Jean Baptiste Minani\textsuperscript{1}, Fatima Sabir\textsuperscript{1}, Yahia El Fellah\textsuperscript{1}, and Naouel Moha\textsuperscript{1}

\textsuperscript{1}Affiliation not available

December 22, 2023
Taxonomy For IoT Systems Testing: Practical Guidance for Practitioners

Jean Baptiste Minani
CSSE Department
Concordia University
Montreal, QC, Canada

Fatima Sabir
Computer Science Dept.
University of the Punjab
Lahore, Pakistan

Yahia El Fellah
Software Engineering and IT
École de Technologie Supérieure
Montreal, QC, Canada

Naouel Moha
Software Engineering and IT
École de Technologie Supérieure
Montreal, QC, Canada

Abstract—The Internet of Things (IoT) has revolutionized the way we interact with technology and devices. Several IoT systems are being deployed across diverse domains, including but not limited to health, transportation, agriculture, and manufacturing. They fulfill critical tasks and, thus, must function correctly and securely and meet the users’ expectations. However, testing IoT systems poses many challenges, primarily due to their distributed nature, dynamism, and heterogeneity as well as the multiple layers of which they are composed, i.e., device, edge, cloud, and application layers. The absence of testing guidance can hinder the quality of IoT systems. Testing guidelines, including taxonomy, are vital for proper IoT systems testing. In the context of software testing, taxonomy organizes and categorizes testing aspects, helping testers to understand what, how, and when to test. However, no IoT systems testing taxonomy exists, and traditional software testing taxonomy may not sufficiently meet IoT systems testing requirements. To address this, we introduce an IoT-specific testing taxonomy, informed by a review of 83 primary studies and validated through surveys with 16 IoT industry practitioners. We assess its effectiveness by conducting an empirical evaluation with 12 testers. The results show that our taxonomy can help IoT testers become more efficient by fostering their understanding of various aspects of testing IoT systems. This taxonomy can help the testers to increase test coverage, enhance the efficiency and effectiveness of testing efforts, and ensure thorough testing of important system aspects, thus ensuring functional correctness, improving the security of IoT systems, and better meeting users’ expectations.

Index Terms—IoT Testing Taxonomy, IoT Testing, Software Engineering Taxonomy, Quality Assurance Taxonomy.

I. INTRODUCTION

The Internet of Things (IoT) refers to a network of physical devices that are connected and exchange data through the internet [1]. Cisco predicted that this network will consist of 500 billion devices by 2030 [2], making computing power ubiquitous across IoT systems. Ensuring the proper functioning of IoT systems is crucial due to their direct impact on personal lives and public safety [3]. The growing number of IoT systems increases the risk of connectivity and scalability challenges, underscoring the need for security and reliability, especially in safety-critical applications. While a reference model with seven layers has been proposed by Cisco, IBM, and Intel [4], there is no universally accepted IoT system architecture, as it varies based on business needs [5]. Nevertheless, many IoT systems have four layers [6]–[9]. Device, network, cloud, and application, as shown in Fig. 1.

Like traditional software, IoT systems require testing at all layers to prevent bugs that could lead to financial losses or even loss of lives of people [10]. However, testing IoT systems is still challenging due to their diverse and distributed nature, as well as testing lower layers [11], [12]. To improve test coverage, a taxonomy can be a valuable guide [13], [14]. Taxonomy can aid IoT testers in understanding and applying various testing aspects systematically, ensuring they do not overlook any critical aspect. By providing structured guidance, taxonomy enhances the quality and efficiency of IoT systems testing. At present, there exists no taxonomy tailored for testing IoT systems, which is what we aim to propose to meet the needs of practitioners.

