






# Testing Practices, Challenges, and Developer Perspectives in Open-Source IoT Platforms


1<sup>st</sup> Daniel Rodriguez-Cardenas   
 William & Mary  
 Williamsburg, VA, USA  
 dhrodriguezcar@wm.edu

2<sup>nd</sup> Safwat Ali Khan   
 George Mason University  
 Fairfax, VA, USA  
 skhan89@gmu.edu

3<sup>rd</sup> Prianka Mandal   
 William & Mary  
 Williamsburg, VA, USA  
 pmandal@wm.edu

4<sup>th</sup> Adwait Nadkarni   
 William & Mary  
 Williamsburg, VA, USA  
 apnadkarni@wm.edu

5<sup>th</sup> Kevin Moran   
 University of Central Florida  
 Orlando, FL, USA  
 kpmoran@wm.edu

6<sup>th</sup> Denys Poshyanyk   
 William & Mary  
 Williamsburg, VA, USA  
 denys@cs.wm.edu

**Abstract**—As the popularity of Internet of Things (IoT) platforms grows, users gain unprecedented control over their homes, health monitoring, and daily task automation. However, the testing of software for these platforms poses significant challenges due to their diverse composition, *e.g.*, common smart home platforms are often composed of varied types of devices that use a diverse array of communication protocols, connections to mobile apps, cloud services, as well as integration among various platforms. This paper is the first to uncover both the practices and perceptions behind testing in IoT platforms, particularly open-source smart home platforms. Our study is composed of two key components. First, we mine and empirically analyze the code and integrations of two highly popular and well maintained open-source IoT platforms, *OpenHAB* and *HomeAssistant*. Our analysis involves the identification of functional and related test methods based on the *focal method approach*. We find that *OpenHAB* has only 0.04 test ratio ( $\approx 4K$  focal test methods from  $\approx 76K$  functional methods) in Java files, while *HomeAssistant* exhibits higher test ratio of 0.42, which reveals a significant dearth of testing. Second, to understand the developers’ perspective on testing in IoT, and to explain our empirical observations, we survey 80 open-source developers actively engaged in IoT platform development. Our analysis of survey responses reveals a significant focus on automated (unit) testing, and a lack of manual testing, which supports our empirical observations, as well as testing challenges specific to IoT. Together, our empirical analysis and survey yield 10 key findings that uncover the current state of testing in IoT platforms, and reveal key perceptions and challenges. These findings provide valuable guidance to the research community in navigating the complexities of effectively testing IoT platforms.

**Index Terms**—Internet of Things, Software Testing, Maintenance, Unit test, Developer study

## I. INTRODUCTION

The Internet of Things (IoT) platforms have proliferated the broader computing landscape in recent years and have ushered in new types of digital interactions. Some estimates project that the number of IoT devices will eclipse nearly 30 billion by 2030 [1]. IoT devices power increasingly popular smart home ecosystems that aid users in automating daily tasks ranging from controlling lights to implementing home security [2], [3]. IoT devices have also enabled new types of digital health

ecosystems that help users to track their fitness and body [4], and have even begun to power utilities at the scale of cities [5]. While all of these applications of IoT devices and ecosystems have afforded novel, convenient computing paradigms, the engineering of such systems is not without its challenges.

IoT ecosystems are inherently heterogeneous by nature and comprise smartphones, servers, devices, communication hubs, online services, and end-user applications (see Fig. 1). The design and engineering of each of the individual components of these ecosystems, alongside their integration, pose significant challenges to the design, implementation, maintenance, testing, and evolution of their software components [6].

Software testing practices, in particular, are important for IoT systems to ensure that both the functional and security/safety-related requirements are properly met. While prior studies have examined common bugs and general software development challenges for IoT platforms [6]–[8], we know little about the current state of software testing in consumer-oriented IoT platforms such as smart homes. That is, the research community has a limited understanding of typical software testing activities, processes, deficiencies, and challenges that engineers currently face while working across all of the typical components of IoT platforms.

To address this research gap, this paper presents an in-depth study of the testing practices, and perspectives, in open source smart home platforms. Our study has two major components. First, we mine and empirically analyze the testing-related code ( $\approx 37K$  test methods, from 12,904 test files) of two of the largest and most active open-source IoT platforms, *OpenHAB* [9] and *HomeAssistant* [10]. Our choice to study these two popular platforms in-depth was guided by two key study design considerations: (i) they represent testing practices across the range of IoT platform components, including, but not limited to, core platform code, device integrations, and integrations for online services, and (ii) they are entirely open source, *i.e.*, not only is the platform code available, but so are the integrations, which are critical from a testing perspective. Using a rigorous, systematic, open-coding process,

we identify the key purposes of testing-related activities and their prevalence across our studied platforms. Moreover, we quantify test coverage by defining the metric of *test ratio*, which leverages the notion of the *focal method (fm)* (i.e., a method with defined parameters, which is to be tested [11]) and the *focal test method (ftm)*, i.e., a method to test all or part of an *fm*). Intuitively, the test ratio can be defined as the ratio  $ftm/ftm + fm$  (see Section II for a detailed overview of *fm* and *ftm*).

Second, we conducted a survey of 80 open-source developers who work on IoT platforms and asked them about their testing practices, experiences, and perspectives regarding the testing of IoT platforms. A systematic, thematic analysis of these survey responses helped us understand the *rationale* behind current testing practices and pain points experienced by practitioners, and provided actionable insights.

Our study resulted in 10 *key findings* ( $\mathcal{F}_1 - \mathcal{F}_{10}$ ) that characterize the current landscape of software developers' priorities ( $\mathcal{F}_2, \mathcal{F}_8, \mathcal{F}_9, \mathcal{F}_{10}$ ), preferred approaches and perceptions regarding them ( $\mathcal{F}_3, \mathcal{F}_4$ ), and challenges they face ( $\mathcal{F}_5, \mathcal{F}_6, \mathcal{F}_7$ ) when testing IoT platforms. Particularly, we find a significant lack of testing of both platform components and integrations ( $\mathcal{F}_1$ ), characterized by their computed *test ratio*. A majority, i.e., sensors, control-state, sensor components and integrations with popular *add-ons* such as 'Hue', 'Nest' [12], 'SamsungTV', 'homekit' *add-ons* fall below the test ratio of 0.5. Of *HomeAssistant's* 937 *add-ons*, 610 or 65% have a test ratio of below 0.5 and an average test ratio of 0.42 ( $\mathcal{F}_1$ ). Our analysis also reveals what types of test targets developers prioritize, e.g., *OpenHAB* reports 16 *add-ons* tests, 12 network tests, and includes rule-based and authentication tests ( $\mathcal{F}_2$ ).

Our analysis of survey responses reveals that developers generally focus on performance, scalability, real-world scenarios, real-time collaboration, and security testing ( $\mathcal{F}_8, \mathcal{F}_{10}$ ). We find that while automated testing is the preferred choice among participants for identifying defects early with enhanced reliability ( $\mathcal{F}_3$ ), some developers advocate manual testing for human intuition and adaptability in real-world scenarios ( $\mathcal{F}_4$ ). Developers also expressed significant challenges unique to IoT platforms, particularly the validation of compatibility across devices and platforms, debugging of firmware updates, and the challenge of addressing issues in low-power mode ( $\mathcal{F}_5 - \mathcal{F}_7$ ). Finally, a few developers express concerns about poor documentation and the absence of organizational standards for testing IoT platforms ( $\mathcal{F}_9$ ).

In summary, this paper makes the following contributions:

- An empirical investigation into the purpose of testing related code in the *OpenHAB* and *HomeAssistant* open source projects.
- A survey and thematic analysis of responses from 80 open source developers who work on IoT ecosystems, to investigate developer perspectives and experiences regarding testing practices, tools, processes, and challenges.
- Ten findings derived from the above two studies that inform important directions for future research.

