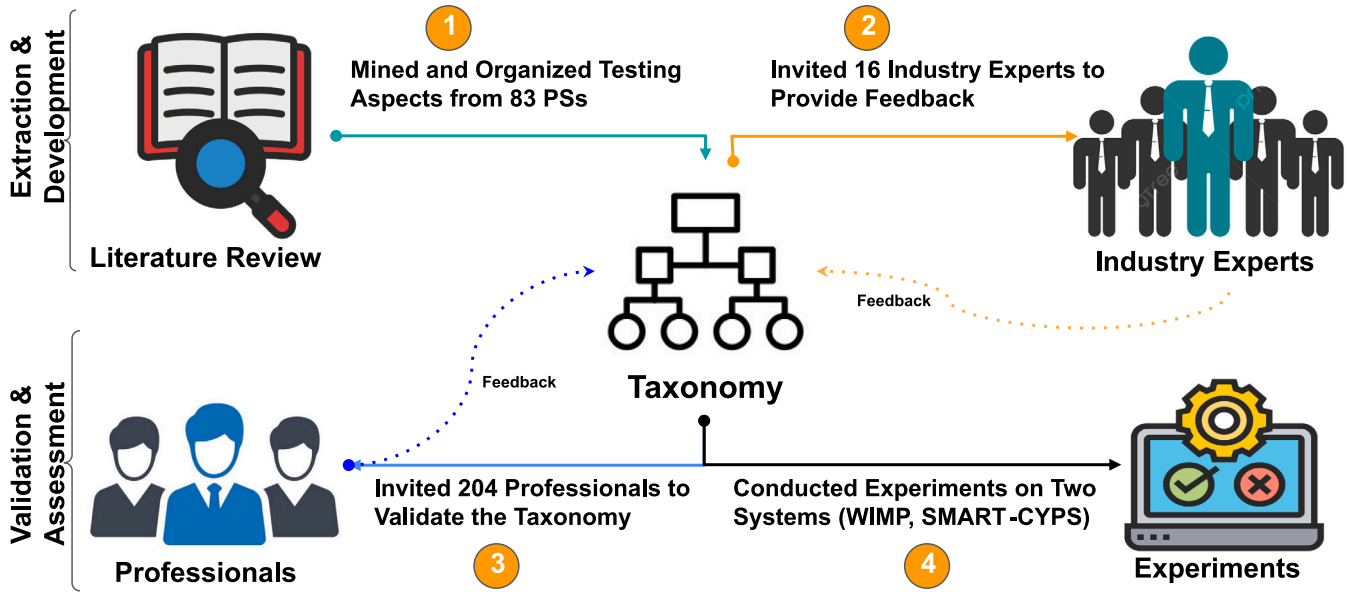


Fig. 3. Research methodology.

- Usefulness: The degree to which a taxonomy is perceived by users as being practical, applicable, and beneficial for their specific needs and tasks.

2.4. Need for IoT system testing taxonomy

Testing IoT systems presents unique challenges not encountered in traditional software testing. Due to the heterogeneous nature of the devices and protocols, testing must go beyond application to also include the devices themselves, data processing and storage, their connectivity, interoperability, and dealing with different communication protocols. Furthermore, the dynamic nature of IoT environments, where devices frequently connect and disconnect and systems scale dynamically, complicates the testing process even further.

The complexity of interactions across different layers and components demonstrates that traditional software testing taxonomies may not be sufficient. Traditional methods do not fully address the specific needs of IoT systems, such as device connectivity, diverse communication protocols, and dynamic conditions. Therefore, we need a customized IoT testing taxonomy that comprehensively covers these areas. The taxonomy proposed by Firesmith (2015) addresses the needs of traditional systems while the testing techniques proposed in ISO-29119 (ISO Standards, 2021) can only address "WHAT" aspect of testing traditional systems without addressing the specific needs of IoT systems. Although the taxonomy in Firesmith (2015) and the testing techniques in ISO-29119 (ISO Standards, 2021) provide useful information for testing in general, they do not address the specific needs of IoT systems. Therefore, to the best of our knowledge, none of the previous studies provided a comprehensive taxonomy for end-to-end IoT systems testing. In our study, we analyzed IoT testing literature and developed an IoT system testing taxonomy. We improved it with inputs from 16 industry practitioners, and 204 IoT professionals to ensure its alignment with industry needs and bridge the gap in IoT system testing. We assessed its effectiveness with empirical evaluation on two systems involving 12 testers each.

3. Research method

The method used in this study comprises three phases, as shown in Fig. 3: taxonomy development, feedback collection, and empirical evaluation.

3.1. Taxonomy development

In this section, we explain the steps to construct the taxonomy and answer RQ1-RQ7.

3.1.1. Step 1 – Identification of relevant literature

We followed the Preferred Reporting Items for Systematic Reviews and Meta-Analyses statement (PRISMA) by Page et al. (2021) and retrieved relevant scientific articles published from January 2012 until December 2022. In this section, we have reused some data from our recent study on a systematic literature review (SLR) for IoT system testing, specifically for the identification and selection of primary studies (PSs) (Jean Baptiste et al., 2024c). Specifically, we extracted data pertinent exclusively to the development of our taxonomy. We present only the key steps here to focus on the essential aspects of our methodology. Fig. 4 summarizes the PRISMA process we used for the literature review. We started by defining the search strategy to be executed in different digital libraries.

We defined the following search strategy by applying the Population, Intervention, Comparison, and Outcome process (PICO) proposed by Cooke et al. (2012).

> [IoT OR internet of thing OR IoT system OR internet of thing system OR IoT platform OR internet of thing platform OR IoT application OR internet of thing application OR IoT software OR internet of thing software] AND (test OR bug OR defect OR failure OR anomal* OR quality OR verification OR validation) AND (method OR technique OR approach OR process OR type OR level OR practice OR tool OR framework OR layer OR component OR constituent OR attribute OR metric OR objective OR stage OR phase OR environment OR target OR artifact OR artefact)

We selected 8 online digital libraries: ACM Digital Library, Compendex, IEEE Xplore, ScienceDirect, SpringerLink, Scopus, Web of Science, and Wiley. Those digital libraries are the most commonly used for literature reviews in software engineering (Dyba et al., 2007). We executed the search strategy and retrieved 8,294 articles from these 8 digital libraries. We removed 985 duplications and we obtained 7,305 articles. We screened these using the following inclusion and exclusion criteria.

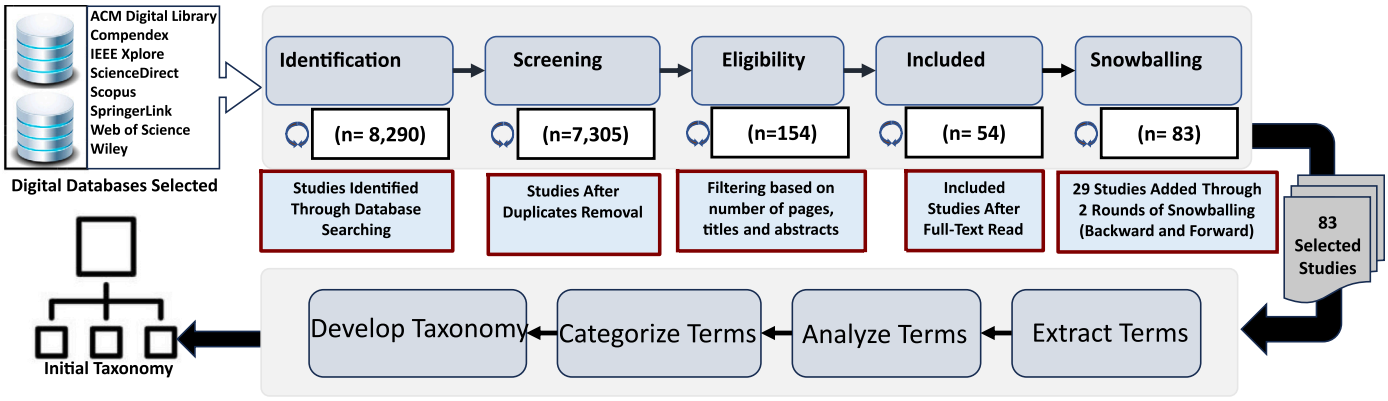


Fig. 4. Initial taxonomy construction process.

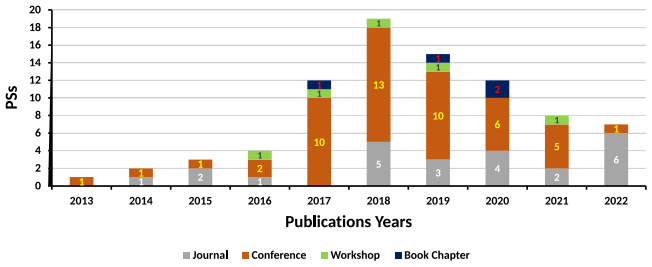


Fig. 5. Distribution of PSs by Year.

Inclusion Criteria. We considered the following inclusion criteria for article selection:

- The article is written in English
- The article is published between 2012 and 2022
- The article is published in a journal, conference, or workshop
- The article explicitly discusses at least one aspect of IoT systems testing
- The article has at least 4 pages

Exclusion Criteria. We considered the following exclusion criteria:

- The article has less than 4 pages
- The article is not a primary article
- [R3C7] The article is not related to security testing
- The article has not been peer-reviewed
- The article is a graduate thesis or project report
- The article does not have its full text available online

[R3C7] We chose to exclude PSs about IoT security testing despite the importance of this topic for two reasons. First, this topic deserves its own study and has already been the subject of recent surveys, e.g., Nawir et al. (2016). Second, security testing for IoT systems is a vast and complex topic, which would have overshadowed other objectives and increased the complexity and length of this article. We applied these criteria, and we obtained 54 articles.

We performed two rounds of backward and forward snowballing and identified an additional 29 studies, bringing the total to 83 studies.

Fig. 5 shows the publication trends of reviewed articles.

3.1.2. Step 2 – Data extraction and analysis

Two authors of this paper manually analyzed the selected studies and mined different testing aspects to develop the initial taxonomy, resolving any disagreement on extracted aspects through discussion sessions to reach a consensus. Table 4 shows the form used for data extraction.

Table 4
Data extraction form.

Field	Description
Study	The title of the study (paper)
Focus	The main focus of the study (paper)
Aspect	Any testing aspect from testing objectives/reasons, testing approaches, testing environment, testing tools, testing metrics, testing artifacts, testing stage, testing responsibility

The studies analyzed and the data extraction form we used can be found online on ptidej website.[4] and Zenodo[5]

3.1.3. Step 3 – develop the initial taxonomy

To develop the initial taxonomy, we followed the method proposed by Usman et al. (2017). This method consists of 13 activities grouped into 4 phases, as shown in Table 5.

(1) *Planning.* This step consists of defining the initial aspects of the taxonomy to be developed, such as the objective of the taxonomy, the software engineering knowledge area associated with this taxonomy (i.e., the object to be classified), the data collection method, the classification structure type (e.g., tree or facet-based), and the classification procedure type (i.e., qualitative or quantitative). In this phase, we conducted six activities (B1 to B6). The knowledge area associated with the designed taxonomy is *testing* (the outcome of **B1**). The main objective of the taxonomy is *to identify and classify the testing aspects reported in the existing literature* (the outcome of **B2**). The subject matter of the taxonomy is *IoT system testing* (the outcome of **B3**). We chose *hierarchy* as the classification structure (the outcome of **B4**) to relate categories and sub-categories in a parent–child relationship as proposed by Kwasnik (1999). We selected a *qualitative* procedure to classify testing terms (the outcome of **B5**) according to Wheaton (1968). We used a *systematic literature review* to identify testing aspects (the outcome of **B6**).

(2) *Identification and Extraction.* We carried out 2 activities (B7 and B8). We extracted all testing terms from the *83 PSs* (the outcome of **B7**). To extract the testing terms from 83 PSs, we followed the steps in Section 3.1.1 and Section 3.1.2. Activity **B8** enabled us to control the consistency of extracted terms. We performed the following actions to categorize the extracted terms: (1) We removed duplicates, (2) whenever we identified a testing term possibly represented by another term in different

[4] https://www.ptidej.net/downloads/replications/jss24a/
[5] https://zenodo.org/records/14515044

Table 5
Taxonomy design method [Adapted from Usman et al. (2017)'s study].

Phase	ID	Activity
Planning	B1	Define SE knowledge area of the study
	B2	Describe the objectives of the taxonomy
	B3	Select and describe the subject matter to be classified
	B4	Select classification structure type
	B5	Select classification procedure type
	B6	Identify the sources of information
Identification and Extraction	B7	Extract all the terms
	B8	Perform terminology control
Design and Construction	B9	Identify and describe the taxonomy dimensions
	B10	Identify and describe the categories of each dimension
	B11	Identify and describe the relationships
	B12	Define guidelines for using and updating the taxonomy
Validation	B13	Validate the taxonomy

✺ **SE**: Software Engineering.

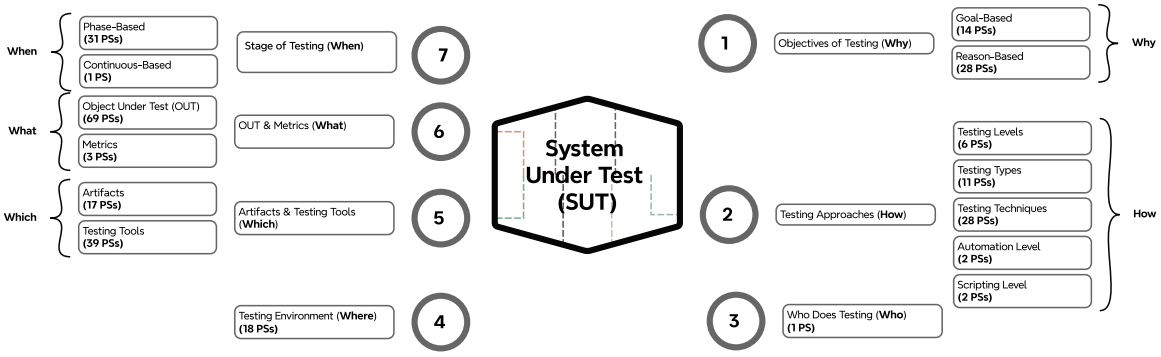


Fig. 6. Testing aspects from primary studies.

studies with the same meaning, we performed terminology unification. We used a Delphi-inspired process (Niederberger and Spranger, 2020), where two authors of this paper independently define the category of the extracted terms, resolving any conflict through discussions until a consensus was reached. As a result, we identified testing aspects shown in Fig. 6.

(3) *Design and Construction.* In this phase, we described the categories of testing aspects. We identified any relevant subcategories for each testing aspect. We used extracted terms and controlled through B7 and B8 to identify and describe taxonomy dimensions and categories. We selected one dimension, named *IoT system testing* (the outcome of **B9** at the very top, making it the parent of the entire taxonomy. We identified *7 aspects (objectives, targets, approaches, timing, environment, role, and tools and metrics*, each representing a distinct category within the taxonomy, as shown in Fig. 6 (the outcome of **B10**). We used hierarchical classification relationships to relate categories and subcategories (the outcome of **B11**). We provided a guideline through the usage scenario in Section 3.5, and we will review the taxonomy every two years to determine if updates are necessary. (the outcome of **B12**).

(4) *Validation.* [R3C8] Usman et al. (2017) proposed various approaches for validating taxonomies, including expert opinion, case studies, experiments, and benchmarking. In this study, we validated this taxonomy with two rounds of surveys (i.e., expert opinion), as explained in Sections 3.2 and 3.3. We also conducted an empirical study with testers (i.e., case study and experiment), as presented in Section 6.