To achieve our objective, we reviewed 83 IoT testing-related primary studies (PSs) selected from 8 digital libraries. We follow the guidelines proposed in [15]–[17]. We improved the proposed taxonomy by conducting a survey with IoT practitioners. We assessed its effectiveness by conducting an empirical evaluation with 12 testers. The purpose of this taxonomy is twofold: 1) To remove the confusion among different concepts used in IoT systems test-
ing, which can mislead the IoT systems testers. 2) To serve as a check-list for the IoT systems testers. This testing taxonomy categorizes and organizes testing aspects for improved clarity, benefiting researchers, practitioners, and future developments. The main contributions of this study are:

1. We propose a comprehensive guide to IoT testing, covering techniques, types, levels, metrics, and more, serving as valuable training material for testers and stakeholders to understand the diverse aspects of IoT systems testing.
2. We provide a structured “6Ws and 2Hs” taxonomy with relevant terminology for clarity to foster testers’ understanding and enhance the effectiveness of testing IoT systems.
3. We provide practical guidance with a use case, showing “why”, “what”, “how”, “when”, and “where” together with artifacts to be produced, bridging the gap between theory and practice for end-to-end (E2E) testing.

The rest of this paper is organized as follows: Section II discusses related work. Section III describes our research methodology. Section IV presents the taxonomy and answers to our RQs, while Section V reports our discussion. Section VI presents possible threats that could affect the validity of our study. Finally, Section VII concludes with future work.

II. RELATED WORK

Several studies proposed taxonomies for software testing to guide the testing teams. One study [18] 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 is not suitable for IoT systems testing. Another study [19] discussed 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. Another study [20] introduced a taxonomy for requirements engineering and software testing (REST). It focuses on aligning software requirements and resultant software. The proposed taxonomy was validated through an industry survey. This validation highlighted its potential to enhance both requirement engineering and software testing processes. However, it does not serve the needs of testing IoT systems. A distinct perspective emerged in the work of [21], 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. Nevertheless, this taxonomy may not be very useful for testing many aspects of IoT systems. In [22], authors 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. Another study [23] concentrated on categorizing IoT bugs but did not address actual IoT system testing. Yet another study [24] focused on taxonomy for software evaluation. The authors attempted to address “How” and “What” aspects of software testing and did not consider other aspects. Another study [25] focused on taxonomy for evaluating open-source software. However, this study did not address any aspect of IoT systems testing.

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, to ensure its alignment with industry needs and bridge the gap in IoT system testing. We assessed its effectiveness with empirical evaluation involving 12 testers.

III. RESEARCH METHOD

The methodology used in this study comprises three phases, as indicated in Fig. 2. Taxonomy development, feedback collection, and empirical evaluation.

A. Taxonomy Development

We followed preferred reporting items for systematic reviews and meta-analyses (PRISMA) [26], and reviewed scientific articles to identify different terms used in testing IoT systems. We used the mined terms to develop the initial taxonomy. Fig. 3 shows the steps we followed. 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 [27]. We defined the following search strategy: “(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 challenge OR concern OR problem OR layer OR component OR constituent OR attribute OR metric)”.

We retrieved 8,290 articles from 8 digital libraries. We removed duplicates, non-English papers, and general papers that did not focus on IoT systems testing, and screened 154 articles. We assessed the quality of 154 articles and selected 54 articles. We performed 2 rounds of backward and forward snowballing to identify an additional 29 articles, bringing the total to 83 articles included in our review. We mined the terms used in those articles to develop the initial taxonomy.

B. Collecting Feedback

We conducted a survey with 16 industry practitioners to collect feedback on our initial taxonomy. We used the collected feedback to improve our taxonomy.

1. Identify the participants. We contacted alumni groups from various universities and used social media like Facebook,
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 proposed taxonomy. The form can be accessed online.

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.

C. Conducting Empirical Evaluation

To evaluate the proposed taxonomy’s effectiveness, we conducted an empirical study with two groups, each consisting of 6 participants. We tasked both groups with creating a test strategy for a specific IoT system. One group had access to the taxonomy, while the other did not. Before the experiment, we provided both groups with the necessary system documentation. We compared the number of test cases and scenarios, and aspects identified by each group. All the materials we used can be accessed online. The complete taxonomy can be accessed online.