- A replication package that contains all of our data, analysis code, and qualitative results to facilitate reproduction and replication of our work [13].

## II. BACKGROUND

The Internet of Things (IoT) represents the interconnectedness of everyday physical objects or 'things' via the internet. These objects are equipped with sensors, gateways, software, hubs, and other technologies, enabling them to collect and exchange data not only among themselves but also with other systems and applications via the Internet. In our study, we outline our research questions concerning the landscape of the most common IoT platforms, practices, and practitioners' perspectives. This section offers an overview of IoT platforms, including their layers, components, and common tests.

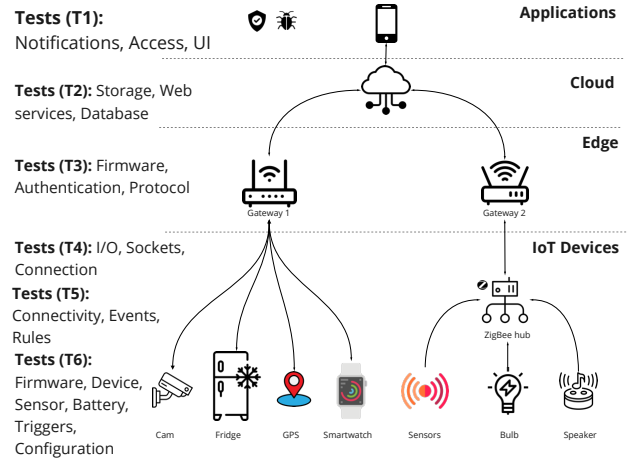


Fig. 1: IoT platform layers and common tests per layer [6]

**IoT platforms.** An IoT platform is a software suite that provides the framework and tools to facilitate the development, and management of applications and connected devices. Those platforms and devices constitute an ecosystem that automates places like homes. In general, any IoT platform follows the architecture and layers described in Fig. 1 [6]. Fig. 1 depicts four layers (i.e., IoT devices, edge, cloud, and applications). The *IoT devices* layer involves smart programmable devices interacting with the physical world through embedded sensors and actuators. The *edge* layer consists of gateway devices with fewer resource constraints, capable of locally handling telemetry data collection, processing, and routing. These gateways interpret diverse communication protocols like MQTT, CoAP, and HTTP, managing device-device and device-cloud interoperability. The *cloud* layer comprises remote IoT cloud servers that accumulate and process telemetry data, communicating with heterogeneous IoT devices for remote control and monitoring. IoT cloud servers utilize a rule engine to enable users to write automation logic, defining interoperability behaviors within the IoT system. Finally, a user can utilize *applications (apps)* to interact and control the IoT devices.

In our research, we evaluated open-source IoT platforms, ultimately choosing *OpenHAB* [9] and *HomeAssistant* because

of their widespread adoption. Both platforms are vendor-agnostic supporting a large range of devices and network protocols and allowing to connect and interact with products from different manufacturers (*i.e.*, Amazon, Alexa, Nest, Hue, Homekit, etc). Both platforms allow users to create complex interaction rules (*i.e.*, turn off the lights at a given hour if the motion sensor is not activated) using a user interface. Both platforms are also actively supported by community and diverse developers offering a wide range of add-ons, integrations, and custom components.

**IoT testing.** Each layer within the IoT platform is susceptible to failure, and the impact can cascade to subsequent layers. Moreover, IoT devices, often deployed in unpredictable and challenging environments, may encounter extreme conditions. Potential tests are outlined in Fig. 1 are listed based on the most common test for software components [14].

*T1: Application Tests* - This layer focuses on user experience and interface testing to facilitate coordination and access to IoT devices. Common tests include UI/UX assessments, notification functionalities, and smartphone integration [14].

*T2: Cloud Tests* - IoT device data is transmitted to cloud-based platforms for storage, analysis, and accessibility. Tests in this layer typically cover storage integrity, database access, and web service functionalities.

*T3: Communication Protocols* - Various communication protocols such as Wi-Fi, Bluetooth, and cellular networks enable IoT devices to connect to the internet and other devices. Tests here include router firmware evaluations, network authentication, and protocol compliance checks.

*T4: Network Connectivity* - IoT devices require stable connections to broader networks through standard protocols. Tests focus on input/output data validation, socket connections, and data delivery reliability.

*T5: Intermediate Hubs* - Intermediate hubs facilitate device control with specific requirements. Tests involve event validation, connectivity checks, and rules configuration, where rules define configurable routines for device actions.

*T6: Firmware and Resource Optimization* - IoT devices utilize firmware to monitor, control, and optimize processes. Tests concentrate on optimizing limited resources such as battery and memory, alongside fundamental configuration and trigger event validations.

A common functional software artifact deployed on each device is the *add-ons*. For instance, devices such as cameras, sensors, and speakers can be seamlessly integrated into the platform as *add-ons*. Each *add-ons* comprises multiple software *components*, including but not limited to battery management, sensor control, event handling, data collection, and network connectivity. These *components* serve as the fundamental *building blocks across all IoT layers*. Consequently, components are omnipresent throughout the IoT platforms. Each platform can have collaborations with private companies such as Hue, Amazon Alexa, or Apple HomeKit.

**Focal Methods.** Our goal is to identify testing practices across widely-used open-source IoT platforms. One common approach for assessing test coverage involves running tests

and determining which functional blocks and lines of code are executed. However, applying this method across diverse programming languages, add-ons, and core components— in IoT platforms like *OpenHAB* and *HomeAssistant* — poses significant challenges. Executing test coverage in these environments requires configuring various files, setting up specific environments, and integrating multiple devices for testing purposes [15], [16].

Our investigation adopts an agnostic approach to identify tests within IoT add-ons and component code. To achieve this, we utilize the concept of focal methods to pinpoint functional and test methods. According to Tufano et al. [11], a focal method (*fm*) refers to a method within a functional block that has defined parameters. Each *fm* can be tested by zero or *n* focal test methods. A focal test method (*ftm*) is a functional code block specifically designed to test either a portion or the entirety of an *fm*. The primary purpose of *ftms* is to capture the intent and structure of human-written tests, ensuring traceability between tests and code under test.

Focal methods provide a mechanism for identifying both functional methods and their corresponding tests within a component. The concept of a *fm* (Fig. 2) is based on the principle that all code is organized into files, and these files follow a structured format dictated by the syntax of the programming language (PL) [11]. In object-oriented PLs (*e.g.*, Java), this structure is particularly clear, and five levels of context can be identified:

*Focal Method (fm)*: this first level represents a set of public methods with bodies that contain the most critical information for generating test cases. These are the functional code blocks that can be executed and tested.

*Class Context (fm + c)*: the second level adds the class name to the focal method, providing the necessary context about the class to which the method belongs.

*Class Constructor*: includes the class constructor, which offers details on how to instantiate the class to enable testing.

*Auxiliary Methods*: comprise supporting methods (*e.g.*, getters and setters) that are required to invoke the focal method.

*Class Attributes*: encompass the set of attributes at the class level, which contribute to the functionality of the class.