3.2. Survey – Round 1

We surveyed 16 industry practitioners to collect feedback on the initial taxonomy. We used the collected feedback to improve the taxonomy. Fig. 7 summarizes the steps we used to conduct the survey.

1. *Identify the Participants*. Due to limited information on the version of the taxonomy we had, we chose not to share it publicly for feedback. Instead, we identified professionals with experience in IoT systems through their LinkedIn profiles. We invited them to participate in our study and provide their feedback on the version of the taxonomy that we constructed based on a literature review. We did not specify a target number of professionals; however, we sent requests via LinkedIn private messages to 79 professionals. Ultimately, 16 practitioners voluntarily agreed to participate.
2. *Design the Feedback Collection Form.* We created a feedback collection form consisting of 20 questions grouped into three sections: demographic information, IoT testing experience, and comprehension of the taxonomy. The form can be accessed online.[6]
3. *Send out the Initial Taxonomy and Feedback Form.* We used practitioners' email addresses to share the taxonomy and the link to access the feedback form.
4. *Collect and Analyze the Feedback.* We analyzed the feedback from experts to extract their recommendations.
5. *Update the Taxonomy.* We updated the taxonomy by adding more terms or by removing redundancies.

3.3. Survey – Round 2

We surveyed 204 practitioners to evaluate various aspects of the developed taxonomy and provide their feedback. We used the feedback provided to improve and finalize the taxonomy. We conducted three

[6] https://tinyurl.com/surveyform2024

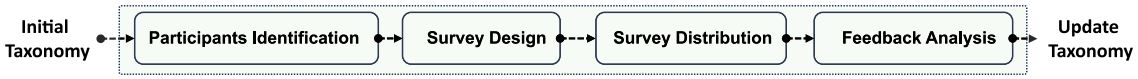


Fig. 7. Steps used to conduct survey.

main activities: designing the survey, distributing the survey and taxonomy, and analyzing the survey results. We followed the same steps as presented in Fig. 7.

- *Design the Survey.* We used Google Forms to create our survey. We asked participants questions on completeness, understandability, helpfulness, meeting user expectations, effectiveness, overall impact for practitioners and stakeholders, and participants' opinions on the categories and subcategories. The link to the survey is online.[7]
- *Distribute the survey.* We obtained an "Ethics Certificate" for research involving human subjects. At this stage, our taxonomy was more developed, and we were receptive to public feedback. We aimed to gather feedback from at least 100 professionals specializing in IoT systems. We distributed the Survey, along with a copy of the taxonomy, via various social media channels, mainly LinkedIn and Facebook IoT groups. We also contacted alumni associations from VIT,[8] CMU,[9] UR-ACEIoT.[10] We did not provide any monetary incentives for the participants and requested those willing to participate to read the taxonomy and ask questions, if any, before taking the survey. The survey was conducted over a period of 45 days, during which 204 practitioners agreed to review the taxonomy and provide their feedback.
- *Analyze the Survey Results.* We analyzed the survey data to finalize the taxonomy.
  The materials we used can be accessed from ptidej website.[11] The complete taxonomy can be accessed online.[12]

3.4. Taxonomy validation

We used two approaches to validate the taxonomy: survey-based validation and empirical study.

3.4.1. Survey-based validation

We validated the taxonomy by surveying 16 and 204 practitioners as described in 3.2 and 3.3 respectively. In addition to the feedback they provided to enhance the taxonomy, we asked the practitioners to assess the taxonomy for its completeness, understandability, level of detail, etc. We analyzed the assessment data provided by practitioners to improve the initial taxonomy.

3.4.2. Case study

We conducted a case study on two separate systems to evaluate how the taxonomy can improve testers' understanding (RQ8). We recruited software engineering students from the IoT lab at Concordia University for the WIMP project, and IoT students from the Center of IoT at the University of Rwanda for the SMART-CYPS project. We asked them to complete a questionnaire providing basic information such as their experience with software projects, their understanding of IoT systems (ranging from basic to advanced), and their educational level (minimum requirement: bachelor's degree). Subsequently, we hired 12 students for each system as part of our case study. [R2C3] We chose these students for both systems because (1) they had some experience in developing or testing software solutions, and (2) they had direct access to the respective systems. For each system, we used two groups, each consisting of 6 participants. We asked each group to develop test cases and scenarios for the given IoT systems. One group had access to the taxonomy, while the other did not. Before the experiment, we explained the systems to both groups and provided the necessary system documentation along with a system demonstration. The task lasted for 2 h. We compared the number of test cases (TCs), scenarios, and aspects identified by each group.

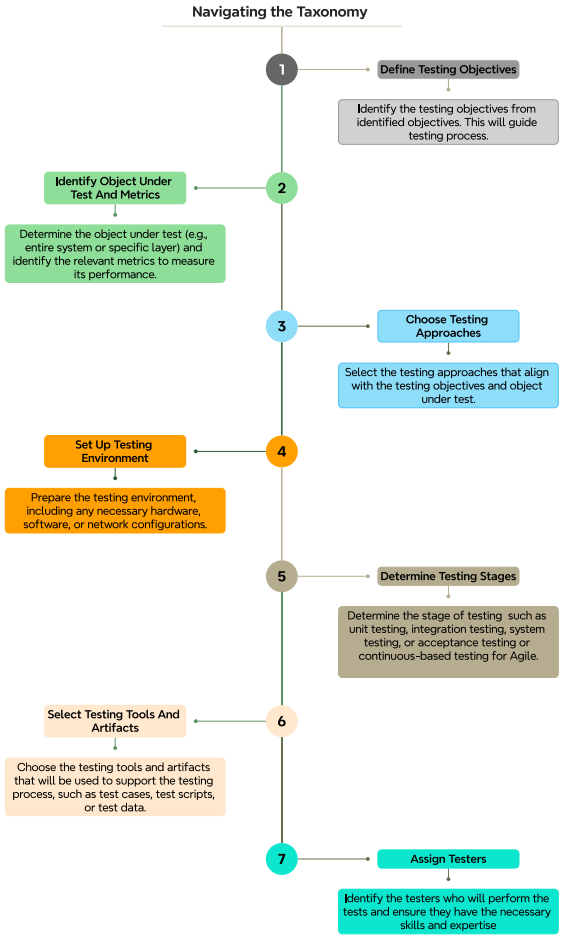


Fig. 8. Steps to guide the practitioners.

3.5. Practitioners guidance

Although the taxonomy offers options for each aspect, testers are not required to consider all of them. However, for each category, testers must make decisions based on what to test, the objectives, etc. Fig. 8 illustrates how testers can navigate the taxonomy.

Fig. 9 shows a (simplified) example for practitioners to navigate the taxonomy. Although we do not depict all the potential options outlined in the taxonomy, it shows that practitioners can select one of the four identified options for the testing environment to execute their tests. Regarding the testing approach, the scripting level is optional and requires the use of automation tools. It cannot enumerate all

[7] https://tinyurl.com/TaxonomyFeedback204
[8] https://vit.ac.in/
[9] https://www.cmu.edu/
[10] https://aceiot.ur.ac.rw/
[11] https://www.ptidej.net/downloads/replications/jss24a/
[12] https://www.ptidej.net/Members/minanijb/Taxonomy/

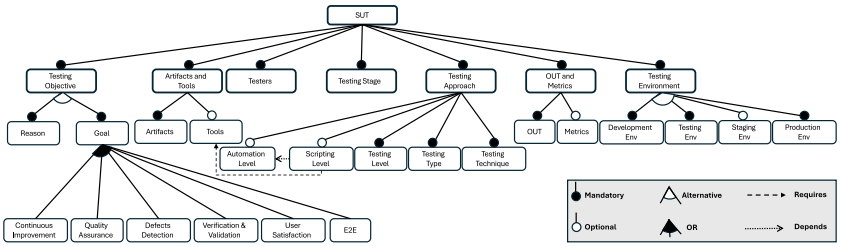


Fig. 9. Simplified example for practitioners to navigate the taxonomy.

conceivable metrics or test automation tools, but guides practitioners in making a choice based on their applicability to their SUT.

3.6. Usage scenario: Developing a test strategy for a smart home IoT system

We further illustrate the use of the taxonomy with the following scenario. A tester is given a task to ensure the quality and reliability of a smart home system. This system includes interconnected devices, such as thermostats, security cameras, and lighting controls, all managed through a central hub. The tester can use the taxonomy to answer the following questions:

1. *Test Objective – "Why".* The primary objective is to ensure the seamless functionality, security, and interoperability of the smart home system.
2. *What to Test – "What".* The tester focuses on IoT devices, networks, and applications and some characteristics: (a) Functionality: Test the core functionalities of each IoT device, communication layer, and automation testing of the application. (b) Security: Assess the system security measures, including data encryption, user authentication, and protection against cyber threats. (c) Interoperability: Verify interoperability between different devices and protocols to ensure smooth communication and compatibility.
3. *How to Test – "How".* The tester can choose among different options:
   (a) Manual Testing: Perform manual tests to validate user interfaces and user experience with the application, and interoperability between devices. (b) Automated Testing: Implement automated tests for repetitive tasks, e.g., unit tests, for all devices and the application. (c) Penetration Testing: Conduct penetration tests to identify vulnerabilities and weaknesses in the security.
4. *Who Does Test – "Who".* The tester can enlist the help of:
   (a) QA Team: Testers responsible for executing test cases, documenting defects, and ensuring overall test coverage. (b) Development Team: Developers responsible for implementing requirements, unit tests, and bug fixing. (c) Security Experts: Colleagues who could conduct in-depth security assessments and identify potential vulnerabilities.
5. *Where to Conduct the Test – "Where".* The tester can choose between:
   (a) Testing Environment: Set up a dedicated test environment mirroring the production environment to simulate real-world scenarios. (b) Production Environment: Perform on-site testing in real homes to assess device functionality and user experience using the production servers.
6. *When to Test –"When".* The tester can use and combine:
   (a) Iterative Testing: Conduct testing throughout the development lifecycle, including early-stage testing, integration testing, and regression testing. (b) Patch (or Pre-release) Testing: Intensify testing efforts before major releases to identify and rectify critical issues. (c) Maintenance (or Post-deployment) Testing: Continue testing post-deployment to monitor system performance, address user feedback, and release updates.

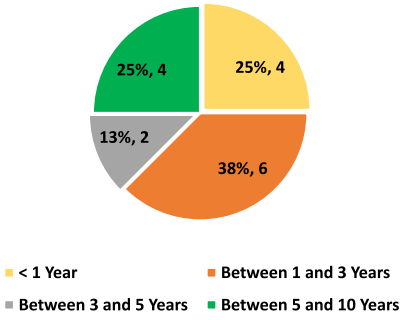


Fig. 10. Participants' experience in IoT.

7. *Which Tools and Artifacts – "Which".* The tester can choose:
   (a) Automation Tools: Appium and Selenium for automated testing of web interfaces and mobile apps, respectively. (b) Artifacts: Bug reports or TCs.

The testers are not expected to check all the boxes within each category. However, depending upon the nature of the project or its priorities, testers may select applicable elements from each category, guided by the defined objectives of the testing endeavour.

4. Practitioners' review

We conducted two rounds of surveys with industry practitioners to collect their reviews on this taxonomy, and we used their feedback to improve it.

4.1. Round 1

We surveyed 16 industry practitioners (12 males and 4 females) from 7 countries (UAE, USA, Canada, Japan, Pakistan, India, and Rwanda), to collect feedback on our initial taxonomy. 25% of participants have 5 to 10 years of experience in IoT, while 12.5% have 3 to 5 years of experience. The majority, 62.5%, have 1 to 3 years of experience in IoT. Fig. 10 shows the details of the participants.

We used the feedback provided by 16 experts to improve the initial taxonomy. The participants evaluated the taxonomy, and we summarized their evaluation results in Fig. 11. To ensure participant anonymity, we assigned unique identifiers (P1 to P16) to represent each of the 16 participants in this paper. We consolidated "strongly agree" and "agree" into "agree", then aggregated their corresponding values. For instance, combining 43.75% strongly agreeing with 12.50% agreeing results in a total of 56.25% agreement.

4.1.1. Completeness, helpfulness, understandability, and usefulness

We asked the practitioners to assess the completeness, understandability, usefulness, helpfulness, expectation matching, importance, and effectiveness of the taxonomy on a scale of 1 (strongly disagree) to 5 (strongly agree) as shown in Fig. 11.

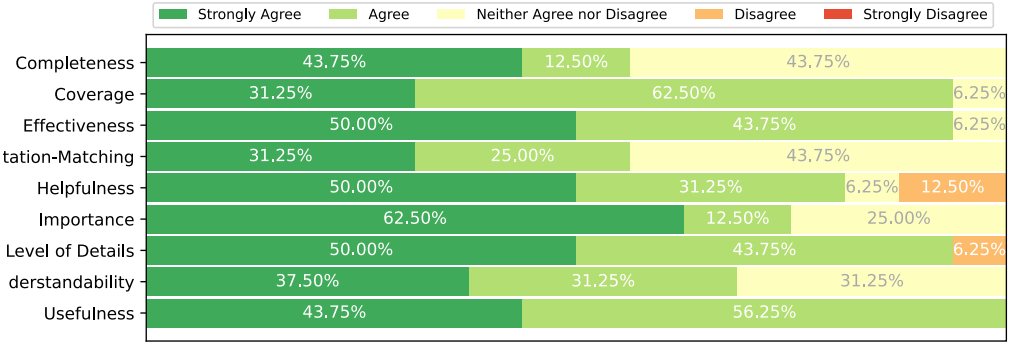


Fig. 11. Evaluation results - Round 1.

Completeness. 56.25% of the participants agreed that our taxonomy is complete. The comment received from the participants was to include "security" on testing types, and "business scenario (E2E)" as the goal of testing.

> The proposed taxonomy covers most of the angles that should be taken into consideration for IoT tests. (P6)

Understandability. 68.75% of the participants agreed that the taxonomy is understandable while providing some suggestions.

> The proposed taxonomy is understandable when I read it, especially in its graphical visualization. However, the only thing you should mention is how the testers can read it with brief descriptions of different terms used. (P13)

Usefulness. All participants agree that the taxonomy is useful. No participants disagreed on the usefulness of the taxonomy.

> This taxonomy is useful since it provides the common terminology and taxonomy for people to create and review IoT test plans. Given the common terminology and taxonomy as a framework, the supplier companies, subcontractor companies, and user companies can productively align on the test strategy compared to building it from scratch. (P9)

Helpfulness. 81.25% of the participants agreed that the taxonomy is helpful. However, the participants suggested adding a granular guide for the testers.

> This taxonomy can help testers develop a comprehensive test plan that covers all the different components and elements of an IoT system, including devices, connectivity, data management, analytics, applications, security, and cloud computing. It can also help them design test cases that take into account the various challenges associated with testing IoT systems, such as data volume, device diversity, and complex interactions between components. (P7)

4.1.2. Importance and expectation

We asked the practitioners to assess the importance and the degree to which the taxonomy matched the practitioners' expectations using a scale from 1 (strongly disagree) to 5 (strongly agree).

Importance. 75.0% of the participants agreed on the importance of the taxonomy, while 25.0% of the participants did not agree or disagree that the taxonomy is important to the practitioners.

> Any practitioner will be able to understand different types of testing, which are important in the preparation of a test strategy. (P1)

Expectation of Practitioners. 56.25% of the participants agreed that the taxonomy matches their expectations as testers, while 43.75% did not agree or disagree that this taxonomy matches their level of expectation.