IV. IOT SYSTEM TESTING TAXONOMY

A. Taxonomy Descriptions

We organize the taxonomy with 6Ws and 2Hs as in Fig. 2.

1. **Why testing (W).** We begin by explaining the reasons for testing IoT systems.

2. **Which tools to use (W).** We highlight the tool categories to aid in choosing the appropriate tools for testing purposes.

3. **Who to test (W).** A person or organization that is responsible for conducting the test.

4. **When to test (W).** We summarize key stages of testing throughout the development lifecycle.

5. **Where to test (W).** We discuss the possible environment where the test will be conducted.

6. **How to test (H).** We explain different testing approaches.

https://tinyurl.com/surveyform2024
https://tinyurl.com/Taxo2024
https://sites.google.com/view/iotsystemtestingtaxonomy/home?authuser=6

1) **Why Testing (W):** This section focuses on the objective of testing as shown in Fig. 5(a). We organized the objective of testing based on the reason for testing or testing driven towards specific goals, i.e., driver-based. On reason-based, we identified three reasons: conducting initial testing, retesting, and regression testing. On driver-based, testing can focus on a specific component such as a device, network, cloud, or application. It can also focus on end-to-end (E2E) business scenarios. E2E testing can be functional (FR) or non-functional (NFR) requirements-based. For FR based, we define four drivers:

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

- **Acceptance Testing.** Acceptance testing is owned by the customer to verify that the system works in accordance with the customer’s expectations.

- **User Need Testing.** User-need testing verifies if the IoT system meets stakeholders’ needs.

- **Design Driven Testing.** Design-driven testing verifies if the IoT system conforms to the provided designs.

On NFR based, we identify the following drivers:

- **Performance Driven 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 Driven Testing.** Security testing is conducted to evaluate the degree to which the item under test (IUT), 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.

- **Interoperability Driven Testing.** It refers to the degree to which the system operates (that is, interfaces and collaborates) effectively with specified external systems. It helps to test that various devices from different vendors can communicate and understand each other.

- **Compliance Driven Testing.** Compliance testing ensures that the IUT complies with laws, standards, or regulations.
- **Integration Driven Testing.** It focuses on checking interface between different components of the IUT [29].
- **Compatibility Driven Testing.** Compatibility testing measures the degree of satisfaction to which an IUT can function satisfactorily alongside other independent products [29].
- **Usability Driven Testing.** Usability testing checks the user behavior in using the system. Testing the UI and its navigation helps elucidate if any efficient design might help to make navigation simple and desirable, as well as preventing users from doing something they should not need to do [31].
- **Reliability Driven Testing.** Reliability testing evaluates the ability of an IUT to perform its required functions, including evaluating the frequency with which failures occur, when it is used under stated conditions for a specified period [29].
- **Maintainability Driven Testing.** Maintainability testing is conducted to evaluate the effectiveness and efficiency with which an IUT may be modified [29].
- **Portability testing.** Portability testing assesses how easily an IUT 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 [32].
- **Device Life Expectancy (DLE) Driven Testing.** DLE testing is conducted to evaluate the time interval from when a device is sold to when it is discarded [33].
- **Distributivity Driven Testing.** Distributivity testing is conducted to check how components of IUT can be spread across various devices in the network [34].
- **Dynamically Driven Testing.** Dynamically testing checks how components and connectors can be created, connected, or removed during system execution [35].

2) **Which Artifacts and Tools (W):** This section focuses on various testing artifacts and tools used for testing IoT systems as shown in Fig. 5. The identified artifacts include test cases, test data, test scenarios, test scripts, defects reports, test plans, and test strategies. We define those artifacts based on ISO [36].

- **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 test cases.
- **Test Cases.** A test case is any single atomic test consisting of test preconditions, test inputs, expected test output, and expected test postconditions [30].

- **Test Data.** Test data are data used to satisfy the input requirements for executing one or more test cases.
- **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.
- **Traceability Matrix.** A traceability matrix is a form of table that links requirements to their origin and traces them throughout the project life cycle [37].