### III. RESEARCH QUESTIONS

In this research, we seek to learn more about IoT developers' perspectives regarding their test objective, common practices and techniques, test case design decisions, and challenges they face. By understanding what components are commonly tested, and the techniques used, we can not only gain insights into what areas might benefit from the development of new tools or techniques but also identify which parts of the system developers view as most critical or complex. This will guide the efficient allocation of testing resources and better focus future testing strategies, ultimately improving the reliability and performance of IoT applications. To evaluate how developers are currently testing IoT platforms, the practices they follow, and the challenges they face, we pose the following RQs:

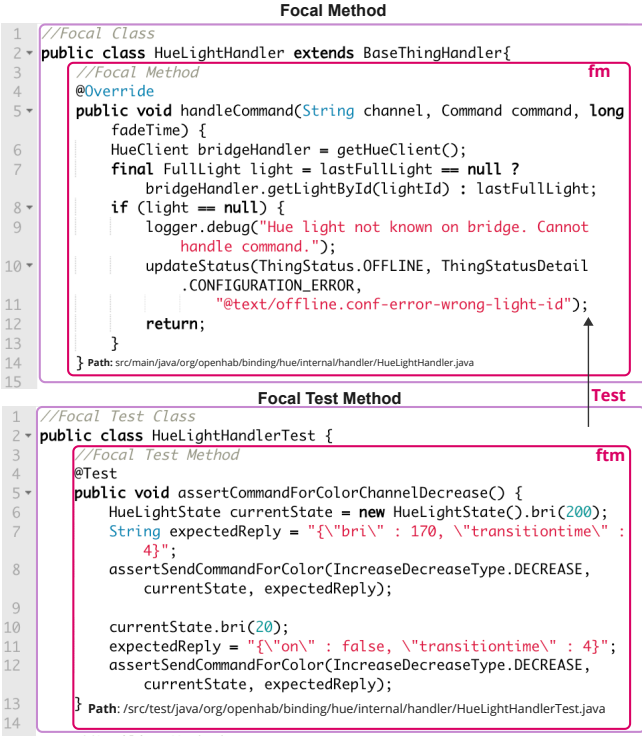


Fig. 2: Focal method (fm) and focal test method (ftm) example.

**RQ<sub>1</sub>** What are the proportion of focal test methods for add-ons and commonly components in open-source IoT platforms?

**RQ<sub>2</sub>** What is the purpose of current testing-related code and processes in open-source IoT platforms?

**RQ<sub>3</sub>** How do developers design tests for IoT platforms?

**RQ<sub>4</sub>** What tools, techniques, and processes do developers currently employ for testing IoT platforms?

**RQ<sub>5</sub>** What challenges do developers face when testing IoT platforms?

#### IV. IOT PLATFORM ANALYSIS

The following section describes the steps for collecting, curating, standardizing, and labeling each component from *OpenHAB* [9] and *HomeAssistant* [10] as well as includes the results from our analysis.

##### A. Methodology

To answer our **RQ<sub>1</sub>** we designed a semi-automated pipeline to collect and classify testing artifacts we found from *OpenHAB* and *HomeAssistant*.

1) *Data extraction*: Our data is collected from the public GitHub repositories reported at *HomeAssistant* and *OpenHAB* platforms (see Figure 3 ①). Each platform reports a set of projects depending on the core architecture, brands, or operative systems. Notably, each platform adopts a distinct code organization and architecture. For instance, *HomeAssistant*, designates a *test* directory within its repository, exclusively housing test codes for each *add-ons*. While *OpenHAB* developers

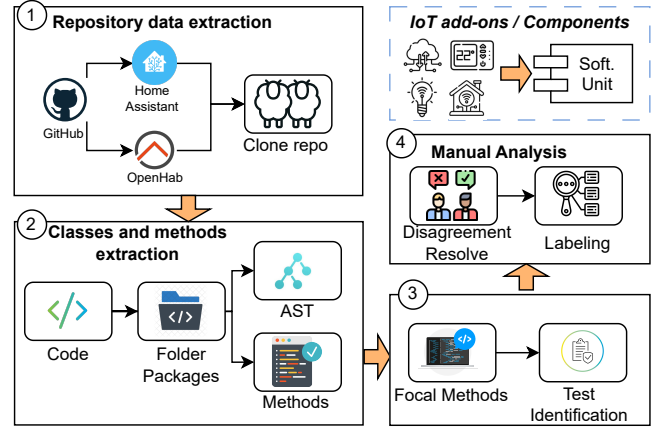


Fig. 3: *HomeAssistant* and *OpenHAB* platform analysis steps.

maintain test codes for each *add-ons* within their project folders. *OpenHAB*'s repositories exhibit a variety of code languages, with Java being the most prevalent, accompanied by instances of Python, JavaScript, and Kotlin (see Table I). We noticed that *OpenHAB* lacks a standardized naming convention for Java classes, with names such as "Test", "Stub", or "Mock", adding a layer of complexity to the analysis.

2) *Automated test identification*: The step-② includes the identification of the source folder, classes, and methods. We use regex expressions to isolate folders with functional code from tests. For instance, Java-based *add-ons*, usually match the test folder and file name (e.g., *src/test/java/FooTest.java*) to the corresponding functional code path (e.g., *src/main/java/Foo.java*). Fig. 2 illustrates a linked *ftm* with the correspondent *fm* *HueLightHandler* class. That is not the case for different PLs like JavaScript or Python. In that case, we look for name matching, for example, *x.python* is tested by *x\_test.python*. We also aim to identify classes and methods via abstract syntax tree (AST) with tree-sitter tool [17]. The AST helps to confirm a valid code file and identifies structures, such as the methods, functions, and variables.

3) *Test analysis*: **RQ<sub>1</sub>** requires ascertaining the number of tests per component and their proportional relationship with the functional code. Therefore, we focused on identifying *fm* and *ftm* as described in Sec. II at step-③. A *fm* exclusively incorporates the signature, parameters, and body function. To identify a test we look not only to the test folder and files but also a *ftm* inside each file (i.e., method within a test class with the *@Test* annotation) ( Fig. 3). Once we identify the *focal methods* and *focal test methods* per file, we count them to obtain a proportion by file, component, *add-ons*, and platform.

4) *Manual Analysis*: The fourth step-④ aims to validate the identification of focal methods and focal test methods. It allows us to confirm our proportion number to answer **RQ<sub>1</sub>** and also enables the labeling process to answer **RQ<sub>2</sub>**.

To inspect the *OpenHAB* platform, we selected the top 36 *add-ons* ranked by the highest count of *fm* and classes. Then we analyzed the top 10 test files with the most significant number



of methods and classes. In this way, we ended up manually analyzing a total of 360 test files from a total of 5,597 test files. Labeling is about assigning a common category to the given *ftm* intention. In this step, two authors look through the *ftm* to describe their intended purpose and functionalities. The authors scrutinize each test file to identify the components under examination and assign multiple labels accordingly. For instance, in files such as `test_sensor.py`, developers frequently assess functionalities such as air quality, temperature, storage component setups, signal strength, network speed, voltage stability, and battery levels. These features are then generalized by labeling them as 'sensor', 'storage', and 'network' (see Figure 4).

The manual inspection procedure for *Home Assistant* includes an additional step. We observed that *HomeAssistant* repeats a set of filenames across multiple *add-ons*. For instance, `test_config.py` appears in 552 different *add-ons*, and `test_sensor.py` appears in 267 *add-ons*. The test files with the same name serve similar testing objectives over different *add-ons*; For example, `test_notify.py` tests various types of asynchronous notifications for both Google Mail and Slack components and *add-ons*. Leveraging this fact, we narrowed our focus to the top 43 filenames, each appearing more than 10 times. For each of these 43 filenames, we randomly selected 10 instances and conducted a manual inspection of the source code. We selected the top filenames since they represent the biggest IoT systems and 10 is the average to frequency. This yielded a total of 430 manually inspected files for *HomeAssistant*. The number of inspected files for *OpenHAB* and *HomeAssistant* constitutes a representative sample, providing a confidence level (z-score) above 95% (see Tab. I).

Table I summarizes our test file selection process. The labeling process was executed in two phases. After each phase, the authors met to finalize the labels and resolve disagreements. This collaborative approach ensured alignment and prevented mislabeling or focus on incorrect label categories.