> I've been doing a literature review for the last couple of months, and this taxonomy is one of the best ones I've come across. It is detailed, explained well, and direct. (P8)

4.1.3. Effectiveness for test strategy

We asked the practitioners to assess the effectiveness of the taxonomy for testers when creating the test strategy and test cases (TCs). 93.75% of the practitioners agreed that the taxonomy can effectively support the testing of IoT systems, while 6.25% neither agreed nor disagreed.

> The proposed taxonomy provides an opportunity to have third-party perspective to validate test strategies. (P11)

Some Contributions from Participants. In the testing approach, practitioners suggested incorporating specific security tests such as API security testing, penetration testing, and vulnerability testing. Additionally, they recommended including data privacy testing, user experience testing, patch testing, and model-based testing (MBT). Regarding the objective of testing, they suggested incorporating other factors such as continuous improvement and defect detection.

4.2. Round 2

We used the feedback collected in Round 1 and improved the taxonomy. Further, we invited 204 practitioners to evaluate the revised version of the taxonomy. We used a 1 (strongly disagree) to 5 (strongly agree) scale. Fig. 13(a) shows the evaluation results. We ensured participant anonymity by assigning unique identifiers (P1 to P204) to represent each of the 204 participants. Fig. 12 shows the details of the participants.

Completeness. 95.1% agreed on its completeness. 4.9% did not agree or disagree.

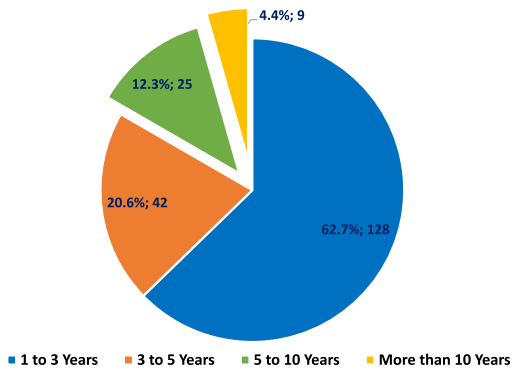
> The proposed taxonomy for an IoT system appears well-organized, covering key aspects such as devices, connectivity, data processing, security, applications, and management. This structure enables a comprehensive understanding of the components and considerations within the IoT ecosystem. (P26)

Coverage. 99.5% of practitioners agreed on the coverage of this taxonomy in terms of testing dimension and level of detail. 0.5% did not agree or disagree.

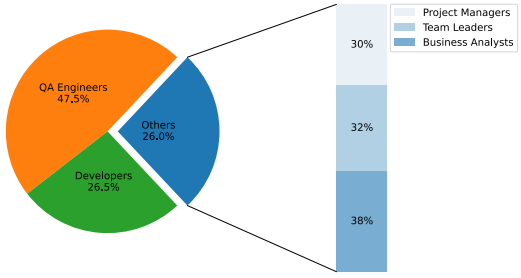
> This taxonomy is well-designed and tackles all aspects of IoT systems. Any practitioner must check this taxonomy before deploying the IoT system. (P176)

Effectiveness. 98.5% of practitioners strongly agreed on how the taxonomy can help the testers to be more effective, while 1.5% did not agree or disagree.

Expectation. 98.0% of practitioners believed that the taxonomy could meet the expectations of the testers, while 2.0% did not agree or disagree.



(a) Participants'Experience in IoT



(b) Participants' Professions

Fig. 12. Participants' details for survey – Round 2.

When it comes to testing an IoT system, the proposed taxonomy can meet my expectations. It can provide a structured framework for understanding and exploring different aspects of testing specific to IoT systems. This can include areas like connectivity testing, interoperability testing, security testing, and more. So, the taxonomy can be a valuable tool for testers to ensure that all necessary dimensions of testing are covered when it comes to IoT systems. (P60)

Understandability. 98.0% of the practitioners believed that the taxonomy is understandable, while 2.0% neither agreed nor disagreed.

The structure and design of the taxonomy demonstrate a commendable effort to provide a comprehensive framework for testing IoT systems. (P31)

Usefulness. 98.6% of the practitioners agreed that the taxonomy is useful, while 1.4% of the participants did not agree or disagree.

This taxonomy is very useful, and complete, and will help the testers to understand all dimensions of testing. (P45)

We further asked the practitioners to evaluate the completeness of each dimension. We used "Yes" or "No" for each dimension. In the case of "No", we asked the practitioners to mention what they believed is missing. We used their feedback to improve the taxonomy. Fig. 13(b) shows the evaluation results. At least more than 96% (196) of the practitioners believed that each of the categories in the taxonomy is complete.

Some Contributions from Participants. On the objective of testing, practitioners suggested adding patch testing, Go-Live testing, backup, and recovery testing to reason-based testing. They also suggested adding cross-functional teams to those responsible for testing (testers). Cloud was suggested by practitioners for the test environment, while the acceptance phase, Go-Live phase, and maintenance phase are suggested for the stage of testing.

We used feedback provided by those who did not strongly agree or were neutral to improve the taxonomy.

5. The taxonomy: TaxIoTe

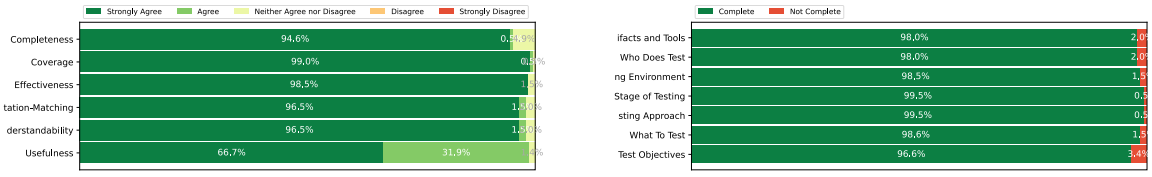
In this section, we present the description of the taxonomy for IoT system testing, referred to as *TaxIoTe*. We describe each aspect in *TaxIoTe* by answering some of our research questions (RQs).

5.1. What are testing objectives (RQ1)?

We compiled and classified the objectives for testing, as depicted in Fig. 14.

We organized the objective of testing based on the reason and goal for testing. [R1C3] On *reason-based*, we identified six reasons: conducting initial testing, retesting, regression testing, patch testing, backup and recovery testing, and Go-Live testing. On *goal-based*, we identified continuous improvement, customer satisfaction, verification and validation, quality assurance, defects detection, battery drainage checking, sensor calibration checking, network coverage and range checking, and geolocation capability checking. It can also focus on end-to-end (E2E) business scenarios. E2E testing can be functional or non-functional requirements-based as shown in Fig. 14. Due to space constraints, we are unable to provide detailed descriptions for each item in the taxonomy here, but we will provide a comprehensive description of each item on our GitHub repository and ptidej website. We suggest the following explanations for certain objectives. *Continuous improvement* in IoT system testing aims to regularly enhance both functionality and the quality of IoT systems. *User satisfaction* in IoT systems testing ensures that the system meets or exceeds user expectations. [R1C2] *Verification and Validation* are separate processes that both use testing as one of their principal practices. *Verification* is the process of evaluating the product or system to ensure that it meets the specified requirements (i.e., building the product right) (ISO, 2022). *Validation* is the process of evaluating the product or system to ensure that it meets the user's needs and expectations (i.e., building the right product). *Quality Assurance* is the process that ensures products or services meet quality standards and expectations (ISO, 2022). *Defects Detection* focuses on identifying errors, bugs, faults, and failures in IoT systems. [R1C3] *Battery drainage checking* ensures that an IoT device's battery life meets expectations and that power consumption is optimized. This involves measuring battery drain under various usage scenarios. *Sensor calibration checking* verifies that an IoT device's sensors are accurately calibrated to provide reliable and precise data, such as temperature, humidity, or pressure readings. *Network coverage and range checking* ensures that an IoT device can maintain a stable and reliable connection to the network within a specified range and coverage area. *Geolocation capacity checking* verifies that an IoT device can accurately determine its location and provide location-based services, such as GPS tracking or location-based alerts. For functional requirements (FR), we defined the following objectives:

- *Requirement Testing.* The objective is to verify and validate specific requirements and specifications of an IoT system. Test cases (TCs) and test scenarios are designed and executed based on the requirements of the IoT system.



(a) Overall Taxonomy Evaluation

(b) Completeness of each Aspect

Fig. 13. Evaluation results - Round 2.

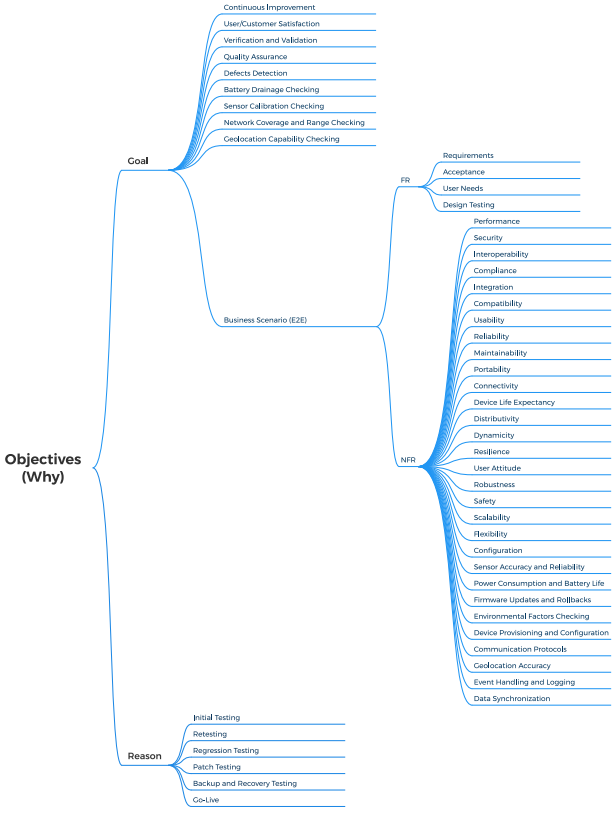


Fig. 14. [R1C3] Testing objectives.

- *Acceptance Testing.* Acceptance testing is owned by the customer to verify that the system works in accordance with the customer's expectations (Rogers, 2004).
- *User Need Testing.* User-need testing verifies if the IoT system meets stakeholders' needs.
- *Design.* Design-driven testing verifies if the IoT system conforms to the provided designs.

For non-functional requirements (NFR), we identified the following goals:

- *Performance Testing.* Performance testing is conducted to evaluate the degree to which a test item accomplishes its designated functions within given constraints of time and other resources.
- *Security Testing.* Security testing is conducted to evaluate the degree to which the SUT, and associated data and information, are protected so that unauthorized persons or systems cannot use, read, or modify them, and authorized persons or systems are not denied access to them (Alaqail and Ahmed, 2018).
- *Interoperability Testing.* It refers to the degree to which the system operates (that is, interfaces and collaborates) effectively with specified external systems (Firesmith, 2014). It helps to test that various devices from different vendors can communicate and understand each other.
- *Compliance Testing.* Compliance testing ensures that the SUT complies with laws, standards, or regulations.
- *Integration Testing.* It focuses on checking the interface between different components of the SUT (Alaqail and Ahmed, 2018).
- *Compatibility Testing.* Compatibility testing measures the degree of satisfaction to which a SUT can function satisfactorily alongside other products (Alaqail and Ahmed, 2018).
- *Usability Testing.* Usability testing checks the user behavior in using the system. Testing the UI and its navigation helps elucidate if a better design can make navigation simple and desirable, as well as prevent users from doing something they should not need to do (Hagar, 2022).
- *Reliability Testing* Reliability testing evaluates the ability of a SUT to perform its required functions, including evaluating the frequency with which failures occur (Alaqail and Ahmed, 2018).
- *Maintainability Testing.* Maintainability testing is conducted to evaluate the effectiveness and efficiency with which a SUT may be modified (Alaqail and Ahmed, 2018).
- *Portability Testing.* Portability testing assesses how easily a SUT can transition from one environment to another and the extent of modifications required to make it functional in diverse environments.
- *Connectivity Driven Testing.* Connectivity testing is conducted to check how IoT device connects to other devices, and endpoints such as gateways (Taivalsaari and Mikkonen, 2018).
- *Device Life Expectancy (DLE) Testing.* DLE testing is conducted to evaluate the time interval from when a device is sold to when it is discarded (Lee et al., 2023).
- *Distributivity Testing.* Distributivity testing is conducted to check how components of SUT can be spread across various devices in the network (Tanenbaum and Steen, 2015).
- *Dynamicity Testing.* Dynamicity testing checks how components and connectors can be created, connected, or removed during system execution (Cavalcante et al., 2015).
- *Resilience.* The ability of an IoT system to recover quickly from failures, errors, or disruptions, and to continue operating effectively.
- *User Attitude.* Testing to ensure that an IoT system is user-friendly, intuitive, and meets user expectations, including usability, accessibility, and overall user experience.
- *Robustness.* The ability of an IoT system to withstand and resist failures, errors, or disruptions, and to continue operating effectively in adverse conditions.
- *Safety.* Testing to ensure that an IoT system does not pose any risks or hazards to users, and that it operates in a safe and secure manner.
- *Scalability.* The ability of an IoT system to handle increased load, traffic, data, or added devices without compromising performance.
- *Flexibility.* The ability of an IoT system to adapt to changing requirements, protocols, or technologies, and to integrate with other systems or devices.
- *Configuration.* Testing to ensure that an IoT system can be properly configured, customized, and updated and that its settings and parameters can be easily managed.
- *Sensor Accuracy and Reliability.* This involves verifying that sensors in the IoT devices accurately and consistently measure and report data. For example, temperature sensors should report correct temperature readings under various conditions.

- *Power Consumption and Battery Life.* Ensuring that the IoT devices have efficient power consumption and that the battery life meets the expected requirements.
- *Firmware Updates and Rollbacks.* Verifying that firmware updates can be applied successfully over-the-air (OTA) and that devices can revert to a previous version if an update fails.
- *Environmental Factors Checking.* Testing the device's performance under various environmental conditions, such as extreme temperatures, humidity, and other factors that the device might encounter in real-world use.
- *Device Provisioning and Configuration.* Ensuring that new devices can be correctly set up and configured to join the network. This includes testing the process of adding a device, configuring it, and ensuring it starts functioning as expected.
- *Communication Protocols.* Testing the implementation and performance of communication protocols specific to IoT, ensuring that protocols like MQTT, CoAP, and others are correctly implemented, efficient, and secure.
- *Geolocation Accuracy.* Verifying the accuracy and reliability of location-based services provided by the device. This includes testing GPS and other location technologies to ensure they report correct locations.
- *Event Handling and Logging.* Ensuring that the device correctly handles events (e.g., sensor triggers, and network changes) and logs relevant information.
- *Data Synchronization.* Verifying that data collected by the device is correctly synchronized with the cloud or other central systems. This includes ensuring data integrity and consistency across all platforms.

NFR focuses on the quality attributes of IoT systems. Our listing is not exhaustive; therefore, any other quality attributes not mentioned here could be added.

We identified various reasons for testing. *Initial testing* verifies the basic functionality and quality aspects of the IoT system before deployment. *Retesting* is conducted to verify that defects identified during initial testing have been fixed successfully. *Regression testing* ensures that changes or updates to the IoT system do not adversely affect existing functionalities. *Patch testing* validates the effectiveness and stability of applied patches or updates to the IoT system, ensuring that they do not introduce new issues or disrupt existing functionalities. *Backup and recovery testing* ensures the reliability and effectiveness of data backup and recovery processes in the event of system failures. *Go-Live testing* verifies the readiness of the IoT system for deployment in a live environment.