A testing tool automates various testing tasks, encompassing test execution and result recording [30]. These tools are categorized into two groups: one for running tests (e.g., Selenium, Appium) and the other for managing test environments (real testbeds or emulators/ emulators).

- **Physical Testbeds.** Testbeds that consist of hardware (physical) and software components.
- **Virtual Testbeds.** Virtual testbeds consist of real software on virtualized infrastructure [38].
- **Hybrid Testbeds.** Hybrid testbeds comprise a mix of physical hardware and virtualized components or resources.
- **Emulators.** Emulators are software tools that mimic the hardware and software of a real device [39].
- **Simulators.** Simulators are software tools that mimic the software and environment where IoT devices operate, often for testing and development purposes [39].

3) **Who Test (W):** This section focuses on the individuals responsible for testing of IoT systems as shown in Fig. 6a. We categorize those testers into two main groups: organization-based and role-based. Within the organization-based category, testing can be the responsibility of the following:

- **Developing organization.** The organizational agency 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.

On the other hand, the role-based classification includes:

- **Developer.** The developer of the system can also write and execute test cases, especially for unit tests.
- **QA Engineer.** QA engineers ensure the quality of the system through testing and process enhancement.
- **Infrastructure Engineer.** Infrastructure engineers evaluate the security, scalability, reliability, and performance of the infrastructure to ensure the system’s functionality.
- **Security Engineer.** Security engineers are responsible for testing the system’s security and addressing vulnerabilities.
- **System Administrator.** System administrators can conduct various tests (cfr. infrastructure engineer [IV-A3]) and take care of resource utilization, backup, and recovery.
multiple testing types, testing levels, testing techniques, and methodologies used to test IoT systems as indicated in Fig. 7.

- Agile-Based Testing. In agile-based testing, features are tested as they are developed.
- DevOps-Based 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.

5) Where To Test (W): This section focuses on environments to conduct the test as shown in Fig. 6c. We identified four environments: development, testing, staging, and production environment.

- Development Environment (DE). DE is the collection of hardware and software tools used to build a system.
- Testing Environment. A testing environment (also known as a testbed) is an integrated environment for testing consisting of both hardware (i.e., computers, network devices, sensors, and actuators) and software (i.e., operating systems, middleware, test tools, and test scripts) [30].
- Staging Environment. A staging environment is a copy of your production environment used just before actual deployment in a production environment.
- Production Environment. A production environment is where the latest versions of a system or updates are pushed live to the intended users.

6) How To Test (H): This section focuses on the approaches used to test IoT systems as indicated in Fig. 7. Those include testing types, testing levels, testing techniques, automation levels, and scripting levels. We will explain each category in subsequent sections.

- Level-Based Testing. There are four testing levels: unit, integration, system, and acceptance testing [39]. Unit testing checks individual IoT system components separately. Integration testing gradually tests larger subsystems as they integrate into the whole system [30]. System testing examines the entire system, not just individual components. Acceptance testing (see Section IV-A1). Regression testing verifies code changes haven’t introduced new defects [30].
- Type-Based Testing. We identified several types of testing named after their test objectives. The definitions for these testing types are provided in Section IV-A1.
- Automation Level. Test cases 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 [36].
- Scripting Level. Scripted testing follows a documented test script [36], adhering to pre-defined test cases and steps. In contrast, unscripted testing involves no formal preparation, documentation, or test scripts.
- Technique-Based Testing. 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, either functional or non-functional, with no knowledge of the internal structure or code of the component or system [30]. This testing is restricted to the externally visible behavior and characteristics of the item being tested. Combinatorial testing is a testing technique based on exercising combinations of P-V pairs [36]. Decision-table testing is a specification-based test design technique based on exercising decision rules in a decision table [36]. Scenario testing is a testing technique that uses scenarios, i.e. speculative stories, to help the tester assess a test system. UI navigation testing is a testing technique that focuses on verifying the navigation flow and user interface (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 test cases are selected just inside, on, and just outside each boundary of an equivalence class of potential test cases [30]. Equivalence class testing divides the set of test inputs into different equivalence data classes.
  - White-Box Testing. The white-box testing is based on an analysis of the internal structure and the code of a component or system under test [30]. 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 test cases. 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., sure every condition within a decision determines every possible outcome of that decision). data flow control focuses on data variables and their values to check that the value of the variable passing is correct.