### B. Results from the Analysis

In this section, we present our findings after applying our methodology for extracting both *focal methods* (*fm*) and *focal test methods* (*ftm*) for *OpenHAB* and *HomeAssistant* platforms. Table I outlines popular *add-ons* for each platform ordered by their number of source files. Some *add-ons* are well-known brands such as Alexa [18], Nest [12], or Homekit [19]. We calculate the test ratio as  $ftm/(ftm + fm)$ , with a threshold of 0.5 to flag low scores. While this threshold doesn't guarantee each *fm* has a corresponding *ftm*, it helps assess the *ftm* distribution. We report the average and standard deviation.

In our analysis of *OpenHAB*, we identified 406 *add-ons* with a total of  $\approx 7K$  source files. We observe a low test ratio averaging only 0.04. That means that there are more functional methods than methods for testing them. Notably, some components like SamsungTV lack any identified *ftm* in Java versions. We found a total of  $\approx 76K$  *fm*, but just identified  $\approx 4K$  *ftm*. Additionally, our analysis extended to extra *OpenHAB* repositories not included as *add-ons* in the

TABLE I: Most common *add-ons* per platform and test ratio

OpenHab common add-ons test ratio score					
Add-on	Source files	Test files	fm	ftm	Test Ratio
tapocontrol	8	8	399	35	0.08
enigma2	6	6	100	109	0.52
mielecloud	66	65	766	632	0.45
loxone	32	31	274	196	0.42
omnikinverter	10	1	39	24	0.38
wemo	52	11	199	58	0.23
nest	109	22	562	109	0.16
hue	12	11	982	125	0.11
irobot	1	1	74	6	0.08
samsungtv	32	32	218	0	0.00
Total	7071	5597	76453	4585	-
avg[std]	-	39.41[68.65]	188.31[220.86]	11.29[43.03]	0.04[0.09]
min - max	-	1 - 632	0 - 635	0 - 635	0 - 0.60
Home Assistant common add-ons test ratio score					
recorder	57	51	714	1114	0.61
tplink	23	27	108	156	0.59
template	26	34	304	331	0.52
hassio	24	20	235	247	0.51
unifiprotect	29	33	260	261	0.50
esphome	39	39	318	306	0.49
zha	36	33	348	332	0.49
matter	29	69	175	166	0.49
group	22	21	207	195	0.49
homekit	27	30	316	296	0.48
Total	9176	7307	40480	32735	-
avg[std]	9.80[5.92]	7.81[9.16]	43.24[60.84]	34.97[79.47]	0.42[0.16]
min - max	2 - 57	0 - 110	1 - 714	0 - 1654	0.02 - 0.88

\* *add-ons* with more than 20 source files for *HomeAssistant*. Gray rows indicate well-known components. Underscore indicates above the threshold. Bottom the total, avg, min, and max number of elements across all *add-ons*.

main project. For example, we observe that Z-Wave and Alexa exhibit test ratios of 0.25 and 0.2, respectively.

For *HomeAssistant*, we observe higher test ratios observing an average of 0.42 but with a high variability of 0.16. Therefore, we could observe some components with even more *ftm* than *fm* (i.e., recorder) but some components with a low test ratio (e.g., homekit). Interestingly, well-known brands like Alexa and google\_assistant both with 0.38 of test ratio. We also noticed a lower number of test classes. This is due to the lack of class definitions, however, we identified  $\approx 40K$  *fm* and  $\approx 32K$  *ftm* across the total of *add-ons*.

In our analysis for *HomeAssistant*, we identified that 937 *add-ons* which contained test codes, only 327 of those have a test ratio above the threshold of 0.5. We select the 0.5 threshold as the upper values represent beyond the half proportion on tested code, however, practitioners can use higher thresholds. In other words, *HomeAssistant* reports 65% of *add-ons* with a poor number of tested methods. The same analysis for *OpenHAB* reports that just 3 *add-ons* achieve this threshold.

**Finding 1 ( $\mathcal{F}_1$ )** – Only 3 out of 406 *OpenHAB* *add-ons* surpass our test ratio threshold of 0.5. *HomeAssistant* has 327 *add-ons* that have a better test over the 0.5 threshold but reports high variability with a standard deviation of 0.16.

Figure 4 on the left, depicts the test ratio among the *add-ons* and the labeling components within *add-ons* on the right. The test ratio distribution demonstrates a larger number of implemented *add-ons* for *HomeAssistant* and a stable number of *ftm* above 0.2 but below 0.5. Nevertheless, *OpenHAB* test ratio is mostly below 0.2. *HomeAssistant*'s exclusive tested components like 'switch', 'button', 'cover', and 'trigger' indicate the platform's concentrated efforts towards refining user-facing elements and interaction mechanisms. In contrast, *OpenHAB*'s exclusive tested components such as 'rule',

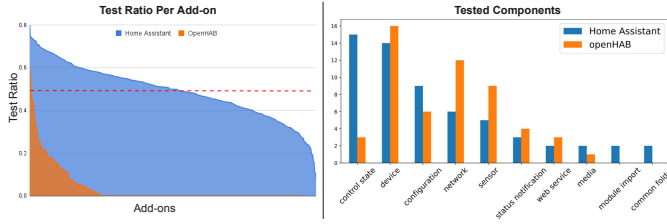


Fig. 4: Left: Test ratio per *add-ons*. Right: Top ten tested components for *HomeAssistant* and *OpenHAB*

'authentication', and 'status' reflect the platform's strong emphasis on rule-based automation, security, and system organization.

The components common to both *HomeAssistant* and *OpenHAB*, such as 'scene', 'light', 'configuration', and 'sensor', signify functionalities fundamental to any smart home platform. These components represent core elements necessary for device management, configuration settings, and environmental sensing. 'Events', 'control state', and 'status notification' highlight the focus on real-time updates and event-driven actions, ensuring users stay informed about their smart home ecosystem's status (see Figure 4 top). These components signify a common commitment between *HomeAssistant* and *OpenHAB* to ensure the reliability, functionality, and interoperability of essential features within smart homes.

**Finding 2 ( $\mathcal{F}_2$ )** – *HomeAssistant* developers concentrated on enhancing user-facing elements and interactions, whereas *OpenHAB* developers prioritized rule-based automation. However, both platform developers focused on the core functionalities of smart home platforms.

**RQ<sub>1</sub>**  $\mathcal{F}_1$  and  $\mathcal{F}_2$  Demonstrate that *OpenHAB* has a poor test ratio and *HomeAssistant* has 65% *add-ons* below the test ratio threshold. The most common tested components are devices, network, and sensor parameters for *OpenHAB* and control state, device, and configuration for *HomeAssistant*. *HomeAssistant* exclusively tests components such as switch, button, and cover, while *OpenHAB* reports rule-based, status, and ecosystem group tests.

To answer **RQ<sub>2</sub>** we provided 12 common test types commonly employed in regular software testing scenarios, such as web solutions, services, and applications (Tab. II). We identify several of these testing types in our analyzed platforms. The simplest and most prevalent test type is the unit test. While *HomeAssistant* and *OpenHAB* report usability tests, we note that the complete implementation and results are not available in the repositories. Brands and communities potentially could employ an issue tracker tool independent of the GitHub repositories. As a result, we categorize usability and regression testing as "Maybe", indicating a potential implementation. Interestingly, we observe automation pipelines for *HomeAssistant*, encompassing platform deployment, and

some device configuration and testing. However, similar scripts or pipelines are not evident for *OpenHAB*. Table II also maps the tests to participants' preferences, which we describe in Section V.