Takeaway: Although the objectives for testing IoT systems are adaptable to specific contexts, they provide comprehensive coverage across various layers from devices and networks to data processing and applications. Moreover, these objectives can also be applied to testing web, desktop, mobile, and embedded systems.

5.2. What are testing tools and artifacts (RQ2)?

We organized testing tools and artifacts for testing IoT systems, as shown in Fig. 15.

The identified artifacts include TCs, test data, test scenarios, test scripts, defects reports, test plans, test strategies, and traceability matrix. We define those artifacts based on ISO (ISO, 2022).

- *Test Scenario.* A test scenario is defined as any functionality that can be tested (e.g., check the login functionality). One test scenario can have multiple TCs.
- *TCs.* A test case is any single atomic test consisting of test preconditions, test inputs, expected test output, and expected test postconditions (Firesmith, 2014).

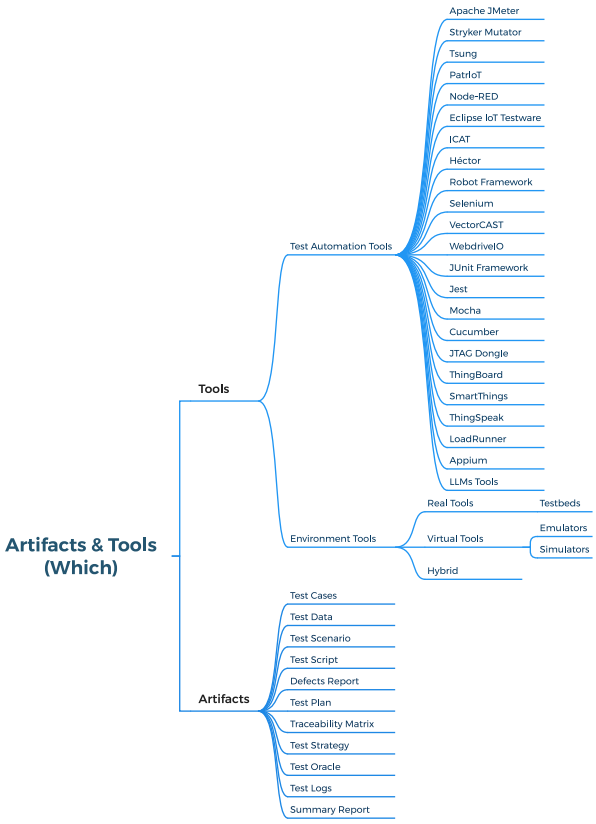


Fig. 15. [R2C4] Which tools to use and artifacts produced.

- *Test Data.* Test data are data used to satisfy the input requirements for executing one or more TCs.
- *Test Script.* A test script is a procedure specification document specifying one or more test procedures.
- *Defect Report.* A defect report is a report indicating whether a specific test case has passed or failed.
- *Test Plan.* The test plan consists of a detailed description of test objectives to be achieved and the means and schedule for achieving them, organized to coordinate testing activities for some test item or set of test items.
- *Test Strategy.* Test strategy is a document that outlines how testing will be conducted.
- *Test Log.* A record of the testing activities that have been performed on a software system, including the test cases that were run, the results of those tests, and any defects or issues that were identified.
- *Summary Report.* It summarizes the test execution, including the number of test cases executed and key metrics such as test coverage, pass/fail rates, and defect trends.
- *Traceability Matrix.* A tool to trace the requirements of a system to the corresponding test cases that have been created to validate those requirements (IEEE, 2017).
- *Test Oracles.* A source for determining expected results for test cases and comparing them to the actual results of the SUT.

[R2C4] Testing tools are software applications or frameworks designed to support and automate various aspects of the testing process, including test case creation, execution, defect tracking, performance monitoring, and result analysis (Firesmith, 2014). These tools are categorized into two groups: one for running tests (e.g., Selenium, Appium, and many LLM-based testing tools, such as ChatGPT, CodeX, CodeT5, GPT-4, and GPT-3 (Wang et al., 2024a)), and the other for managing test environments (testbeds or simulators/emulators (Dias et al., 2018; Fortino et al., 2020)). We identified 23 tools in this taxonomy namely

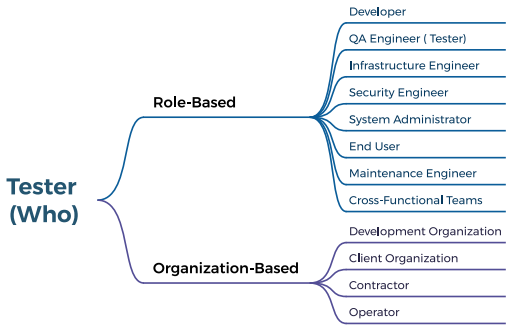


Fig. 16. Who test.

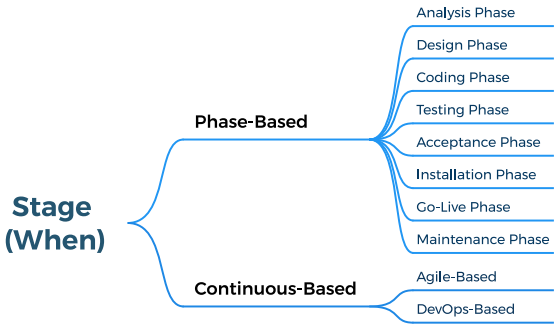


Fig. 17. Stages of testing.

Apache JMeter, Stryker Mutator, Tsung, PatrIoT, Node-RED, Eclipse IoT Testware, ICAT, Héctor, Robot Framework, Selenium, VectorCAST, WebdriveIO, JUnit Framework, Jest, Mocha, Cucumber, JTAG Dongle, ThingBoard, SmartThings, ThingSpeak, LoadRunner, Appium, and LLMs Tools.

- *Testbeds.* Testbeds that consist of hardware and software components where IoT systems can be tested.
- *Virtual Tools* Virtual tools consist of real software on virtualized infrastructure (Becker et al., 2022).
- *Emulators.* Emulators are software tools that mimic the hardware and software of a real device (Firesmith, 2014).
- *Simulators* Simulators are software tools that mimic the software and environment where IoT devices operate, often for testing and development purposes (Firesmith, 2014).
- *Hybrid Environments.* Hybrid Environments comprise a mix of physical hardware and virtualized components.

Test automation tools are excluded from the taxonomy due to their dynamic nature; the roster of known tools is subject to rapid changes. Additionally, numerous other artifacts may exist and could be considered. This taxonomy includes only those artifacts identified in existing literature or reported by practitioners.

Takeaway: The taxonomy excludes test automation tools due to their dynamic and rapidly evolving nature, focusing instead on artifacts common to all software systems. It is also important to recognize that many practitioners are not yet fully aware of LLM-based testing tools, such as ChatGPT, CodeX, CodeT5, GPT-4, and GPT-3(Wang et al., 2024a), which are gaining relevance in the field.

5.3. Who is responsible for testing (RQ3)?

We summarized the roles of individuals or organizations responsible for testing, as shown in Fig. 16.

We categorized the testers into 2 main groups: organization-based and role-based. Within the organization-based category, testing can be the responsibility of the following:

- *Developing Organization.* The organization responsible for development also conducts the testing.
- *Client Organization.* The client is responsible for testing.
- *Contractor.* Testing is outsourced to a third party.
- *Operator.* The operator may conduct some tests such as security and performance assessments.

The role-based classification includes:

- *Developer.* The developer of the system can also write and execute TCs, especially for unit tests.
- *QA Engineer.* QA engineers ensure the quality of the system through testing and process enhancement.
- *Infrastructure Engineer.* Infrastructure engineers evaluate various aspects of the infrastructure, including performance, security, reliability, maintainability, and distributivity, to ensure that the system meets quality requirements.
- *Security Engineer.* Security engineers are responsible for testing the system's security and addressing vulnerabilities.
- *System Administrator.* System administrators can conduct various tests and take care of resource usage, backup, and recovery.
- *End User.* End users can conduct their tests to determine whether the system aligns with their needs and requirements and whether it is acceptable for use.
- *Maintenance Engineer.* Maintenance engineers may test system updates, patches, and bug fixes for stability, compatibility, and functionality.

Takeaway: Testers are categorized according to their organization or job role, which allows specialists such as developers, QA teams, clients, and maintenance engineers to conduct targeted tests at various stages of the system's lifecycle. The formation of a cross-functional team, composed of individuals from diverse roles, may facilitate a unified understanding of testing outcomes among all stakeholders.

5.4. What are testing stages (RQ4)?

We summarized the stages of testing for the phase-based development approach, as shown in Fig. 17.

In phased development, testing occurs at various phases such as analysis, design, and coding. In many cases, testing is done in the testing phase. In continuous development, testing is conducted based on Agile or DevOps methodologies.

- *Agile Testing.* In agile-based testing, features are tested as they are developed.
- *DevOps Testing.* In DevOps, the new build is run through a series of tests that will thoroughly check if the new build is ready for production.

Takeaway: In phased development, testing aligns with stages like analysis and coding, while Agile and DevOps in continuous development, test features as they emerge. For the dynamic nature of IoT systems, Agile or DevOps methodologies are preferable for their ability to handle rapid changes and complex integrations.

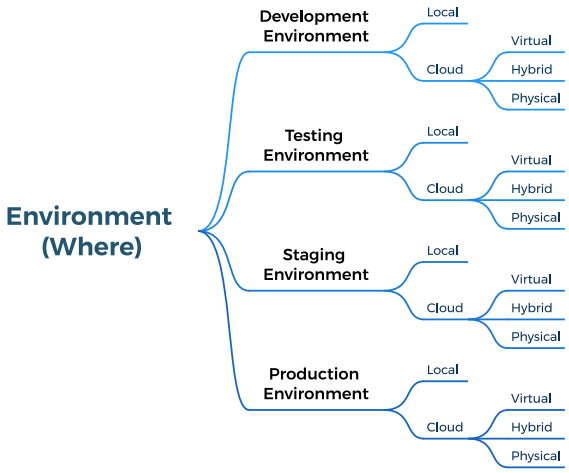


Fig. 18. Testing environments.

5.5. What are testing environments (RQ5)?

We categorized the options for the testing environment, as shown in Fig. 18.

We identified four environments: development, testing, staging, and production environment.

- *Development Environment (DE).* DE is the environment where the system is developed.
- *Testing Environment (TE).* TE is the environment where the testers ensure the quality of the system, open bugs, and review bug fixes.
- *Staging Environment (SE).* SE is a closely replicated version of a production environment designed for system testing purposes. It serves as a platform to evaluate source code, builds, and updates to ensure their quality and functionality under conditions closely resembling those of the production environment.
- *Production Environment (PE).* PE is the environment where the latest versions of a system or updates are pushed live to the intended users.

Takeaway: Four key test environments (i.e., development, testing, staging, and production) serve distinct purposes from system development to live deployment. For IoT systems, the staging environment is preferred as it mirrors the production environment, allowing for thorough testing under real-world conditions before deployment.

5.6. What are testing approaches (RQ6)?

We identified testing levels, testing types, testing techniques, automation, and scripting levels, as shown in Fig. 19. We explain each category in this section based on ISO-29119 standards (ISO Standards, 2021).

- *Testing Levels.* There are six testing levels: unit, integration, system, acceptance, regression, and patch testing (Tan and Cheng, 2018). *Unit testing* checks individual IoT system components separately. *Integration testing* gradually tests larger subsystems as they integrate into the whole system (Firesmith, 2014). *System testing* examines the entire system, not just individual components. *Acceptance testing* (see Section 5.1). *Regression testing* verifies code changes have not introduced new defects (Firesmith, 2014).
- *Testing Types.* We identified several types of testing named after their test objectives. The definitions for these testing types are provided in Section 5.1.
- *Testing Techniques. Technique-based testing* focuses on techniques used to assess the system functionality and quality to determine if it meets the specified requirements.
  - *Black-Box Testing.* Any testing with no knowledge of the internal structure or code of the component or system (Firesmith, 2014). This testing is restricted to the externally visible behavior and characteristics of the item being tested. *Combinatorial testing* is a testing method that uses multiple combinations of input parameters to perform testing (ISO, 2022). *Decision-table testing* is a specification-based test design technique based on exercising decision rules in a decision table (ISO, 2022). *Scenario testing* is a testing technique that uses scenarios, i.e. speculative stories, to help the tester assess a test system. *User interface (UI) navigation testing* is a testing technique that focuses on verifying the navigation flow and UI elements of a SUT. *State-based testing* evaluates the behavior of the system based on different states. *Boundary-value testing* is a testing technique in which TCs are selected just inside, on, and just outside each boundary of an equivalence class of potential TCs (Firesmith, 2014). *Equivalence class testing* divides the set of test inputs into different equivalence data classes.
  - *White-Box Testing.* White-box testing is based on an analysis of the internal structure and the code of a component or system under test (Firesmith, 2014). *Control flow testing* such as *branch coverage, path coverage, condition coverage, loop coverage, and modified condition/decision coverage (MC/DC)*, focuses on the execution order of statements to develop the TCs. Branch coverage ensures that each branch is executed, thus ensuring that all reachable code is executed. Path coverage ensures that all possible paths are tested. Condition coverage focuses on the execution of different conditions independently of each other. Loop coverage focuses on the validity of the loop constructs. MC/DC ensures that each condition independently affects the decision outcome (i.e., ensuring that every condition within a decision determines all possible outcomes of that decision). *Data flow testing* focuses on variables and their values across the different components of a system.
  - *Patterns-based Testing.* Testing patterns are recurring, proven approaches or strategies used to design and execute effective TCs. Those patterns include *periodic readings testing* that evaluates system behavior during regular intervals. *Triggered Readings Testing* focuses on system behavior triggered by specific events. *Alerts testing* that verifies the generation and handling of system alerts. *Actions testing* that tests the actions or responses initiated by the system. *Actuators testing* that ensures proper functioning of actuators.
  - *Experience-based Testing.* Experience-based testing is a testing technique solely based on the experience of the tester. Those include *bug hunt testing*, a structured search for defects in the absence of formal TCs. *Error guessing testing*, using testers' experience to guess the potential error-prone areas of the system. *Exploratory testing*, experience-based testing in which the testers simultaneously design and execute tests based on the tester's existing relevant knowledge, prior exploration of the test item (ISO, 2022).
  - *Random Testing.* Testing techniques that are based on generating random inputs and TCs. Those include *fuzz testing* and *monkey testing*. *Fuzz testing* is an approach in which random inputs, called fuzz, are used to cause the system to fail (ISO, 2022; Firesmith, 2014). *Monkey testing* is a method that applies random inputs to a system, aiming to crash it, without predefined TCs.
  - [R1C2] *Verification and Validation (V&V). Verification* is the process of evaluating the product or system to ensure that it meets the specified requirements (i.e., building the product right) (Firesmith, 2014). Verification can be performed in many ways (e.g., demonstration, inspection, review, and testing). *Validation* is the process of evaluating the product or system to ensure that it meets the user's needs and expectations (i.e., building the right product) (Firesmith, 2014). Validation can be performed using many techniques (e.g., review and workshop). *Static testing* is the evaluation of a