- **Patterns-Based Testing.** Testing patterns are recurring, proven approaches or strategies used to design and execute effective test cases. Those patterns include **periodic readings testing** that evaluates system behavior during regular intervals. **Triggered Readings Testing** that 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 the actuators.

- **Experience-Based Testing.** Testing approaches that are based on the tester’s experience and intuition. Those include **bug hunt testing,** a structured search for defects in the absence of formal test cases. **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 [36].

- **Random Testing.** Testing techniques that are based on generating random inputs and test cases. 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 [30], [36]. Monkey testing is a method that applies random inputs to a system, aiming to crash it, without predefined test cases.

- **Verification and Validation (V&V).** Verification is any activity used to determine if the system is being built or updated correctly [30]. Verification can be performed in many ways (for example, demonstration, inspection, review, and testing). Validation is any activity used to determine if the correct system is being developed [30]. Validation can be performed using many techniques (for example, review and workshop). Static testing is the evaluation of a 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 test cases. 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 test cases systematically and automatically [36].
- **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 [36].
- **Keyword-Driven Testing.** Test cases 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 out which one performs better.
- **Functionality Testing.** Any testing intended to verify functionality by causing the implementation of a system function to fail in order to identify associated defects [30].

7) **What To Test And Metrics (W):** This section covers what to test, including the target item (specific IoT system layer or end-to-end) and the metrics as shown in Fig. 8. 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.

8) **How Well Is Test (H):** To assess how well the system is tested, we refer to the metrics in IV-A7 section.

**B. Feedback of Practitioners**

Table II shows the demographics of the practitioners who participated in our study. We used the feedback provided by 16 experts to answer RQ1, RQ2, and RQ3.

- RQ1: What is the level of understandability, completeness, usefulness, accuracy of details, and helpfulness of the pro-
posed taxonomy for IoT systems testing?

- **RQ1**: What extent does the proposed taxonomy meet practitioners’ expectations and its overall importance for IoT systems testers?

- **RQ2**: How the proposed IoT systems testing taxonomy can impact test effectiveness?

1) **Completeness and Usefulness (RQ1)**: We asked the experts to assess the completeness, level and accuracy of details, helpfulness, understandability, and usefulness of the proposed taxonomy on a scale of 1 to 5 as shown in Fig[9]

**Completeness.** 44% of the participants believed that our taxonomy is very exhaustive (scale=5 or 100%). 13% of the participants feel that our taxonomy is very complete (scale=4 or 80%), while forty-four percent (44%) of the participants believe that the taxonomy is moderately complete (scale=3 or 60%). **Coverage Accuracy.** 31% of the participants find the coverage of our taxonomy to be very accurate (scale=5 or 100%). 63% of the participants find it to be accurate (scale=4 or 80%), with only 6% of the participants finding it somewhat accurate (scale=3 or 60%). **Level of Details.** 50% of the participants confirmed that the level of detail of our taxonomy is very appropriate (scale=5 or 100%). 44% of the participants found our proposed taxonomy to be highly appropriate (scale=4 or 80%). **Helpfulness.** 44% of the participants agreed that our taxonomy is very helpful (scale=5 or 100%). 31% of the participants found it to be helpful (scale=4 or 80%). **Understandability.** 38% of the participants found our taxonomy to be excellent (scale=5 or 100 %). 62% of the participants found it to be understandable (scale=4 (80%), scale=3 (60%)). **Usefulness.** 44% of the participants found our proposed taxonomy to be very useful (scale=5 or 100%), while 56% of the participants found it to be useful (scale=4 or 80%).