TABLE II: Test types and the observed type of test in repositories. Challenges reported on each type of test in the survey

Test process Test Type	Platforms RQ <sub>2</sub>		Purpose using testing techniques		
	Home Assistant	Openhab	Automated	Manual	Testing IoT
Unit Testing	✓	✓	ND	ND	✓
Integration Testing	✓	✓	ND	ND	✓
Functional Testing	✓	✓	✓	✓	✓
End-to-End Testing	✓	✓	ND	✓	✓
Performance Testing	✓	✓	✓	ND	✓
Security Testing	✓	✓	ND	ND	✓
Usability Testing	Maybe	Maybe	✓	✓	✓
Regression Testing	Maybe	Maybe	✓	✓	✓
Accessibility Testing	✓	✓	✓	✓	✓
Compatibility Testing	✓	✓	✓	✓	✓
Database Testing	✓	✓	ND	ND	✓
API Testing	✓	✓	✓	ND	✓
CI/CD Testing	✓	✓	✓	✓	✓
Exploratory Testing	✓	✓	✓	✓	✓
User Acceptance	✓	✓	✓	✓	✓

**RQ<sub>2</sub>** Tests for API services and device-cloud communication insurance. Some scripts test databases and parameter configurations for multiple devices. Continuous Integration is also considered in the process.

## V. IOT DEVELOPERS' PERSPECTIVES ON TESTING

The analysis of platform code in Section IV develops a data-driven characterization of the extent of testing in smart home platforms. However, this characterization must also be complemented with an understanding of *why* the state of testing is as it is. To this end, we present a survey-based study of developer perspectives on IoT testing, with the goal of understanding the key pain points experienced by developers, their priorities, and preferences that may affect how testing is carried out. This section describes the design of our survey, our coding and thematic analysis approach, and the key findings.

TABLE III: Open-ended questions from survey

ID	Question
Q01	What are the testing techniques that you employ and why?
Q02	What is your process for designing test cases for different types of products?
Q03	What tools/APIs/frameworks do you use to support the testing of IoT products?
Q04	How do you evaluate the effectiveness of your test cases/suites?
Q05	How do you typically resolve flaky tests?
Q06	What are the main challenges that you have encountered when testing IoT products?
Q07	What would you improve in the current software testing process for IoT products?
Q08	What would you improve in the current software testing or debugging tools?
Q09	Have you ever faced any specific debugging-related challenges for IoT products?

### A. Survey Design

Our survey consists of several questions on current testing practices and preferences, challenges the developers face, and the improvements they envision and organized as follows:

- **Demographic Information:** basic demographic information such as age, gender, and education.
- **Background Information:** includes employment status, years of general programming experience, IoT-related programming experience, and testing experience. We also asked how the participants learned about software testing.

- **Testing Practices and Preferences:** type of IoT product participants work on and the type of documentation they use for specifying IoT-related requirements for those products. We then asked about their preferred testing approaches and why they follow them. Furthermore, we asked them to provide information about their process of designing test cases and the tools they use to support the process. Next, we asked them about their evaluation process of those test cases. We also asked them if they had encountered any flaky tests during their testing process and how they resolved them. Finally, we asked whether they created test cases for reported bugs and the origins of those bugs.

- **Challenges and Expectations:** After obtaining information on IoT developers' current testing practices, we asked them about the challenges they encountered when testing and debugging the products. Furthermore, we asked them what improvements they desired in the current testing and debugging process and tools.

TABLE IV: Demographic information over n=80 participants

	Age				Education					
	18-29	30-39	40-49	50-64	High School	Collage	Vocational	Bachelor	Master	Doctorate
n	6	48	24	2	3	12	14	45	5	1
%	7.5	60	30	2.5	3.75	15	17.5	56.25	6.25	1.25

	Gender		Programming Experience				Testing Experience			
	Male	Female	0-3	3-5	5-10	10+	0-3	3-5	5-10	10+
n	72	8	5	7	21	47	19	20	39	2
%	90	10	6.25	8.75	26.25	58.75	23.75	25	48.75	2.5

1) *Participant Recruitment:* We recruited participants from multiple IoT platform developers' community forums (e.g., OpenHab community, Home Assistant community, SmartThings community, and Google Nest community) by posting a flier. We received a total of 186 responses, of which we discarded 106 responses due to (i) failed attention check questions, (ii) not finishing the survey, (iii) ambiguous responses, or (iv) duplicate responses. Finally, we obtained 80 valid responses (denoted as **P1-P80**), which we analyzed and presented our findings in Section V-C. Our survey took an average of 15 minutes to complete, and we offered a 10 USD Amazon Gift Card to each participant. Table IV provides demographic information about all of 80 participants. Most participants were male (90%), all were at least 18 years old, and most were between 30 to 49 years of age (90%).

2) *Ethical Consideration:* The study protocol was approved by our Institutional Review Board (IRB). Participants were informed about the study's goal before participating, and they willingly provided their consent to participate in the study and to disclose anonymized survey responses and quotes.

### B. Coding and Analysis

We used descriptive statistical analysis to present the quantitative results. To analyze nine free-text questions, we used thematic analysis with an inductive coding approach [20]. Two out of three authors randomly selected a question and coded the data independently. After completing the coding, authors met and discussed any disparity in their codes and finalized the code after reaching a consensus. After all the responses were coded, all three authors discussed to extract themes or patterns in the answers.

### C. Results from the Analysis of Survey Responses

In this section, we analyze the responses provided by the participants and answer **RQ<sub>3</sub>**, **RQ<sub>4</sub>**, and **RQ<sub>5</sub>**.

1) *Test design and evaluation:* OQ<sub>1</sub> and OQ<sub>3</sub> responses were used to answer **RQ<sub>3</sub>**. To design appropriate tests for their apps, developers start by learning the test requirements. Then they create a test plan by determining test goals, which may involve identifying corner cases, and in the process, building test scenarios. As **P63** states, "Once I have a clear understanding of the product, I create a test plan that outlines the scope of testing, including the devices, protocols, and communication methods involved in the IoT system." Some developers emphasized performance testing to assess speed and scalability. Security testing is also deemed crucial to developers, especially, as **P16** states, "...for products dealing with sensitive information."

Test effectiveness is evaluated by assessing its maintainability and execution time. Coverage-based evaluation is another popular assessment in which developers consider requirements and use-case coverage as well as code coverage and mutation analysis. Compliance with organizational standards is also considered to measure test case effectiveness.

**RQ<sub>3</sub>** Developers emphasize creating a comprehensive test plan after defining the product and testing scope, focusing on devices, protocols, and communication. Performance and security testing are crucial for sensitive data, with evaluation based on maintainability, execution time, coverage, and compliance.

2) *Current testing practices and preferences:* Table II from Section IV not only maps common testing techniques to our observations (of the presence/absence of the techniques) in our analysis of OpenHAB and HomeAssistant, but also maps them to participants' responses regarding why they use (or don't use) the specified techniques (e.g., for manual analysis, automated analysis, or IoT testing in general). We observe that developers lean towards one technique over the other based on the nature of the test. For instance, developers opt for manual testing to ensure requirement compliance, evaluate interfaces, explore corner cases, and validate solutions. Conversely, they emphasize that automated testing is beneficial for evaluating performance, ensuring regression tests, and implementing continuous integration. For certain test types, we could not find any related answers; thus, we designated them as "Not Defined" (ND), signifying a lack of evidence regarding the use of automated or manual techniques, particularly for security evaluation or database testing.

Participants mention a set of tools oriented to the test and quality assurance such as Cucumber and Kibana, In addition to those tools, testing IoT protocols and connectivity associated with scripting are essential for device and network testing. We observe that just a few of them are dedicated to monitoring and data visualization.

Our analysis of the comments from participants leads to an understanding of their preferences. Particularly, we found that almost all of our participants expressed a preference

for automated testing, followed by manual testing and semi-automated testing. The participants value the automated tests mainly because of their efficiency, speed, and coverage. The automated test enables cross-platform testing, continuous testing, and performance testing. There is a focus on identifying defects early in the development process to reduce costs and improve reliability, as **P46** states, “Automated testing helps catch bugs early in the development process, reducing the cost of fixing them later.”.