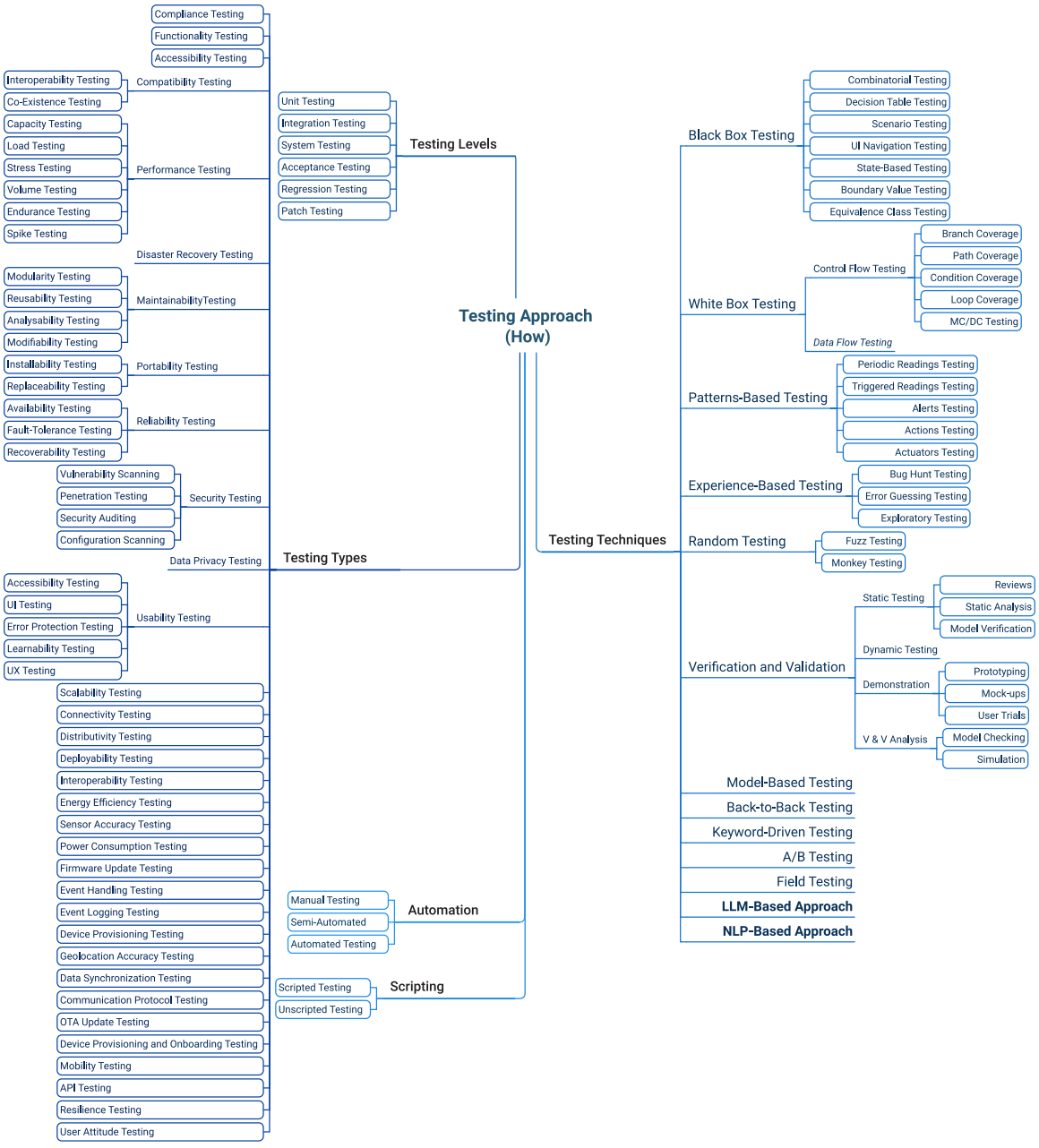


Fig. 19. [R2C4] Testing approaches.

test item where no execution of the code takes place. Static testing can be performed by manual examination of documents or code (e.g., *review*, by automated code analysis tools (e.g., *static analysis*), and by verification of system models or specifications (e.g., *model verification*). *Dynamic testing* involves executing code and running TCs. *Demonstration methods* focus on showcasing system functionalities. It includes *prototyping* (i.e., building a simplified version of the system), *mock-up* (i.e., a non-functional representation of the user interface), and *user trials* (i.e., involving end-users in testing). *V&V Analysis* is an analysis technique for formal verification of the system models (i.e., *model checking*) or creating simulations to assess the system behavior (i.e., *simulation*).

- *Model Based Testing (MBT).* MBT uses models to generate TCs systematically and automatically (ISO, 2022).
- *Back-to-Back (B2B) Testing.* In B2B testing, an alternate system version is used to produce expected results for comparison with the same test inputs (ISO, 2022).
- *Keyword-driven Testing.* TCs are written using keywords to represent test actions and data.
- *A/B Testing.* A/B testing is a statistical method to compare two systems and find which performs better.
- *LLM-Based Techniques.* Large Language Models (LLMs) represent a subset of machine learning models that utilize deep learning techniques and substantial datasets to generate human-like text. These models excel in various tasks, including text generation, question answering, and adapting to diverse scenarios. Recent research highlights an increasing trend in the integration of LLMs into the testing of various systems, demonstrating their versatility and effectiveness in automating and enhancing testing processes (Wang et al., 2024a).
- *NLP-Based Techniques.* Natural Language Processing (NLP) is a domain within artificial intelligence that focuses on analyzing and interpreting human language. It uses its own rules and statistical methods for structured tasks such as information extraction

and translation. By leveraging explicit linguistic rules and knowledge, NLP systems are designed to parse and understand grammar, syntax, and semantics. These systems utilize predefined rules to analyze sentence structure, identify key terms, and interpret the finer details of language. This enables NLP to discern the meaning and intent behind words expressed by humans. The application of NLP techniques in automating testing processes is notable, especially in converting requirements specified in natural language into executable tests (Wang et al., 2020; Fischbach et al., 2023).

  - *Functionality Testing.* Any testing intended to verify functionality by causing the implementation of a system function to fail to identify defects (Firesmith, 2014).

- *Automation Levels.* TCs can either be run manually by a human (*i.e., manual testing*), or they can be executed by a test automation tool (*i.e., automated testing*). *Semi-automated testing* mixes automated testing and manual testing (ISO, 2022).
- *Scripting Levels.* Scripted testing follows a documented test script (ISO, 2022), adhering to pre-defined TCs and steps. In contrast, unscripted testing involves no formal preparation, documentation, or test scripts.
- *Mobility Testing.* Evaluate device performance and connectivity while in motion, simulating real-world use cases where devices are mobile (e.g., in vehicles, on drones).
- *Field Testing.* Deploy devices in actual use-case scenarios outside the lab to gather performance data in real-world conditions, assessing connectivity, power consumption, and sensor accuracy.
- *Over-the-Air (OTA) Update Testing.* This type of testing focuses on evaluating the process of remotely updating the device firmware, ensuring successful updates, and handling failures.
- *Real-world Environment Simulation.* This is a strategy used to create test environments that mimic real-world conditions to evaluate how IoT devices perform under various environmental factors.

The taxonomy lacks some details, especially for dynamic testing and model-based testing. Although some details could be provided in other published materials such as ISTQB (Stapp et al., 2024) or in ISO 29119 standards (ISO Standards, 2021), we included only aspects that we identified from SLR or that were suggested by practitioners. Practitioners may consult those materials for more details if required.

Takeaway: This taxonomy covers a wide range of testing approaches suitable for many types of systems, including specific approaches unique to IoT systems. For IoT systems, leveraging emerging approaches like LLMs could enhance testing efficiency and effectiveness by automating tasks.

5.7. What are OUT and metrics (RQ7)?

We identified the target item (specific IoT system layer or end-to-end) and the metrics, as shown in Fig. 20.

Items under test include end-to-end (E2E) scenarios or specific layers. E2E testing focuses on testing the entire system with all components integrated. E2E testing can be performed on the real system. Testers can also use models to represent the desired behavior of a system under test (SUT). Items under test can be specific layers such as device, network, cloud, or application layer. Metrics fall into two categories: domain-specific and general metrics, with the latter categorized into three groups: code-level, functional-based, and non-functional-based metrics as indicated in Fig. 20. For functionally-based metrics, we classify them into two primary categories: Defect/Bug-based and Specification-based metrics. Under Defect/Bug-based metrics, we identified three metrics: the number of bugs or defects detected, the number of failures or crashes, and the number of code smells. [R2C4] For Specification-based metrics, we identified the following metrics known from traditional software systems: the number of requirements covered, the number of features covered, and the number of test cases that

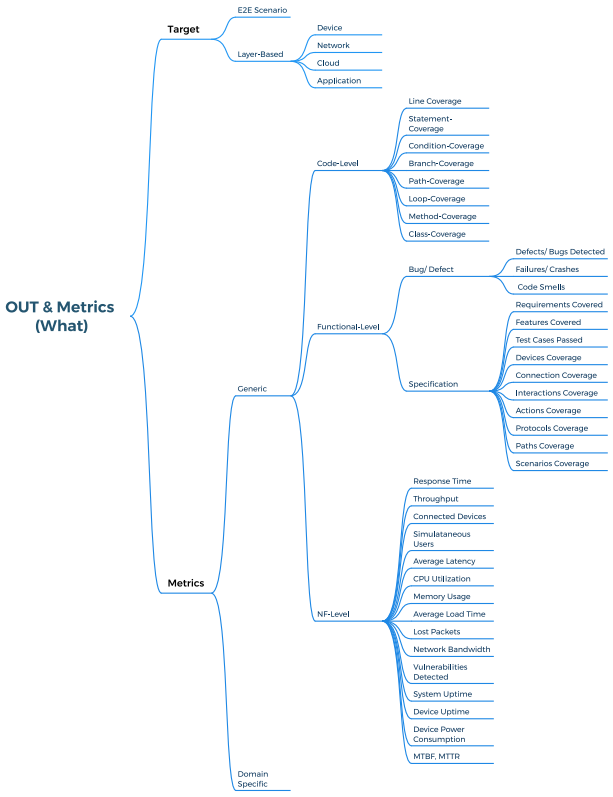


Fig. 20. [R2C4] What to test.

passed or failed. However we have added IoT specific metrics such as: action/operation/command coverage, connection coverage, device coverage, interaction coverage, protocol coverage, and path coverage.

- An action or operation or command refers to a specific task or function that the system performs. Action coverage is the degree to which tests exercise the actions or behaviors triggered by commands.
- Connection in IoT refers to the methods and technologies used to establish communication between IoT devices and networks, allowing data exchange between devices, gateways, and cloud platforms. Connection coverage refers to the degree to which communication paths between different components in an IoT system are covered by tests.
- IoT devices are physical objects embedded with sensors, software, and other technologies that enable them to connect and exchange data over the Internet. Device coverage refers to the extent to which distinct devices within IoT system are covered by tests.
- Interaction refers to the bidirectional or unidirectional exchange of data, commands, or signals between devices, application layers, or other system components. Interaction coverage is the degree to which the data exchange and communication flows between interconnected IoT components (devices, cloud services, mobile apps, robots, etc.) are covered by tests.
- A path is a specific sequence of actions to achieve a particular goal or objective within a use case. Path coverage measures how well tests cover distinct execution paths, including all possible combinations of decisions and conditions, even if those paths are not explicitly defined in the use case specification.
- A protocol in IoT systems is a set of rules that governs how devices and components communicate and exchange data. Examples include MQTT for lightweight communication, HTTP for web-based interactions, WebSocket for real-time communication, and Bluetooth for short-range connectivity. Protocol Coverage in IoT systems refers to the extent to which all communication protocols used within the IoT

system (such as WebSocket, HTTP, COAP, MQTT, etc.) have been included in tests.

Takeaway: The taxonomy highlights end-to-end (E2E) scenarios for testing the integrated functionality of entire systems, which may be beneficial for IoT systems due to their interactions across multiple layers, such as device, network, cloud, and application. Metrics are grouped into domain-specific and general categories, the latter including code-level, functional, and non-functional metrics. Functional-level metrics could be particularly relevant in the context of IoT systems.

6. Empirical evaluation

We conducted the experiments to assess *how does taxonomy improve testing (RQ8)*. We used two systems as our case studies: "Where Is My Professor" (WIMP) and "Smart Crop Yield Prediction System" (SMART-CYPS).

6.1. Selection of participants

[R2C3] The participants involved in the empirical evaluation were not part of the previous surveys conducted. For each system, 12 master's and Ph.D. students were selected as participants. These students were carefully chosen based on their affiliation with two specific labs that had access to the IoT systems used in the evaluation. The first group of students was from the Ptidej Lab at Concordia University, Canada, where the WIMP system is developed and maintained. This affiliation made them ideal candidates for the experiment. While the lab includes many students, only those with prior experience in developing or testing IoT systems were selected. Their experience was verified through an analysis of their LinkedIn profiles. The second group of students was from the University of Rwanda's Center of Excellence for IoT, where one of the authors had access to another IoT system, SMART-CYPS. Admission to this program requires students to have at least two years of experience in IoT systems. In summary, these students were chosen because (1) they had relevant experience in developing or testing IoT systems, and (2) they had direct access to the respective systems. The primary focus of the study was to evaluate the breadth of test coverage in terms of the number of tests created with and without the taxonomy, rather than the quality of individual tests. The tests created by the students were not executable. The decision not to execute these tests on the systems was due to the need to convert them into executable scripts, which was beyond the scope of the current study. Our evaluation is solely based on the number of tests and scenarios covered by each participant.

6.2. Experimentation with WIMP

WIMP Description. WIMP,[13] or "Where Is My Professor" is an IoT-based system that enables students to track the availability of their professors in real-time. The main objective of this system is to collect data from various IoT devices, such as sensors, cameras, and beacons, to provide an automated response on the professor's availability by analyzing the collected data. This system has smart sensors, such as the WEMO smart plug,[14] as well as devices like the Fitbit Watch,[15] which serve the purpose of location tracking and play a pivotal role in the decision-making process. The core modules of WIMP are the following:

1. Internal user management module that includes user accounts, profiles, permissions, etc.
2. Data collection module for real-time tracking of professor's availability and location by continuously collecting data from various IoT devices.
3. Availability module to provide an automated response about professor's availability based on analysis of the data collected from various IoT devices.

Fig. 21(a) shows the high-level architecture of WIMP. More details about WIMP can be found online[16]

Participants. We conducted an empirical study with 12 participants, randomly dividing them into two groups: Group 1 (G1) and Group 2 (G2). G2 received a copy of the taxonomy, while G1 did not. We collected the details of participants as recommended in Dutta et al. (2023). G1 comprises three male and three female participants, all holding at least a bachelor's degree and possessing a minimum of three years of experience in software engineering. Additionally, two participants in G1 have prior working experience in IoT systems. G2 comprises two female and four male participants, all holding at least a bachelor's degree and having more than three years of working experience in software engineering. None of the participants in G2 have prior working experience in IoT systems. We asked each participant to study and prepare test scenarios (TSs) and Test Cases (TCs) for the three modules mentioned above, with each participant given 2 h maximum.

Results. Table 7 shows the number of TSs and TCs identified by each participant. We observed that the group that used the taxonomy identified more TSs and TCs compared to the group that did not. G1 primarily focused on functional aspects, while G2 identified more non-functional aspects such as connectivity, security, performance, and compatibility on top of functional aspects. [R2C4] Table 6 shows an example of a test created by one participant for functional testing based on modified template provided by Wang et al. (2020)

[R2C3] Our empirical results show that the taxonomy can guide testers with no prior experience in testing IoT systems by helping them understand the various dimensions of testing in IoT systems, thus creating more tests than the testers without taxonomy.