The overall feedback shows that our proposed taxonomy is complete at 80%. Its coverage accuracy is 85%. Its level of detail is 86.25%. Its helpfulness is 77.5%. Its understandability is 81.25%, while its usefulness is at 88.75%.

2) **Importance and Expectations (RQ2)**: We asked the practitioners to assess the importance and the degree to which the proposed taxonomy matched the expectations of practitioners using a scale from 1 to 5 as shown in Fig[10a]. **Importance.** 63% of the participants found the proposed taxonomy to be very important (scale=5 or 100%). 13% viewed it as highly important (80%), while 44% agreed that it is important (60%). **Matching Expectation of Practitioners.** 31% of the participants found our taxonomy to be very satisfying (scale=5 or 100%). 25% of the participants found it to be highly satisfying (80%), while 44% of the participants found it to be moderate (60%). The overall feedback shows that the importance of our proposed taxonomy is at 87.5%. Matching the expectations of the practitioners is ranked at 77.5%.

3) **Effectiveness for Test Strategy (RQ3)**: We asked the practitioners to assess the effectiveness of the proposed taxonomy for testers when creating the test strategy and test plan as shown in Fig[10b]. 50% of the practitioners agree that the proposed taxonomy can extremely support (scale=5 or 100%) the testers in creating a complete test strategy. 44% of the practitioners believed that our taxonomy is highly supportive (scale=4 or 80%) for creating the test strategy and plan. Only 6% ranked the support of the proposed taxonomy at 60%. The overall feedback shows that the practitioners ranked the effectiveness of the proposed taxonomy for testers in the preparation of the test plan and test strategy at 88.75%.

---

**TABLE I: Industry Practitioners Involved**

<table>
<thead>
<tr>
<th>#</th>
<th>GNR</th>
<th>Country</th>
<th>Experience</th>
<th>Occupation</th>
<th>Educ.</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>F</td>
<td>UAE</td>
<td>5 to 10 years</td>
<td>Developer</td>
<td>Bachelor</td>
</tr>
<tr>
<td>P2</td>
<td>M</td>
<td>USA</td>
<td>1 to 3 years</td>
<td>Developer</td>
<td>Master</td>
</tr>
<tr>
<td>P3</td>
<td>F</td>
<td>USA</td>
<td>1 year</td>
<td>Project Man.</td>
<td>Master</td>
</tr>
<tr>
<td>P4</td>
<td>M</td>
<td>Canada</td>
<td>1 to 3 years</td>
<td>QAE</td>
<td>Master</td>
</tr>
<tr>
<td>P5</td>
<td>M</td>
<td>Canada</td>
<td>1 to 3 years</td>
<td>Developer</td>
<td>Master</td>
</tr>
<tr>
<td>P6</td>
<td>M</td>
<td>India</td>
<td>5 to 10 years</td>
<td>Developer</td>
<td>Master</td>
</tr>
<tr>
<td>P7</td>
<td>M</td>
<td>USA</td>
<td>1 to 3 years</td>
<td>Developer</td>
<td>Master</td>
</tr>
<tr>
<td>P8</td>
<td>M</td>
<td>Canada</td>
<td>3 to 5 years</td>
<td>Researcher</td>
<td>Master</td>
</tr>
<tr>
<td>P9</td>
<td>M</td>
<td>Japan</td>
<td>5 to 10 years</td>
<td>Project Man.</td>
<td>Master</td>
</tr>
<tr>
<td>P10</td>
<td>F</td>
<td>Pakistan</td>
<td>1 to 3 years</td>
<td>Developer</td>
<td>Bachelor</td>
</tr>
<tr>
<td>P11</td>
<td>M</td>
<td>Qatar</td>
<td>1 to 3 years</td>
<td>Team Leader</td>
<td>PhD</td>
</tr>
<tr>
<td>P12</td>
<td>M</td>
<td>Pakistan</td>
<td>1 year</td>
<td>QAE</td>
<td>Master</td>
</tr>
<tr>
<td>P13</td>
<td>M</td>
<td>Pakistan</td>
<td>1 year</td>
<td>QAE</td>
<td>Master</td>
</tr>
<tr>
<td>P14</td>
<td>M</td>
<td>Pakistan</td>
<td>3 to 5 years</td>
<td>Developer</td>
<td>Master</td>
</tr>
<tr>
<td>P15</td>
<td>M</td>
<td>Rwanda</td>
<td>5 to 10 years</td>
<td>Developer</td>
<td>Bachelor</td>
</tr>
<tr>
<td>P16</td>
<td>F</td>
<td>India</td>
<td>1 year</td>
<td>QAE</td>
<td>Master</td>
</tr>
</tbody>
</table>