**Finding 3 ( $\mathcal{F}_3$ )** – Automated testing is preferred by most participants. There is a focus on identifying defects early in the development process to reduce costs and improve reliability. Consistency in results and repeatability are also highlighted, along with the importance of generating detailed logs and reports.

Some participants prefer manual testing to test new features or achieve flexibility to requirements changes. Moreover, developers emphasized the need to develop a deep understanding of the system, guided by human intuition, e.g., as **P4** states, “I want to see the nuts and bolts of how a thing works so I can better understand how to make it work the way I want.”. Participants also claimed that security issues are easily verified using manual testing, which is particularly interesting, given that a recently study Ami et al. [21] found that in general, developers prefer automated security tools for testing over manual analysis, given their rigor and ability to catch what developers miss. Finally, developers expressed that manual testing also helps evaluate user experience and accessibility.

**Finding 4 ( $\mathcal{F}_4$ )** – Some developers recommend manual testing for its ability to bring in human intuition, adaptability, and a nuanced understanding of user experiences which can help test real-world scenarios.

Some developers adopt a semi-automated approach to balance automation and manual testing, including human judgment and interactions for programs that cannot be automated. As **P64** explains: “I use semi-automated testing because it allows me to strike a balance between manual testing and automation. Some aspects of IoT devices and applications require human judgment and interaction that can’t be fully automated.”. Testers decide to use a hybrid between automated and manual testing mostly because of the flexibility in performing regression tests adding new features and testing real-world scenarios.

Finally, we find that developers resolve flaky tests mostly by updating and reviewing environment setups, analyzing logs, and optimizing test scripts. For resolving issues within the source codes, developers analyze test logs, review test scripts, and fix synchronization issues by adjusting wait times or implementing retry mechanisms.

**RQ<sub>4</sub>** Based on  $\mathcal{F}_3$  and  $\mathcal{F}_4$ , Developers favor automated testing for early defect detection, cost reduction, regression testing, and managing changes, supported by monitoring and visualization tools for timely failure identification. However, manual testing remains essential for special cases and log analysis.

3) *Current challenges and future research scope*: Our survey revealed several challenges developers face when testing IoT platforms and devices (**RQ<sub>5</sub>**). A major challenge expressed by several developers was about ensuring seamless connectivity and communication between devices and platforms as challenging, as there is no common platform. Particularly, developers expressed how testing compatibility across a vast array of devices supported by various platforms is a time-consuming task.

**Finding 5 ( $\mathcal{F}_5$ )** – A primary challenge encountered by developers in testing IoT apps is validating compatibility across various devices and platforms.

Similarly, ensuring the devices continue to work seamlessly with over-the-air updates, including both the software and firmware, is a major concern. There are many IoT devices, such as, motion sensors, which run continuously. Delivering an update to these devices poses a significant challenge due to the potential for operational disruptions. As **P17** states, “Firmware over-the-air (FOTA) updates can be problematic, especially when dealing with a large number of devices. Debugging issues related to the update process, like interrupted downloads or failed installations, requires thorough testing.”

**Finding 6 ( $\mathcal{F}_6$ )** – Developers face difficulties in debugging issues when a firmware update disrupts the functionality of an IoT product.

Moreover, performance testing is another challenge for devices that operate on low resources. As **P12** states, “Identifying why a device fails to wake up or operate as expected in low-power states can be a complex task.” Sensor data inaccuracy also affects the debugging process when network interference or extreme weather conditions introduce noise in the data.

**Finding 7 ( $\mathcal{F}_7$ )** – Developers also face challenges during testing or debugging in low-power mode.

Developers perceive that the current test infrastructure lacks equipment and tools to simulate real-world scenarios, including extreme weather conditions or network interruptions. Scalability to handle larger IoT deployments and conduct performance testing is another major concern. Compatibility testing is mentioned for ensuring support over a wider range of devices. Strengthening security tests is also important to identify vulnerabilities and potential weaknesses. Some think that better documentation can help facilitate knowledge sharing and collaboration.

Current tools can benefit greatly if they offer automated test reporting, visualization, and prioritization. As **P45** states,



“Debugging tools should offer more comprehensive support for analyzing and visualizing complex data structures and their changes during runtime.” Cross-platform testing needs improvement according to some developers. As **P50** states, “Improved cross-platform support in debugging tools would be valuable, allowing developers to debug code running on different operating systems seamlessly.” Real-time collaboration in testing is another talking point among the developers as it can allow multiple team members to debug and troubleshoot issues simultaneously.

**Finding 8 ( $\mathcal{F}_8$ )** – Performance, scalability, real-world scenarios, and real-time collaboration are the major concerns for developers facing IoT components debug and test.

The open-ended responses gathered from the survey help us unveil some crucial aspects of IoT app testing that require attention and consideration. Particularly, we found that only a few developers worry about lack of standards and documentation, relative to the issues highlighted in  $\mathcal{F}_8$ .

**Finding 9 ( $\mathcal{F}_9$ )** – Few developers are concerned about poor documentation and lack of organizational standards and procedures for testing IoT platforms.

Finally, the acknowledgment of compliance with organizational standards as a metric for measuring test case effectiveness underscores the importance of aligning testing procedures with established benchmarks. Likewise, the emphasis on security testing within intricate IoT environments resonates as a critical area, where vulnerabilities or misconfigurations might remain undetected until a security breach occurs.

**Finding 10 ( $\mathcal{F}_{10}$ )** – Security testing is a critical area where vulnerability detection becomes more difficult due to the diversity of devices and components and lack of IoT oriented testing tools.

**RQ<sub>5</sub>** IoT development faces challenges like communication platform limits, knowledge gaps, and unclear user requirements due to poor documentation. Key issues include testing diverse devices, addressing security, optimizing power, ensuring real-time responsiveness, and managing updates and firmware. Persistent problems involve standardization, third-party integration, and testing across networks.

## VI. DISCUSSION

While we managed to find answers to our five RQs, the overarching goal of our research is two-fold: to 1) learn about the general testing practices of IoT developers, and 2) Uncover the key gaps related to testing IoT platforms. Based on these goals, we present the following discussion topics.

### A. Primary Focus on Unit Testing

Our analysis of *OpenHAB* and *HomeAssistant* demonstrates that the primary focus is on unit testing. This empirical observation also resonates with findings from our user survey,

i.e., a majority of the participants solely rely on automated testing ( $\mathcal{F}_3$ ), and manual testing techniques such as user acceptance or exploratory testing are generally absent from practice, except in rare cases ( $\mathcal{F}_4$ ).

Particularly, User Acceptance Testing (UAT) offers evaluation of an application from the end-user perspective and validates the readiness of its deployment to the real-world environment [22]. During our survey, IoT developers did not emphasize on this type of testing. The reason for not employing UAT is because of its dependence on manual effort and its time consuming nature. Although recent works have focused on automating this process using large language models [22]. Exploratory testing can help in the continuous integration and delivery pipeline for a large-scale software system [23], which was one of the more common concerns among the participants. While there is recent work on performing exploratory testing using static analysis [24] or in a gamified way [25], additional research is necessary to evaluate the usability and efficiency of such techniques in the IoT context.

Similarly, we observe no integration testing in our analysis of *OpenHAB* and *HomeAssistant*. One explanation for this observation could be the sheer difficulty of testing across various diverse platforms and devices, as perceived by developers in our study ( $\mathcal{F}_5$ ), particularly given concerns regarding the unpredictable impact of firmware/software updates ( $\mathcal{F}_6$ ).

### B. Compatibility Testing and Future Solutions

Several participants talk about the importance of compatibility testing in the context of IoT apps and integrations ( $\mathcal{F}_5$ ), which is challenging due to the general fragmentation in the smart home landscape, in terms of platforms, communication protocols, networking standards, operating systems, and types of devices and sensors. Ensuring compatibility among these diverse technologies is crucial for IoT platforms. The absence of robust compatibility testing across all technologies may lead to post-release issues, ranging from user inconvenience to severe security vulnerabilities. This is a timely and critical challenge, as developers invest substantial time in repetitive tests across varied technologies, dealing with debugging issues that arise post-deployment.