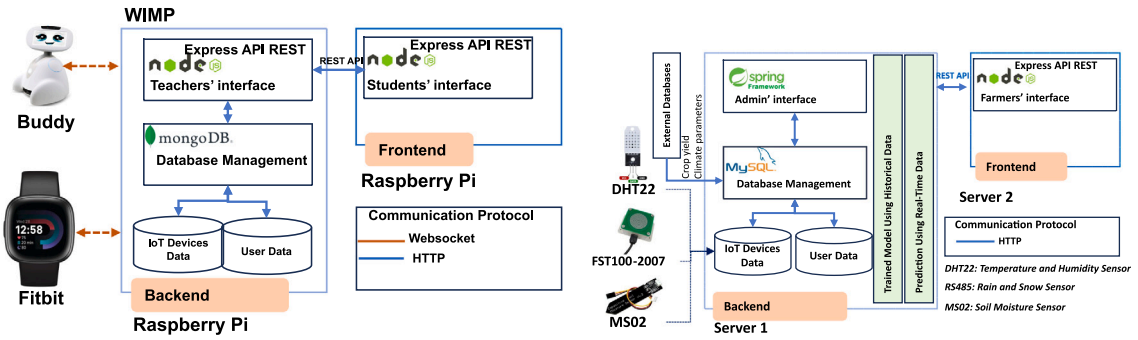
Table 6
Example of test created by participant.

	Operation	Target	Inputs	Expectations
1	ReadData	Fitbit		heart_rate: 95 bpm
2	SendData	Fitbit	heart_rate: 95 bpm	OK
3	ReceiveData	cloudApp	heart_rate: 95 bpm	DATA_RECEIVED
4	AnalyzeData	cloudApp	heart_rate: 95 bpm	
5	sendCommand	cloudApp	action: move; distance: 0.2; speed: 0.3	OK
6	ReceiveCommand	Robot	action: move; distance: 0.2; speed: 0.3	DATA_RECEIVED
7	ExecuteMove	Robot	distance: 0.2; speed: 0.3	MOVE_COMPLETED
8	SendFeedback	Robot	status:MOVE_COMPLETED	OK
9	ReceiveFeedback	cloudApp		MOVE_COMPLETED

6.3. Experimentation with SMART-CYPS

SMART-CYPS Description. SMART-CYPS is an intelligent system powered by the Internet of Things and Machine Learning, specifically designed for crop yield prediction to enhance food security. The primary objective of SMART-CYPS is to tackle the challenges arising from climate change in agriculture and food security through the integration of IoT and machine learning technologies. This system collects real-time weather and soil humidity data from farms, using an architecture that includes IoT devices, a backend layer, and data processing for prediction and weather forecasting. A central component of SMART-CYPS is its dashboard, which dynamically visualizes critical agricultural parameters. An embedded machine learning model within the system

[13] https://ptidejteam.github.io/wimp-wiki
[14] https://www.belkin.com/products/wemo-smart-home/
[15] https://www.fitbit.com/global/en-ca/home
[16] https://ptidejteam.github.io/wimp-wiki/docs/architecture



(a) WIMP- High-Level Architecture (b) SMART-CYPS High-Level Architecture

Fig. 21. IoT systems used in the case study.

Table 7
Empirical evaluation results for WIMP.

G1				G2			
Tester	TSs	TCs	AC	Tester	TSs	TCs	AC
T1	6	33	3	T7	18	48	9
T2	5	10	2	T8	11	32	8
T3	8	34	1	T9	19	61	10
T4	6	11	1	T10	22	40	8
T5	7	14	4	T11	10	41	4
T6	9	34	1	T12	11	43	7
Sum	**41**	**136**	**12**	Sum	**91**	**265**	**46**
IoTS	-	**10**	**1**	IoTS	-	**43**	**4**
Mean	**7**	**23**	**2**	Mean	**15**	**44**	**8**
STD	**1.34**	**11.07**	**1.15**	STD	**4.67**	**8.90**	**1.89**

✺ **TS**: Test Scenarios; **TCs**: Test Cases; **AC**: Aspects Covered; **STD**: Standard Deviation; **IoTS**: IoT Specificity.

Table 8
Empirical evaluation results for SMART-CYPS.

G1				G2			
Tester	TSs	TCs	AC	Tester	TSs	TCs	AC
T1	16	28	2	T7	28	43	13
T2	18	42	6	T8	22	68	20
T3	4	15	2	T9	19	47	11
T4	8	11	2	T10	22	51	12
T5	10	48	9	T11	25	53	9
T6	13	36	4	T12	29	48	14
Sum	**69**	**180**	**25**	Sum	**145**	**310**	**79**
IoTS	-	**6**	**3**	IoTS	-	**18**	**5**
Mean	**12**	**30**	**4**	Mean	**24**	**52**	**13**
STD	**4.75**	**13.50**	**2.61**	STD	**3.53**	**7.95**	**3.44**

✺ **TS**: Test Scenarios; **TCs**: Test Cases; **AC**: Aspects Covered; **STD**: Standard Deviation; **IoTS**: IoT Specificity.

provides harvest predictions based on current sensor data. Fig. 21(b) shows the high-level architecture of SMART-CYPS.

Participants. We invited 12 participants and tasked them to experiment on SMART-CYPS. 8 (66.7%) were master students in the IoT program, and 4 (33.3%) were final-year students majoring in software engineering with a focus on IoT. 7 (58.3%) were female, while 5 (41.7%) were male. 6 (50%) had between 3 and 5 years of experience in SE. 2 (16.7%) had more than 5 years of experience in SE, while 4 (33.3%) had less than 3 years of experience in SE. 8 (66.7%) had between 1 and 3 years of experience in IoT-related projects. 3 (25%) had less than a year, while 1 (8.3%) had between 3 and 5 years of experience in IoT projects. We divided 12 participants randomly into 2 groups of 6 participants each. One group with taxonomy, and the other without. The group with taxonomy was given 30 min to consult the taxonomy before the experiment. We asked each member of each group to conduct the experiment within a maximum of 2 h. Each participant documented test cases and test scenarios in an Excel file, which was submitted after 2 h.

Results. We analyzed the results as shown in Table 8. G1 indicates the group without taxonomy, while G2 represents the group with taxonomy. The participants with taxonomy created more test cases and scenarios than the participants with no taxonomy. They also identified more aspects compared to the group that did not have the taxonomy.

Takeaway: The findings from case studies show that testers knowledgeable about the taxonomy tend to develop a wider variety of test cases and scenarios, covering more aspects of the system, compared to those unaware of the taxonomy.

7. Recommendations

In this section, we provided key recommendations aimed at enhancing test generation and overall testing processes for IoT systems. Future research should explore and validate these recommendations to solidify their effectiveness and adaptability.

† [R2C6]**Comprehensive Test Planning Frameworks for IoT Systems**. From our experiment, we observed that many testers primarily focus on functionality testing, often overlooking other critical aspects. We strongly recommend the development and adoption of a structured test planning framework to address this limitation. Such a framework should explicitly prioritize and integrate diverse testing aspects beyond functionality, including scalability, interoperability, energy efficiency, protocol coverage, and real-time performance. By doing so, testers can ensure a more comprehensive evaluation of IoT systems.

† **Develop Execution-Target-Aware Test Automation Tools**. Testers often focused on defining test steps, inputs, and expected outputs but were unable to create executable tests due to the varying execution targets (i.e., test runners), which use different technologies, programming languages, and operating systems (OSs). We recommend developing automation tools that map high-level test steps to execution scripts tailored to the target's OS and programming language. These tools would enable the execution of generated tests across diverse IoT environments without requiring testers to have detailed knowledge of the technologies used in the execution target.

† **Leveraging LLMs and AI for Test Automation**. [R3C5] From both the surveys conducted and the reviewed papers, we observed that LLMs and AI are often overlooked in the context of IoT systems compared to traditional systems. We recommend the use of Large Language Models (LLMs) to automate essential testing tasks such

as generating test inputs, test oracles, constraints, test cases, and test scripts. These models have been effective in traditional software systems (Wang et al., 2024a) and could be similarly beneficial for IoT systems, which often rely on requirements-based test case derivation. LLMs can reduce manual effort significantly by using extensive datasets to produce context-aware testing components. Future research should focus on tailoring LLMs to address the unique complexities of IoT systems, assessing their accuracy and efficiency in real-world scenarios.

† **Applying NLP-Based Techniques for Use Case-Driven Test Case Generation**. [R3C5] Based on the papers reviewed and feedback from the surveys, the adoption of NLP techniques in testing IoT systems is very low. Given the limited accessibility of source code in IoT systems, we recommend using NLP techniques to derive test cases directly from use case specifications. This approach has been applied to embedded systems (Fischbach et al., 2023; Wang et al., 2020) and could similarly be considered for IoT systems, where access to the source code may not be available for black-box testing. Future research should focus on refining these NLP techniques to improve their ability to accurately convert structured use case narratives into executable test scripts for complex systems like IoT systems.

† **End-to-End (E2E) Testing**. [R3C5] E2E testing is critical for ensuring the seamless functionality of IoT systems, encompassing all components from sensors to applications. By evaluating interactions and data flows across the entire system, E2E testing helps to uncover issues that might not be apparent in isolated testing. Future research should focus on developing or enhancing existing approaches for various aspects of E2E testing in a real environment, enhancing the ability to identify and resolve issues across the IoT system. This recommendation is based on insights gathered during our study, particularly on testing objectives. Many participants emphasized the importance of focusing on E2E scenario testing, for both functional and non-functional related aspects. Functional requirement-based E2E testing includes commonly known approaches like acceptance testing, allowing users to verify that the developed system meets their needs and expectations, ensuring readiness for deployment. E2E testing is also applicable to non-functional aspects such as performance, security, usability, and scalability. Testing individual layers alone may not require a new approach or tool. From our findings, we identified some tools and approaches for embedded systems. These can be used for testing various devices at the device layer. Likewise, we identified approaches and tools for testing web-based, mobile, or desktop applications, and these can be used for testing the application layer. We observed limited approaches and tools for testing the entire system, referred to as E2E. Existing E2E approaches focus on specific aspects, and they are limited in scope. Bosmans et al. (2019) proposed a hybrid simulation-based testing approach focusing on interactions of IoT system entities. However, testing in a real environment is desirable because IoT systems may misbehave in a real environment while behaving well in a simulated one. Clerissi et al. (2018) proposed an approach based on the behavior of the SUT, such as a state machine diagram. However, for complex IoT systems, producing such a behavioral model may not be feasible or detailed enough for automation of IoT system testing. Kim et al. (2018) proposed an IoT testing framework focusing on conformance and interoperability. However, there are several other aspects that are not yet explored from an E2E perspective. While emphasizing that E2E testing is crucial for ensuring the comprehensive functionality of IoT systems, it is equally important not to overlook testing individual layers or the interactions between layers. For practitioners aiming to implement E2E testing effectively, we propose a three-step approach, as illustrated in Fig. 22.

✍ **Step 1: Testing Each Layer Separately.** This step focuses on evaluating each IoT system layer individually. Testing recommendations include both functional and non-functional tests, tailored to the specific layer:

- ∗ *Device Layer:* Functional testing verifies device functions such as sensing, processing, and actuating (e.g., sensor accuracy, actuator response, and communication protocols). Non-functional testing checks performance, security, reliability, and battery life.
- ∗ *Gateway Layer:* Functional testing focuses on data processing tasks like filtering, aggregation, and transmission. Non-functional testing addresses reliability, scalability, and network performance.
- ∗ *Cloud Layer:* Functional testing includes data processing and storage validation, while non-functional testing assesses aspects like availability, failover, scalability, and security.
- ∗ *Application Layer:* Functional testing ensures user interfaces and functionalities meet requirements, while non-functional testing targets user experience (UX), security, and performance.

✍ **Step 2: Testing Interfaces Between Layers.** This step ensures seamless integration between IoT layers:

- ∗ *Device-Gateway Interface:* Tests reliability of data transmission between IoT devices and gateways.
- ∗ *Gateway-Cloud Interface:* Validates data aggregation, filtering, and transmission from the gateway to the cloud.
- ∗ *Cloud-Application Interface:* Verifies the functionality of APIs for retrieving IoT data and sending commands.

✍ **Step 3: Testing the Entire System.** The final step involves system integration testing, ensuring the end-to-end flow of the entire IoT system. This is analogous to user acceptance testing in traditional software, evaluating whether the system meets user specifications. For example, in a smart home scenario, pressing a button on the mobile app should trigger a seamless actuation process, such as unlocking a door via a connected actuator.

† **Generalizing IoT Testing Taxonomy for other Systems**. [R3C5] Using the taxonomy, the participants created numerous tests that are also applicable to traditional systems. This IoT system testing taxonomy can be adapted to other systems, such as web applications, desktop applications, mobile applications, embedded systems, and more, by omitting IoT-specific components, providing a flexible framework for diverse systems.

8. Discussions

In this section, we discuss some observations and practical implications of this taxonomy for both practitioners and researchers.

8.1. Alignment of testing objectives and testing types

[R1C3] The link between testing objectives and testing types is a critical aspect of understanding their alignment. Fig. 14 defines the goals that testing seeks to achieve, while testing types, shown in Fig. 19, represent the diverse methods used to accomplish these goals. The overlap between these elements is both expected and intentional. Fig. 14 highlights the most commonly reported objectives identified through literature and surveys. Similarly, the testing types encompass a broader range, including not only tests explicitly targeting these objectives but also other forms of testing reported in the literature and surveys, regardless of whether they directly align with specific objectives.

8.2. Practitioners feedback

We created an initial taxonomy based on PSs and refined it through collaboration with industry experts to address the practical needs of IoT systems testing. This collaborative effort ensures that the taxonomy not only encompasses theoretical concepts from academia but also incorporates real-world insights and challenges faced by industry practitioners.

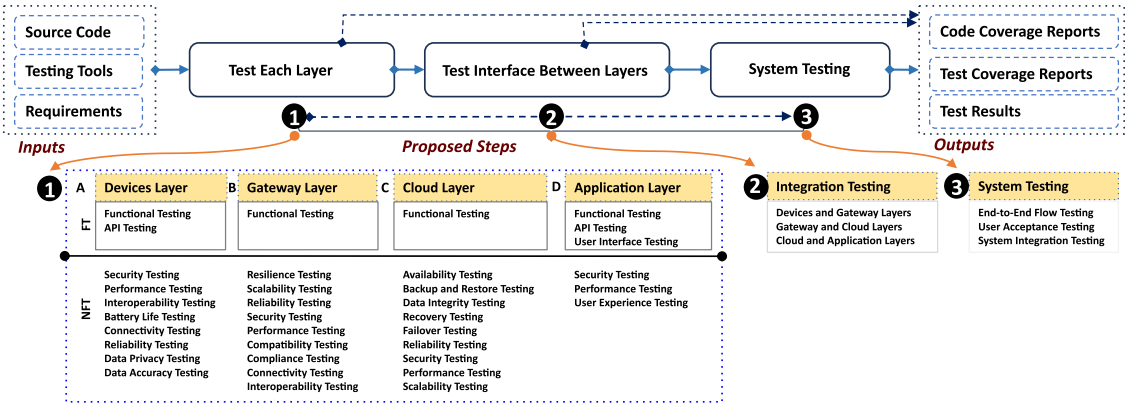


Fig. 22. An Approach for end-to-end testing.