* GNR: Gender; QAE: QA Engineer.
C. Usage Scenario

We use test strategy preparation scenario for illustration.

1. **Define test objective - “Why”**? Testers can refer to “Why” part of the taxonomy to define the objective of testing.
2. **Determine the scope - “What”**? Testers can define what will be tested by referring to “what” part of the taxonomy to define what will be tested and what will not be tested.
3. **Define approach - “How”**? Testers can specify the testing types or techniques by referring to Fig. 7. For instance, if testers select “type-based testing” as a category, they can choose from options like “performance testing”, where they can indicate one or more of the presented options such as: “stress testing, or volume testing”.
4. **Define required resources - “Who”**? The taxonomy presents two options: organization-based testing, where the organization entity to conduct test is known, or role-based testing, where the project team specifies the person to conduct the test such as “developer” or “QA engineer”.
5. **Environment - “Where”**? A test can be conducted in testing environments. This can be physical, virtual, or hybrid.
6. **When to test - “When”**? Testing can occur at the end of a specific SDLC phase or continuously in Agile or DevOps. In a phase-based approach, like the waterfall model, testing is assumed to take place at the end of the coding phase.
7. **Tools and artifacts - “Which”**? The testing team should select testing tools. For instance, if they choose Selenium to test the application layer and set up a physical testing environment with smart devices and a Raspberry Pi, they may focus on a few specific artifacts, such as the test plan, defect report, and test cases.
8. **Metrics - “How-Well”**? Depending on the test target and approach, the tester can choose relevant metrics. For instance, in functional testing, the tester may use metrics such as the number of identified bugs, executed test cases, or covered requirements to gauge the system’s development quality.

With the guidance of this taxonomy, the team must be able to define the testing approach, environment, method, level, type, quality attributes, artifacts, coverage, test layer, tools, focus, and patterns if applicable. This taxonomy can help the testers to write a clear and effective testing plan.

For RQ1, RQ2, and RQ3, we relied on practitioners’ feedback on our initial taxonomy. The version presented in this paper includes all their feedback.

D. Empirical Evaluation

We conducted the experiments to assess how our taxonomy can help the testers be more efficient and improve test coverage (RQ4). We used WIMP (Where Is My Professor) as a case study. WIMP is an IoT system designed to help students track the availability of their professors in real-time based on the data collected from IoT devices. Using the collected data, WIMP can provide automated responses regarding a professor’s availability. More details about WIMP can be found online. 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. Table [IV] shows some details of the testers as recommended in [14]. We asked each participant to study and prepare test scenarios (TSs) and test cases (TCs) for two modules, with each participant given 2 hours maximum.

Table [IV] shows the number of TSs and TCs identified by each participant. We observed that the group that used the proposed 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. Our empirical results show that our
Construct Validity. Construct validity concerns sources and data collection, including the selection of primary studies and terminology mining from them. To address this, we adhered to PRISMA guidelines and utilized recommended databases [27]. Another concern is potential bias in selecting survey participants, their expertise, and their willingness to participate. We mitigated this by involving professionals experienced in developing and testing IoT systems and by not offering financial incentives.