Given the general lack of integration testing and the concern among developers regarding compatibility and testing in the fragmented IoT space, the integration of simulation environments into the testing apparatus offers a promising direction for future research. Simulating real-world scenarios involving varied device types and communication protocols would allow developers to validate changes with increased robustness. Furthermore, advancements in IoT interoperability frameworks, automation in testing procedures, and the potential infusion of machine learning or AI-driven testing solutions hold promise in mitigating compatibility challenges.

### C. Performance Testing and Scalability Challenges

Managing scalability and performance in IoT software emerges as a complex task, particularly in large-scale deployments ( $\mathcal{F}_8$ ). As the IoT platforms expands, devices get

interconnected with a multitude of other devices and data points, which makes testing of these devices and apps more challenging. These challenges are crucial as they directly impact system reliability, efficiency, and user experience. This issue is inherently tied to IoT due to its vast array of user base, and continuous data exchange occurring among the devices.

Upgrading the testing infrastructure to accommodate scalability testing at various levels of deployment can be a potential research direction. Tools enabling continuous integration to identify and resolve performance bottlenecks in real-time can help in mitigating this issue. There are existing tools available that address performance and scalability testing in various domains, such as Apache JMeter [26], Gatling [27], Taurus [28]. However, their applicability in IoT scenarios may be limited due to challenges in simulating complex IoT environments and diverse communication protocols. These unique, IoT-specific challenges demand more specialized testing solutions.

#### D. Improvements in IoT Testing Compliance

The responses from IoT developers regarding organizational standards in testing ( $\mathcal{F}_9$ ) reveal a significant emphasis on adhering to company policies, industry standards, and regulatory requirements. Testing for compliance with industry-specific regulations, such as medical device or automotive safety standards, or privacy regulations such as GDPR [29] and CPRA [30] in the context of smart homes, adds complexity to the testing process. Addressing these challenges requires an approach involving robust compliance testing methodologies with continuous monitoring. The IoT industry can benefit greatly from tools that can analyze historical compliance data, regulatory changes, and industry-specific standards to detect potential areas of non-compliance.

### VII. THREATS TO VALIDITY

Threats to **construct validity** include concerns over the test ratio in automated analysis compared to test coverage, due to the complexity of components and environments. Traceability issues on some platforms obscure links between test and functional code. To address this, we combined quantitative analysis with manual inspection by two authors, providing insights into developers' testing intentions, supported by survey responses.

Threats to **internal validity** refer to the representativeness of the randomly selected tests at the labeling process and reported components. We sorted the *add-ons* and components to identify the largest number of focal methods and focal test methods. We calculate the z-score on our test file sampling and manual inspection to achieve the 95% confidence.

Another potential internal validity of our study lies in the subjectivity inherent in the manual analysis of test codes. To address this, we adopt a paired analysis approach, where analyses are conducted collaboratively. This helps us minimize individual biases. To avoid misdirection while individual coding, we conduct our analysis in multiple phases. We observed a substantial decrease in disagreement rates as we progressed through the latter phases of analysis.

### VIII. RELATED WORK

While studies have examined testing practices for industrial software [31], [32] and mobile apps [14], [33], [34], this research focuses on the challenges of software testing in IoT. It highlights persistent bugs despite significant time, resources, and testing efforts, as well as developers' practices and perceptions of test case design, automation, and quality metrics.

Research on IoT platforms has explored tools for bug detection [16], [35]–[38] and highlighted the complexity of testing such systems [15], [39]. Bures et al. [40] emphasize the need for specialized testing methods tailored to IoT. Surveys focus on bug detection, security failures [3], [6], [41]–[46], and future directions. Zhu et al. [47] predict trends toward intelligent, large-scale testing, emphasizing big data, cloud computing, and AI. Increased adoption of Smart Home solutions has led to security evaluations, such as Google's Nest and Philips Hue platforms, revealing key vulnerabilities and potential misuse [2], [12].

Previous research focuses individually on tools and methods, introducing challenges on specific topics like cloud computing or some security concerns, this study aims to acquire a deeper understanding of the testing methodologies currently employed by IoT developers. We seek to grasp the big picture about how developers are testing IoT platforms and the rationale behind their choices and collect insights regarding potential areas for enhancing future testing practices in IoT.

### IX. CONCLUSIONS

Our examination of the landscape of software testing within IoT platforms derived substantive insights, with 10 key findings from both a mining-based study and a survey with developers.

We took a closer look at two specific platforms – *HomeAssistant* and *OpenHAB*. We find notable evidence signaling the difficulty that developers face with testing IoT platforms, with the majority of *add-ons* and integration apps of both platforms falling short of the 50% test coverage threshold. On average, only 5% *add-ons* contains any test methods for *OpenHAB* and well-known brand like Amazon Alexa exhibits a maximum test ratio of 59%.

A majority of our survey participants prefer automated testing, to try to catch problems early in the development process to save time and make things more reliable. They stress the need for consistent, repeatable tests and detailed logs. But interestingly, some developers still prefer manual testing as it facilitates human intuition and adaptability for real-world situations.

Finally, our research identifies challenges in fixing problems caused by software updates, cross-platform testing, and dealing with issues in low-power devices – among others. In summary, our study sheds light on the testing practices, tools, perceptions, and challenges for IoT platforms, illustrating promising pathways for future research to improve testing for this rapidly growing domain.