8.3. Experimental insights

[R2C6] The experiment provided valuable insights into testing IoT systems on two different platforms. Testers predominantly focused on functionality, device connectivity, and data transmission, possibly because they undervalued other aspects or were accustomed to prioritizing functionality, device connectivity, and data transmission in their testing practices. Additionally, testers struggled to create executable tests due to varying execution targets that required knowledge of different technologies and programming languages. Instead, they concentrated on defining test steps, specifying inputs, expected outputs, and execution targets. This approach is reasonable given their limited knowledge of execution targets (i.e., test runners). Interestingly, some testers without access to the taxonomy performed well, likely due to their prior experience and the time dedicated to the experiment. However, the taxonomy proved particularly beneficial for testers with no prior experience, offering them a structured framework and a broader range of testing options. This suggests that using the taxonomy can significantly enhance testing outcomes, especially for novice testers.

8.4. Navigating the taxonomy for effective testing

[R3C1] The proposed taxonomy provides a structured, seven-step navigation map (Fig. 8) to guide testers in planning and executing comprehensive testing. The process begins with defining testing objectives (reason for testing) and identifying the object under test, which may range from individual layers (e.g., devices, networks) to end-to-end system evaluations. Testers then select appropriate testing approaches, including types (e.g., security, performance), levels (e.g., unit, integration), or techniques (e.g., random, model-based, black-box/white-box). Subsequent steps involve setting up the testing environment, defining testing stages (e.g., phase-based testing or continuous testing), specifying testing artifacts (e.g., bug reports, test reports), and choosing tools for test instrumentation. The final step assigns testers with the necessary expertise to ensure success. Fig. 9 exemplifies how testers can tailor their testing plan, highlighting mandatory aspects such as objectives, artifacts, testers, stages, approaches, objects under test, and environments. Optional considerations like scripting levels, automation, and tools depend on the testing automation-level. For example, if testing is to be done manually, these may be unnecessary. We believe that this taxonomy can serve as a valuable resource for creating a complete IoT and traditional systems testing framework. To ensure accessibility, we will make this guide publicly available, helping practitioners not only understand the taxonomy but also effectively apply it.

8.5. Continuous relevance and adaptability

[R2C8] To ensure the taxonomy remains comprehensive and up-to-date, we began with a systematic review of studies published up to 2022, complemented by insights gathered through industry collaboration via surveys to validate and enrich the taxonomy. Recognizing the importance of incorporating the latest advancements, we extended our review at the time of submission by conducting an updated search for studies published between 2022 and November 2024, applying the same inclusion and exclusion criteria. This process identified a few new studies (i.e., Tewari et al. (2024), Pietrantuono et al. (2023), Jean Baptiste et al. (2024a)) not included in our initial review, from which additional testing aspects such as user attitude testing or resilience testing were extracted to refine the taxonomy where necessary. Furthermore, we will continue to review the literature and gather feedback from industry practitioners to regularly update the taxonomy and incorporate emerging aspects of IoT testing.

8.6. Fragmented IoT system testing aspects

Existing studies, including Tan and Cheng (2018), Medhat et al. (2019) and Kiran Bhagnani (2014), Firesmith (2015), provided a taxonomy focusing solely on testing types. Notably, study (Makhshari and Mesbah, 2021a) focused on bug taxonomy within IoT systems. [R1C4] To the best of our knowledge, no IoT system testing taxonomy existed before our work. Our proposed taxonomy is the first of its kind and is also adaptable for traditional software testing. This contribution not only fills a significant gap in the literature but also lays a foundation for future research in IoT system testing. Furthermore, our taxonomy's adaptability for traditional software testing extends its utility beyond the IoT domain, making it a valuable resource for researchers and practitioners across different domains of software engineering.

8.7. Implication for practitioners

We provided the guide for testing IoT systems and methodology for End-to-End testing to practitioners.

8.7.1. Guidance for testing IoT systems

We observed that many studies often use some terms interchangeably to denote distinct concepts, potentially leading to confusion among testers. We proposed some definitions to clarify those concepts for better understanding. Yet, the taxonomy may not be exhaustive enough to guide the testers; it offers valuable guidance for novice practitioners embarking on their IoT system testing endeavors. By providing clear definitions and categorizations, this taxonomy serves as a foundational tool for establishing a common language within the testing community, fostering clearer communication and reducing ambiguity in IoT testing practices. Additionally, it lays the groundwork for

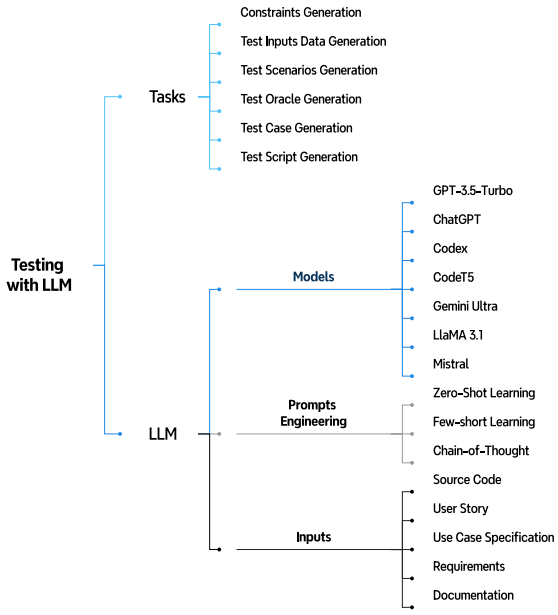


Fig. 23. Test automation tasks with LLMs.
Source: Adapted from Wang et al. (2024b).

further research and refinement, enabling ongoing improvements in testing methodologies and techniques tailored to the unique challenges of IoT environments. We made this taxonomy available online[17] for practitioners and researchers who may want to refer to it.

8.7.2. Revolutionizing testing with LLM

Recent advancements in large language models (LLMs) have opened new avenues for automating various tasks in testing traditional systems. While it is unclear whether any studies have specifically focused on automating testing aspects within IoT systems, we believe that IoT system practitioners can equally benefit from using LLMs to automate many testing tasks. Fig. 23 summarizes some testing artifacts that can be automated using LLMs, as adopted from Wang et al. (2024b), Boukhlif et al. (2024), Yu et al. (2023), Hassan et al. (2024), Schäfer et al. (2024), Wang et al. (2024a), where LLMs have been studied for their potential to enhance test automation.

8.8. Implication for researchers

A promising avenue for future research involves investigating the integration of large language models (LLMs) and artificial intelligence (AI) to enhance the automation of testing processes within IoT systems, potentially advancing IoT system test automation to the next level.

9. Threats to validity

[R2C5] There are threats to the validity of this study, the taxonomy, and its evaluation.

Construct Validity. The primary threat to construct validity lies in the limited scope of systems used for evaluation. The systems were relatively small, with few use cases, and testers not allowed to execute their tests on real systems. Instead, they described tests by specifying the action to be tested, the execution target (i.e., the node where the test will be executed), the inputs, and the expected outputs. Additionally, the focus of the evaluation was on the breadth of testing aspects covered, not the quality of the generated tests. These constraints were necessary due to the challenges of accessing large, real-world IoT systems, but they may have influenced our findings. Additionally, focusing on the breadth of testing, such as the number of aspects covered or the total number of tests created, is acceptable since the main purpose was to evaluate how the taxonomy helps discover more tests rather than assessing their quality. Another potential threat is bias in selecting survey participants, particularly their level of expertise and willingness to participate. To mitigate this, we involved professionals with experience in IoT system development or testing, verified through LinkedIn profile analysis. Subjectivity in responses to our survey can be another threat to the validity. Participants interpreted and assessed criteria (e.g., completeness, coverage, and usefulness) based on their understanding and experience in testing. Despite this concern, we accepted this threat given the number of participants involved. Lastly, we omitted security testing-related studies to manage the scope and complexity of our initial taxonomy, as security is a vast and intricate domain that would have significantly increased its length and complexity. To address this limitation, we incorporated security aspects based on survey feedback and referenced existing taxonomies on security testing as complementary to our work.

Internal Validity. Several factors could affect the internal validity of this study. Bias in the selection and grouping of testers is a potential threat, as differences in commitment, prior experience, and behavior could influence the results. Testers with more experience or time might generate more comprehensive tests even without using the taxonomy. However, our focus was on overall findings rather than individual differences, reflecting the diversity of skills and experience levels common in IoT system testing. Another potential threat is the omission of relevant IoT-specific testing aspects from the latest studies. To address this, we developed the taxonomy from a practitioner's perspective through two surveys and incorporated new aspects identified in recently published studies. The taxonomy remains a living artifact, evolving as new IoT testing aspects, tools, and insights emerge. A further threat is the use of small-scale systems in the evaluation, dictated by the unavailability of large, open-source IoT systems. Additionally, testers with varying levels of understanding and commitment may have influenced the results. Despite these limitations, the overall evaluation of our experiments showed that testers using the taxonomy identified more diverse testing aspects than those without it. Future research will address these limitations by evaluating larger and more complex systems and involving a larger group of testers.

External Validity. The generalizability of the taxonomy is limited by the size and nature of the systems evaluated, as the study did not include large, complex real-world IoT systems due to accessibility challenges. While the results demonstrate that the taxonomy can help identify more testing aspects, further validation is needed to assess its impact in larger-scale and more diverse contexts with additional testers. Broader evaluations using industry-scale systems are recommended for future research.

Conclusion Validity. This study focuses on the coverage of testing aspects rather than the quality of the generated tests. A potential limitation is the reliance on the number of tests created as a primary metric. Despite these constraints, the findings provide valuable insights into the role of the taxonomy in identifying diverse testing aspects.

10. Conclusion and future work

This study introduced a novel IoT-specific testing taxonomy, which was developed from an analysis of 83 primary studies and refined through feedback from 220 industry practitioners. Our taxonomy categorizes seven crucial aspects of IoT systems testing: objectives, tools, testers, stages, environments, objects under test, and approaches. The empirical evaluation of this taxonomy involved 24 testers across two IoT systems (WIMP and SMART-CYPS) as a case study, demonstrating that those equipped with the taxonomy could more effectively identify diverse test cases and scenarios. To facilitate its practical application,

[17] https://www.ptidej.net/Members/minanijb/Taxonomy/index.html

we provided guidelines on how to use the taxonomy and made it available online for unrestricted access. Future work will focus on enhancing the taxonomy by incorporating new aspects and details as identified. We also plan to translate the taxonomy into Japanese and distribute it to Japanese professionals involved in testing IoT systems, further broadening its applicability and impact.

11. Replication package and useful links

The replication packages, published taxonomy, and GitHub repository of this taxonomy can be accessed online.

1. Ptidej Team website
   - On Ptidej website: https://www.ptidej.net/downloads/replications/jss24a/
   - On Zenodo website: https://zenodo.org/records/14515044
2. Published taxonomy
   - Taxonomy on GitHub[18]
   - Taxonomy on Ptidej website: https://www.ptidej.net/Members/minanijb/Taxonomy/index.html

CRediT authorship contribution statement

Jean Baptiste Minani: Writing – review & editing, Writing – original draft, Visualization, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization. **Yahia El Fellah:** Writing – original draft, Validation. **Fatima Sabir:** Writing – review & editing, Data curation. **Naouel Moha:** Writing – review & editing, Supervision. **Yann-Gaël Guéhéneuc:** Writing – review & editing, Supervision. **Martin Kuradusenge:** Writing – review & editing, Validation, Investigation. **Tomoaki Masuda:** Writing – original draft, Validation, Data curation.

Declaration of competing interest

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

Acknowledgment

Thanks to all participating practitioners. This work was partially funded by the Canada Research Chair program.

Data availability

The link is shared in the paper.

References

Abdullah, Amir, Kaur, Harleen, Biswas, Ranjeet, 2020. Layers of IoT architecture and its security analysis. In: New Paradigm in Decision Science and Management: Proceedings of ICDSM 2018. Springer, pp. 293–302.

Ahmed, Bestoun S., Bures, Miroslav, Frajtak, Karel, Cerny, Tomas, 2019. Aspects of quality in Internet of Things (IoT) solutions: A systematic mapping study. IEEE Access 7, 13758–13780.

Alaqail, Hesham, Ahmed, Shakeel, 2018. Overview of software testing standard ISO/IEC/IEEE 29119. Int. J. Comput. Sci. Netw. Secur. (IJCSNS) 18 (2), 112–116.

AltexSoft, 2020. IoT architecture: Key layers and components. https://www.altexsoft.com/blog/iot-architecture-layers-components/. (Accessed 1 December 2023).

Becker, Soeren, Pfandzelter, Tobias, Japke, Nils, Bermbach, David, Kao, Odej, 2022. Network emulation in large-scale virtual edge testbeds: A note of caution and the way forward. In: 2022 IEEE International Conference on Cloud Engineering (IC2E). IEEE, pp. 1–7.

Bosmans, Stig, Mercelis, Siegfried, Denil, Joachim, Hellinckx, Peter, 2019. Testing IoT systems using a hybrid simulation based testing approach. Computing 101, 857–872.

Boukhlif, Mohamed, Kharmoum, Nassim, Hanine, Mohamed, 2024. LLMs for intelligent software testing: A comparative study. In: Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security. pp. 1–8.

Burhan, Muhammad, Rehman, Rana Asif, Khan, Bilal, Kim, Byung-Seo, 2018. IoT elements, layered architectures and security issues: A comprehensive survey. Sensors (ISSN: 1424-8220) 18 (9), http://dx.doi.org/10.3390/s18092796, URL https://www.mdpi.com/1424-8220/18/9/2796.

Cavalcante, Everton, Batista, Thais, Oquendo, Flavio, 2015. Supporting dynamic software architectures: From architectural description to implementation. In: 2015 12th Working IEEE/IFIP Conference on Software Architecture. IEEE, pp. 31–40.

Cheverda, Arina, Jabborov, Ahror, Kruglov, Artem, Succi, Giancarlo, 2022. State-of-the-art review of taxonomies for quality assessment of intelligent software systems. In: 2022 3rd International Informatics and Software Engineering Conference. IISEC, pp. 1–6.

Clerissi, Diego, Leotta, Maurizio, Reggio, Gianna, Ricca, Filippo, 2018. Towards an approach for developing and testing node-RED IoT systems. In: Proceedings of the 1st ACM SIGSOFT International Workshop on Ensemble-Based Software Engineering. pp. 1–8.

Cooke, Alison, Smith, Debbie, Booth, Andrew, 2012. Beyond PICO: the SPIDER tool for qualitative evidence synthesis. Qual. Heal. Res. 22 (10), 1435–1443.

Coppola, Riccardo, Alégroth, Emil, 2022. A taxonomy of metrics for GUI-based testing research: A systematic literature review. Inf. Softw. Technol. 107.

Costa, Victor, Girardon, Gustavo, Bernardino, Maicon, Machado, Rodrigo, Legramante, Guilherme, Neto, Anibal, Basso, Fábio Paulo, de Macedo Rodrigues, Elder, 2020. Taxonomy of performance testing tools: A systematic literature review. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. pp. 1997–2004.

Dias, João, Couto, Flavio, Paiva, Ana, Ferreira, Hugo, 2018. A brief overview of existing tools for testing the Internet of Things. In: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 104–109.

Dutta, Riya, Costa, Diego Elias, Shihab, Emad, Tajmel, Tanja, 2023. Diversity awareness in software engineering participant research. arXiv preprint arXiv:2302.00042.

Dyba, Tore, Dingsoyr, Torgeir, Hanssen, Geir K., 2007. Applying systematic reviews to diverse study types: An experience report. In: International Symposium on Empirical Software Engineering and Measurement. ESEM 2007, IEEE, pp. 225–234.

Engström, Emelie, Petersen, Kai, 2015. Mapping software testing practice with software testing research—SERP-test taxonomy. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 1–4.