Internal Validity. One potential threat to validity is the possibility of omitting relevant IoT systems testing terms and aspects. To address this, we systematically reviewed published studies and consulted 16 industry experts through a survey. The size and expertise of our empirical evaluation participants raise another validity concern. However, given the difficulty in finding testers, and considering our aim to assist practitioners new to IoT testing, we deem this trade-off acceptable. We also considered the assumption that testers have the same level of understanding and commitment in our experiments. This is justifiable, as our primary goal was to evaluate the overall impact of our taxonomy. Lastly, the use of small-scale systems in our empirical evaluation poses a validity threat due to the unavailability of open-source IoT systems. In our future research, we will address this by conducting empirical studies with larger systems.

External Validity. The proposed taxonomy offers insights into testing IoT systems, but its adaptability across various scenarios may not have been extensively examined. While we didn’t identify particular threats to its generalizability, we acknowledge its potential variations in diverse contexts.

Conclusion Validity. Our conclusions are based on reviewed studies, which may have contradictory information. We addressed this through surveys, involving experts.

V. DISCUSSIONS

On feedback from practitioners. We created an initial taxonomy based on PSs and refined it through collaboration with industry experts to address the practical needs of IoT testing.

Fragmented IoT system testing aspects. Existing studies, including [39], [41] and [42], [43], provided a taxonomy focusing solely on testing types. Notably, a study such as [44] focused on bug taxonomy within IoT systems. Prior to our work, no IoT system testing taxonomy existed. Ours is the first, and it is also adaptable for traditional software testing.

PMI Guideline. Our taxonomy aligns with guidelines from the Project Management Institute (PMI) [45] for various testing aspects such as approaches, test environments, and testing types. Those aspects are vital for testing IoT systems and this taxonomy can underscore those guidelines.

Guidance for testing IoT systems. We noticed that many studies often use some terms interchangeably, causing confusion for testers. We proposed definitions to clarify those concepts for better understanding. Yet, our taxonomy may not be exhaustive, it can evolve with the addition of new concepts.

VI. THREATS TO VALIDITY

There are threats to the validity of this study in terms of construct validity, internal validity, and external validity.

Construct Validity. Construct validity concerns sources and data collection, including the selection of primary studies and terminology mining from them. To address this, we adhered to PRISMA guidelines and utilized recommended databases [27]. Another concern is potential bias in selecting survey participants, their expertise, and their willingness to participate. We mitigated this by involving professionals experienced in developing and testing IoT systems and by not offering financial incentives.

Internal Validity. One potential threat to validity is the possibility of omitting relevant IoT systems testing terms and aspects. To address this, we systematically reviewed published studies and consulted 16 industry experts through a survey. The size and expertise of our empirical evaluation participants raise another validity concern. However, given the difficulty in finding testers, and considering our aim to assist practitioners new to IoT testing, we deem this trade-off acceptable. We also considered the assumption that testers have the same level of understanding and commitment in our experiments. This is justifiable, as our primary goal was to evaluate the overall impact of our taxonomy. Lastly, the use of small-scale systems in our empirical evaluation poses a validity threat due to the unavailability of open-source IoT systems. In our future research, we will address this by conducting empirical studies with larger systems.

External Validity. The proposed taxonomy offers insights into testing IoT systems, but its adaptability across various scenarios may not have been extensively examined. While we didn’t identify particular threats to its generalizability, we acknowledge its potential variations in diverse contexts.

Conclusion Validity. Our conclusions are based on reviewed studies, which may have contradictory information. We addressed this through surveys, involving experts.

VII. CONCLUSION AND FUTURE WORK

In this study, we proposed a taxonomy for testing IoT systems by organizing different testing aspects to guide the testers. We validated our taxonomy with input from industry practitioners, enhancing its completeness, coverage, and understandability. We conducted an empirical study involving 12 testers and the WIMP as a case study. The results showed that testers using the taxonomy could assess a wider range of aspects, resulting in the identification of more TSs and TCs. We believe that the proposed taxonomy will contribute to the advancement of IoT systems testing and ultimately improve the quality and performance of IoT systems. In our future work, we will continue to enhance this taxonomy by incorporating new aspects and details as they are identified. We will also conduct empirical studies on larger systems.

ACKNOWLEDGMENT

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

REFERENCES