## REFERENCES

- [1] Anonymous, "Number of internet of things (iot) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030." [Online]. Available: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>
- [2] K. Kaffle, K. Moran, S. Manandhar, A. Nadkarni, and D. Poshyvanyk, "A Study of Data Store-based Home Automation," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. Richardson Texas USA: ACM, Mar. 2019, pp. 73–84. [Online]. Available: <https://dl.acm.org/doi/10.1145/3292006.3300031>
- [3] S. Manandhar, K. Moran, K. Kaffle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Helion: Enabling a Natural Perspective of Home Automation," Jun. 2019, arXiv:1907.00124 [cs]. [Online]. Available: <http://arxiv.org/abs/1907.00124>
- [4] TechnoGym, "Technogym connected gym equipment." [Online]. Available: <https://www.technogym.com/en-US/business/>
- [5] T. Singh, A. Solanki, S. K. Sharma, A. Nayyar, and A. Paul, "A decade review on smart cities: Paradigms, challenges and opportunities," *IEEE Access*, vol. 10, pp. 68 319–68 364, 2022.
- [6] A. Makhshari and A. Mesbah, "IoT Bugs and Development Challenges," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 460–472, iSSN: 1558-1225. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9402092>
- [7] F. Corno, L. De Russis, and J. P. Sáenz, "On the challenges novice programmers experience in developing iot systems: A survey," *Journal of Systems and Software*, vol. 157, p. 110389, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121219301566>
- [8] F. Corno, L. De Russis, and J. P. Sáenz, "How is open source software development different in popular iot projects?" *IEEE Access*, vol. 8, pp. 28 337–28 348, 2020.
- [9] Anonymous, "openHAB." [Online]. Available: <https://www.openhab.org/>
- [10] H. Assistant, "Home Assistant." [Online]. Available: <https://www.home-assistant.io/>
- [11] M. Tufano, S. K. Deng, N. Sundaresan, and A. Svyatkovskiy, "Methods2Test: A dataset of focal methods mapped to test cases," in *Proceedings of the 19th International Conference on Mining Software Repositories*, May 2022, pp. 299–303, arXiv:2203.12776 [cs]. [Online]. Available: <http://arxiv.org/abs/2203.12776>
- [12] Anonymous, "Nest App." [Online]. Available: <https://nest.com/app/>
- [13] Daniel Rodriguez-Cardenas, "Iot testing study online appendix." [Online]. Available: <https://github.com/WM-SEMERU/iot-mining-helion>
- [14] M. Linares-Vásquez, G. Bavota, M. Tufano, K. Moran, M. Di Penta, C. Vendome, C. Bernal-Cárdenas, and D. Poshyvanyk, "Enabling Mutation Testing for Android Apps," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Aug. 2017, pp. 233–244, arXiv:1707.09038 [cs]. [Online]. Available: <http://arxiv.org/abs/1707.09038>
- [15] A. K. Gomez and S. Bajaj, "Challenges of testing complex internet of things (iot) devices and systems," in *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, 2019, pp. 1–4.
- [16] S. Bosmans, S. Mercelis, J. Denil, and P. Hellinckx, "Testing IoT systems using a hybrid simulation based testing approach," *Computing*, vol. 101, no. 7, pp. 857–872, Jul. 2019. [Online]. Available: <https://doi.org/10.1007/s00607-018-0650-5>
- [17] Anonymous, "Tree-sitter Introduction." [Online]. Available: <https://tree-sitter.github.io/tree-sitter/>
- [18] Amazon, "Amazon Alexa." [Online]. Available: <https://www.developer.amazon.com/en-US/alexa/>
- [19] Anonymous, "Homekit." [Online]. Available: <https://www.apple.com/home-app/>
- [20] V. Braun and V. Clarke, *Thematic Analysis: A Practical Guide*. SAGE Publications, 2021. [Online]. Available: <https://books.google.com/books?id=eMArEAAQBAJ>
- [21] A. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, "False negative - that one is going to kill you" - Understanding Industry Perspectives of Static Analysis based Security Testing," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 23–23.
- [22] Z. Wang, W. Wang, Z. Li, L. Wang, C. Yi, X. Xu, L. Cao, H. Su, S. Chen, and J. Zhou, "Xuat-copilot: Multi-agent collaborative system for automated user acceptance testing with large language model," vol. abs/2401.02705, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2401.02705>
- [23] T. Mårtensson, D. Ståhl, A. Martini, and J. Bosch, "Chapter 3 efficient and effective exploratory testing of large-scale software systems," in *Accelerating Digital Transformation: 10 Years of Software Center*. Springer, 2022, pp. 51–81.
- [24] J. Doyle, T. Laurent, and A. Ventresque, "Modelling android applications through static analysis and systematic exploratory testing," in *10th International Conference on Dependable Systems and Their Applications, DSA 2023, Tokyo, Japan, August 10-11, 2023*. IEEE, 2023, pp. 94–104. [Online]. Available: <https://doi.org/10.1109/DSA59317.2023.00022>
- [25] R. Coppola, T. Fulcini, L. Ardito, M. Torchiano, and E. Alégroth, "On effectiveness and efficiency of gamified exploratory GUI testing," in *TOSEM*, vol. 50, no. 2, 2024, pp. 322–337. [Online]. Available: <https://doi.org/10.1109/TSE.2023.3348036>
- [26] E. H. Halili, "Apache jmeter," 2008.
- [27] Gatling. (2023) gatling: Modern Load Testing as Code. [Online]. Available: <https://github.com/gatling/gatling>
- [28] Blazemeter. (2023) taurus: Automation-friendly framework for Continuous Testing. [Online]. Available: <https://github.com/Blazemeter/taurus>
- [29] European Union. (2023) General Data Protection Regulation (GDPR) – Official Legal Text. [Online]. Available: <https://gdpr-info.eu/>
- [30] C. S. Legislature, "California Privacy Rights Act of 2020 ("CPRA")," [https://leginfo.ca.gov/faces/codes\\_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5](https://leginfo.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5), 2020.
- [31] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Software testing: Survey of the industry practices," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1449–1454.
- [32] P. S. Kochhar, X. Xia, and D. Lo, "Practitioners' Views on Good Software Testing Practices," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Montreal, QC, Canada: IEEE, May 2019, pp. 61–70. [Online]. Available: <https://ieeexplore.ieee.org/document/8804445/>
- [33] M. Linares-Vásquez, C. Bernal-Cardenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 613–622.
- [34] X. Wang, Y. Sun, S. Nanda, and X. Wang, "Looking from the mirror: Evaluating iot device security through mobile companion apps," in *USENIX Security Symposium*, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:199523951>
- [35] G. Fortino, C. Savaglio, G. Spezzano, and M. Zhou, "Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 223–236, 2021.
- [36] N. Medhat, S. Moussa, N. Badr, and M. F. Tolba, "Testing techniques in iot-based systems," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, 2019, pp. 394–401.
- [37] L. Zhang, W. He, O. Morkved, V. Zhao, M. L. Littman, S. Lu, and B. Ur, "Trace2TAP: Synthesizing Trigger-Action Programs from Traces of Behavior," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 3, pp. 1–26, Sep. 2020. [Online]. Available: <https://dl.acm.org/doi/10.1145/3411838>
- [38] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, "How Users Interpret Bugs in Trigger-Action Programming," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland UK: ACM, May 2019, pp. 1–12. [Online]. Available: <https://dl.acm.org/doi/10.1145/3290605.3300782>
- [39] G. Reggio, M. Leotta, M. Cerioli, R. Spalazzese, and F. Alkhabbas, "What are iot systems for real? an experts' survey on software engineering aspects," *Internet of Things*, vol. 12, p. 100313, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S254266052030144X>
- [40] M. Bures, M. Klima, V. Rechtberger, X. Bellekens, C. Tachtatzis, R. Atkinson, and B. S. Ahmed, "Interoperability and integration testing methods for iot systems: A systematic mapping study," in *Software Engineering and Formal Methods*, F. de Boer and A. Cerone, Eds. Cham: Springer International Publishing, 2020, pp. 93–112.
- [41] A. Makhshari and A. Mesbah, "Iot bugs and development challenges," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 460–472.
- [42] U. Inayat, M. F. Zia, S. Mahmood, H. M. Khalid, and M. Benbouzid, "Learning-based methods for cyber attacks detection in iot systems: A survey on methods, analysis, and future prospects," *Electronics*, vol. 11,

no. 9, 2022. [Online]. Available: <https://www.mdpi.com/2079-9292/11/9/1502>

- [43] X. Feng, X. Zhu, Q.-L. Han, W. Zhou, S. Wen, and Y. Xiang, "Detecting vulnerability on iot device firmware: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 1, pp. 25–41, 2023.
- [44] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, "Understanding IoT Security from a Market-Scale Perspective," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. Los Angeles CA USA: ACM, Nov. 2022, pp. 1615–1629. [Online]. Available: <https://dl.acm.org/doi/10.1145/3548606.3560640>
- [45] T. A. Ahanger, A. Aljumah, and M. Atiquzzaman, "State-of-the-art survey of artificial intelligent techniques for iot security," *Computer Networks*, vol. 206, p. 108771, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S138912862200007X>
- [46] A. S. Ami, N. Cooper, K. Kafle, K. Moran, D. Poshyvanyk, and A. Nadkarni, "Why Crypto-detectors Fail: A Systematic Evaluation of Cryptographic Misuse Detection Techniques," in *2022 IEEE Symposium on Security and Privacy (SP)*, May 2022, pp. 614–631, arXiv:2107.07065 [cs]. [Online]. Available: <http://arxiv.org/abs/2107.07065>
- [47] S. Zhu, S. Yang, X. Gou, Y. Xu, T. Zhang, and Y. Wan, "Survey of Testing Methods and Testbed Development Concerning Internet of Things," *Wireless Personal Communications*, vol. 123, no. 1, pp. 165–194, Mar. 2022. [Online]. Available: <https://doi.org/10.1007/s11277-021-09124-5>