Fadhil, Jawaher Abdulwahab, Sarhan, Qusay Idrees, 2022. A survey on Internet of Things (IoT) testing. In: 2022 International Conference on Computer Science and Software Engineering. CSASE, IEEE, pp. 77–83.

Felderer, Michael, Zech, Philipp, Breu, Ruth, Büchler, Matthias, Pretschner, Alexander, 2016. Model-based security testing: A taxonomy and systematic classification. Softw. Test. Verif. Reliab. 26 (2), 119–148.

Firesmith, Donald G., 2014. Common System and Software Testing Pitfalls: How to Prevent and Mitigate Them: Descriptions, Symptoms, Consequences, Causes, and Recommendations. Addison-Wesley Professional.

Firesmith, Donald, 2015. A taxonomy of testing. https://insights.sei.cmu.edu/blog/a-taxonomy-of-testing/. (Accessed 23 January 2024).

Fischbach, Jannik, Frattini, Julian, Vogelsang, Andreas, Mendez, Daniel, Unterkalmsteiner, Michael, Wehrle, Andreas, Henao, Pablo Restrepo, Yousefi, Parisa, Juricic, Tedi, Radduenz, Jeannette, et al., 2023. Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. J. Syst. Softw. 197, 111549.

Fortino, Giancarlo, Savaglio, Claudio, Spezzano, Giandomenico, Zhou, MengChu, 2020. Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools. IEEE Trans. Syst. Man, Cybernetics: Syst. 51 (1), 223–236.

Garousi, Vahid, Felderer, Michael, Karapıçak, Çağrı Murat, Yılmaz, Uğur, 2018. Testing embedded software: A survey of the literature. Inf. Softw. Technol. 104, 14–45.

Hagar, Jon Duncan, 2022. IoT System Testing. Apress.

Hassan, Muhammad, Ahmadi-Pour, Sallar, Qayyum, Khushboo, Jha, Chandan Kumar, Drechsler, Rolf, 2024. LLM-guided formal verification coupled with mutation testing. In: 2024 Design, Automation and Test in Europe Conference and Exhibition. DATE, pp. 1–2. http://dx.doi.org/10.23919/DATE58400.2024.10546729.

IEEE, 2017. ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. http://dx.doi.org/10.1109/IEEESTD.2017.8016712, ISO/IEC/IEEE 24765: 2017(E).

Inc. Cisco Systems, 2014. IoT world forum 2014. https://www.globenewswire.com/news-release/2014/10/14/1271271/0/en/The-Internet-of-Things-World-Forum-Unites-Industry-Leaders-in-Chicago-to-Accelerate-the-Adoption-of-IoT-Business-Models.HTML. (Accessed 9 December 2024).

ISO, 2022. ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 1:General concepts. pp. 1–60. http://dx.doi.org/10.1109/IEEESTD.2022.9698145, ISO/IEC/IEEE 29119-1: 2022(E).

[18] https://baptiste2k8.github.io/taxonomy4IoTTesting/

ISO Standards, 2021. IEEE/ISO/IEC International Standard - Software and systems engineering–Software testing: Test techniques. pp. 1–148. http://dx.doi.org/10.1109/IEEESTD.2021.9591574, ISO/IEC/IEEE 29119-4:2021(E).

Jean Baptiste, Minani, El Fellah, Yahia, Ahmed, Sanam, Sabir, Fatima, Moha, Naouel, Guéhéneuc, Yann-Gaël, 2024a. An exploratory study on code quality, testing, data accuracy, and practical use cases of IoT wearables. In: 7th Conference on Cloud and Internet of Things. CIoT, IEEE, pp. 1–5.

Jean Baptiste, Minani, Sabir, Fatima, El Fellah, Yahia, Moha, Naouel, 2023a. Taxonomy for IoT systems testing: Practical guidance for practitioners. Authorea Prepr.

Jean Baptiste, Minani, Sabir, Fatima, El Fellah, Yahia, Moha, Naouel, 2024b. Practical guidance for IoT systems testing: A taxonomy. p. 57, SERP4IoT 2024.

Jean Baptiste, Minani, Sabir, Fatima, Moha, Naouel, Guéhéneuc, Yann-Gaël, 2023b. A multi-method study of internet of things systems testing in industry. IEEE Internet Things J. 1. http://dx.doi.org/10.1109/JIOT.2023.3291233.

Jean Baptiste, Minani, Sabir, Fatima, Moha, Naouel, Guéhéneuc, Yann-Gaël, 2024c. A systematic review of IoT systems testing: Objectives, approaches, tools, and challenges. IEEE Trans. Softw. Eng. 1–29. http://dx.doi.org/10.1109/TSE.2024.3363611.

Jean Baptiste, Minani, Sabir, Fatima, Moha, Naouel, Guéhéneuc, Yann-Gaël, Kuradusenge, Martin, Masuda, Tomoaki, et al., 2024d. TaxIoTe: Taxonomy and practical guide for testing of IoT systems. SSRN.

Khezemi, Nour, Minani, Jean Baptiste, Sabir, Fatima, Moha, Naouel, Guéhéneuc, Yann-Gaël, El Boussaidi, Ghizlane, 2024. A systematic literature review of IoT system architectural styles and their quality requirements. IEEE Internet Things J. 11 (23), 37599–37616.

Kim, Hiun, Ahmad, Abbas, Hwang, Jaeyoung, Baqa, Hamza, Le Gall, Franck, Ortega, Miguel Angel Reina, Song, JaeSeung, 2018. IoT-TaaS: Towards a prospective IoT testing framework. IEEE Access 6, 15480–15493.

Kiran Bhagnani, Shubha Chaturvedi, 2014. Taxonomy of testing techniques. Int. J. Eng. Comput. Sci. 3 (10).

Kundisch, Dennis, Muntermann, Jan, Oberländer, Anna Maria, Rau, Daniel, Röglinger, Maximilian, Schoormann, Thorsten, Szopinski, Daniel, 2021. An update for taxonomy designers: methodological guidance from information systems research. Bus. Inf. Syst. Eng. 1–19.

Kwasnik, Barbara H., 1999. The role of classification in knowledge representation and discovery. Libr. Trends.

Ladisa, P., Plate, H., Martinez, M., Barais, O., 2023. Taxonomy of attacks on open-source software supply chains. In: 2023 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, pp. 1509–1526.

Lee, Seungjin, Park, Jaeeun, Choi, Hyungwoo, Oh, Hyeontaek, 2023. Energy-efficient AP selection using intelligent access point system to increase the lifespan of IoT devices. Sensors (ISSN: 1424-8220) 23 (11), http://dx.doi.org/10.3390/s23115197, URL https://www.mdpi.com/1424-8220/23/11/5197.

Leotta, Maurizio, Ricca, Filippo, Clerissi, Diego, Ancona, Davide, Delzanno, Giorgio, Ribaudo, Marina, Franceschini, Luca, 2017. Towards an acceptance testing approach for internet of things systems. In: ICWE Workshops. Springer International Publishing, pp. 125–138.

Lonetti, Francesca, Bertolino, Antonia, Di Giandomenico, Felicita, 2023. Model-based security testing in IoT systems: A rapid review. Inf. Softw. Technol. 107326.

Makhshari, Amir, Mesbah, Ali, 2021a. IoT bugs and development challenges. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, pp. 460–472. http://dx.doi.org/10.1109/ICSE43902.2021.00051.

Makhshari, Amir, Mesbah, Ali, 2021b. IoT bugs and development challenges. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, pp. 460–472. http://dx.doi.org/10.1109/ICSE43902.2021.00051.

Medhat, Noha, Moussa, Sherin, Badr, Nagwa, Tolba, Mohamed F., 2019. Testing Techniques in IoT-based Systems. In: 2019 Ninth International Conference on Intelligent Computing and Information Systems. pp. 394–401. http://dx.doi.org/10.1109/ICICIS46948.2019.9014711.

Mountrouidou, Xenia, Billings, Blaine, Mejia-Ricart, Luis, 2019. Not just another Internet of Things taxonomy: A method for validation of taxonomies. Internet Things 6, 100049.

Mubarakah, Naemah, et al., 2020. Software engineering taxonomy reviews. In: 2020 4rd International Conference on Electrical, Telecommunication and Computer Engineering. ELTICOM, IEEE, pp. 63–67.

Nawir, Mukrimah, Amir, Amiza, Yaakob, Naimah, Lynn, Ong Bi, 2016. Internet of Things (IoT): Taxonomy of security attacks. In: 2016 3rd International Conference on Electronic Design. ICED, IEEE, pp. 321–326.

Niederberger, Marlen, Spranger, Julia, 2020. Delphi technique in health sciences: a map. Front. Public Heal. 8, 457.

Page, Matthew J., McKenzie, Joanne E., Bossuyt, Patrick M., Boutron, Isabelle, Hoffmann, Tammy C., Mulrow, Cynthia D., Shamseer, Larissa, Tetzlaff, Jennifer M., Akl, Elie A., Brennan, Sue E., et al., 2021. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. Int. J. Surg. 88, 105906.

Pietrantuono, Roberto, Ficco, Massimo, Palmieri, Francesco, 2023. Testing the resilience of MEC-based IoT applications against resource exhaustion attacks. IEEE Trans. Dependable Secur. Comput. 21 (2), 804–818.

Pontes, Pedro Martins, Lima, Bruno, Faria, João Pascoal, 2018. Izinto: A pattern-based iot testing framework. In: Companion Proceedings for the ISSTA/ECOOP 2018 Workshops. pp. 125–131.

Raibulet, Claudia, 2018. Towards a Taxonomy for the Evaluation of Self-* Software. In: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W). pp. 22–23. http://dx.doi.org/10.1109/FAS-W.2018.00020.

Ralph, Paul, 2018. Toward methodological guidelines for process theories and taxonomies in software engineering. IEEE Trans. Softw. Eng. 45 (7), 712–735.

Rao, Tariq Aziz, Haq, E.U., 2018. Security challenges facing IoT layers and its protective measures. Int. J. Comput. Appl. 179 (27), 31–35.

Rogers, R. Owen, 2004. Acceptance testing vs. unit testing: A developer's perspective. In: Conference on Extreme Programming and Agile Methods. Springer, pp. 22–31.

Roggio, Robert, Gordon, Jamie, Comer, James, 2014. Taxonomy of common software testing terminology: Framework for key software engineering testing concepts. J. Inf. Syst. Appl. Res. 7 (2), 4.

Schäfer, Max, Nadi, Sarah, Eghbali, Aryaz, Tip, Frank, 2024. An empirical evaluation of using large language models for automated unit test generation. IEEE Trans. Softw. Eng. 50 (1), 85–105.

Stapp, Lucjan, Roman, Adam, Pilaeten, Michael, 2024. ISTQB Certified Tester Foundation Level: A Self-Study Guide Syllabus V4. 0. Springer.

Taivalsaari, Antero, Mikkonen, Tommi, 2018. On the development of IoT systems. In: 2018 Third International Conference on Fog and Mobile Edge Computing. FMEC, pp. 13–19. http://dx.doi.org/10.1109/FMEC.2018.8364039.

Tan, Teik-Boon, Cheng, Wai-Khuen, 2018. Software testing levels in internet of things (IoT) architecture. In: International Computer Symposium. Springer, pp. 385–390.

Tanenbaum, A., Steen, M.V., 2015. Introduction to distributed systems. In: Distributed Systems: Principles and Paradigms. pp. 1–33, Prentice Hall, (Jan. 15, 2002).

Tewari, Ramanuj, Alsalami, Zaid, Alsailawi, H.A., Kirubanantham, P., Dhabalia, Dharmesh, Saadoun, Osama Nazim, Abdulhussain, Zahraa N., Alwan, Ali Saad, 2024. Testing user Attitudes in IoT-load interface based healthcare attention management system. In: 2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering. ICACITE, IEEE, pp. 831–834.

Touqeer, Haseeb, Zaman, Shakir, Amin, Rashid, Hussain, Mudassar, Al-Turjman, Fadi, Bilal, Muhammad, 2021. Smart home security: challenges, issues and solutions at different IoT layers. J. Supercomput. 77 (12), 14053–14089.

Unterkalmsteiner, M., Feldt, R., Gorschek, T., 2014. A taxonomy for requirements engineering and software test alignment. ACM Trans. Softw. Eng. Methodol. 23 (2), http://dx.doi.org/10.1145/2523088.

Usman, Muhammad, Britto, Ricardo, Börstler, Jürgen, Mendes, Emilia, 2017. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. Inf. Softw. Technol. (ISSN: 0950-5849) 85, 43–59.

Utting, Mark, Pretschner, Alexander, Legeard, Bruno, 2012. A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. 22 (5), 297–312.

Vegas, Sira, Juristo, Natalia, Basili, Victor R., 2009. Maturing software engineering knowledge through classifications: A case study on unit testing techniques. IEEE Trans. Softw. Eng. 35 (4), 551–565. http://dx.doi.org/10.1109/TSE.2009.13.

Villalón, Jose Calvo-Manzano, Agustin, Gonzalo Cuevas, Gilabert, Tomás San Feliu, Puello, José de Jesús Jiménez, 2015a. A Taxonomy for Software Testing Projects. In: 2015 10th Iberian Conference on Information Systems and Technologies. CISTI, IEEE, pp. 1–6.

Villalón, Jose Calvo-Manzano, Agustin, Gonzalo Cuevas, Gilabert, Tomás San Feliu, Puello, José de Jesús Jiménez, 2015b. A taxonomy for software testing projects. In: 2015 10th Iberian Conference on Information Systems and Technologies. CISTI, IEEE, pp. 1–6.

Wang, Junjie, Huang, Yuchao, Chen, Chunyang, Liu, Zhe, Wang, Song, Wang, Qing, 2024a. Software testing with large language models: Survey, landscape, and vision. IEEE Trans. Softw. Eng.

Wang, Junjie, Huang, Yuchao, Chen, Chunyang, Liu, Zhe, Wang, Song, Wang, Qing, 2024b. Software testing with large language models: Survey, landscape, and vision. IEEE Trans. Softw. Eng.

Wang, Chunhui, Pastore, Fabrizio, Goknil, Arda, Briand, Lionel C, 2020. Automatic generation of acceptance test cases from use case specifications: an NLP-based approach. IEEE Trans. Softw. Eng. 48 (2), 585–616.

Wheaton, George R., 1968. Development of a taxonomy of human performance: A review of classificatory systems relating to tasks and performance. Sage J.

White, Gary, Nallur, Vivek, Clarke, Siobhán, 2017. Quality of service approaches in IoT: A systematic mapping. J. Syst. Softw. 132, 186–203.

Yaqoob, Ibrar, Ahmed, Ejaz, Hashem, Ibrahim Abaker Targio, Ahmed, Abdelmuttlib Ibrahim Abdalla, Gani, Abdullah, Imran, Muhammad, Guizani, Mohsen, 2017. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. IEEE Wirel. Commun. 24 (3), 10–16.

Yu, Shengcheng, Fang, Chunrong, Ling, Yuchen, Wu, Chentian, Chen, Zhenyu, 2023. LLM for test script generation and migration: Challenges, capabilities, and opportunities. In: 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security. QRS, pp. 206–217. http://dx.doi.org/10.1109/QRS60937.2023.00029.

Zander, Justyna, Schieferdecker, Ina, 2011. A taxonomy of model-based testing for embedded systems from multiple industry domains. In: Model-Based Testing for Embedded Systems.

Zander, Justyna, Schieferdecker, Ina, Mosterman, Pieter J., 2011. A taxonomy of model-based testing for embedded systems from multiple industry domains.