Functional End-to-End Testing of IoT Systems

Jean Baptiste Minani

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Software Engineering) at

Concordia University

Montréal, Québec, Canada

July 2025

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Jean Baptiste Minani**

Entitled: **Functional End-to-End Testing of IoT Systems**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Emre Erkmen

_____ External Examiner
Dr. Lionel C. Briand

_____ Arms-Length Examiner
Dr. Jamal Bentahar

_____ Examiner
Dr. Todd Eavis

_____ Examiner
Dr. Joey Paquet

_____ Supervisor
Dr. Yann-Gaël Guéhéneuc

_____ Co-supervisors
Dr. Naouel Moha, Dr. Fatima Sabir

Approved by _____
Dr. GPD, Chair of Department of Computer Science and Software Engineering

_____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Functional End-to-End Testing of IoT Systems

Jean Baptiste Minani, Ph.D.

Concordia University, 2025

IoT systems are increasingly deployed across various domains, making end-to-end (E2E) testing crucial to ensuring expected functionality. However, testing IoT systems is challenging due to their heterogeneity, distributed execution, and real-world constraints. Traditional testing approaches are inefficient and limited, focusing on isolated layers rather than complete system interactions. This thesis presents findings from a Systematic Literature Review (SLR) and an Industry Study on IoT system testing, analyzing existing challenges, approaches, and tools. Based on these insights, we propose a taxonomy for testing IoT systems and introduce a framework for evaluating the technical software engineering (SE) testable aspects of IoT systems. Building on this foundation, we propose an approach for functional end-to-end (E2E) testing of IoT systems, leveraging Use Case Specifications (UCSs) written in a restricted format and IoT systems descriptions. Our approach systematically converts UCSs into executable test scenarios, which are further transformed into structured test data (i.e., payload). This payload provides the necessary data for generating test cases that cover multiple layers of the IoT system. The generated test cases are then executed on the system under test (SUT) to detect bugs. To evaluate the proposed approach, we conducted an empirical study on an IoT system, analyzing its effectiveness in test case generation and bug detection. The results demonstrate that our approach detects bugs across multiple layers of the IoT system by leveraging the use of real-time execution

data. Furthermore, it significantly improves test coverage and efficiency, reducing manual effort while maintaining accuracy. These findings indicate that the use of real-time execution data at each layer enhances bug detection in IoT systems.

Dedication

I dedicate this work to God Almighty, whose boundless grace, wisdom, and strength have sustained me throughout this journey. His guidance has been my source of hope and perseverance in times of challenge and triumph. Without His blessings, this achievement would not have been possible. I also dedicate this dissertation to the loving memory of my late parents, whose sacrifices and unwavering love shaped my aspirations. My father, who passed away while I was preparing to enter high school, always believed in my potential and dreamed of seeing me succeed. My mother, who passed away three years later, remained a pillar of strength and encouragement. Although they are no longer here, their love, values, and dreams continue to guide me every day. I extend this dedication to my brothers and sisters, whose sacrifices and support made my education possible. Her unwavering commitment to my success has been a cornerstone of my journey. This work is also dedicated to my beloved wife and children, whose patience, encouragement, and love have given me the strength to persevere. Their support has been my motivation through every challenge and milestone. I am deeply grateful to all the teachers and professors who have shaped my academic path, from my earliest days in school to the completion of this PhD. Their knowledge, mentorship, and dedication have played a crucial role in my growth and achievements. Finally, I dedicate this dissertation to my late friend and brother, Gahunzire Josbert, who passed away just two months ago. His daily encouragement and unwavering support sustained me through some of the most difficult moments of this PhD journey. His memory will always be cherished, and his words will continue to inspire me.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Prof. Yann and Prof. Naouel, for their unwavering guidance, invaluable support, and for securing the necessary resources to make this research possible. Their expertise, patience, and mentorship have been instrumental in shaping this work. I would also like to extend my sincere appreciation to Prof. Fatima for her support and encouragement throughout this journey. I am profoundly grateful to my PhD committee members for their insightful advice and constructive feedback throughout the entire period of my doctoral studies. Their detailed comments and suggestions during each presentation were invaluable and significantly contributed to the success of this thesis. A special thanks to my co-authors and industry partners, particularly Kuradusenge Martin from CEIoT and Tomoaki of NTT, for their outstanding collaboration and contributions. Their expertise and industry perspectives provided crucial insights that strengthened the practical aspects of this research. I extend my heartfelt appreciation to my colleagues at the Ptidej Lab. Their support, thought-provoking discussions, and encouragement created an inspiring research environment that kept me motivated. My deepest gratitude goes to my spouse and children, whose unwavering support and prayers provided me with the strength to persevere through the challenges of this journey. Their patience and sacrifices have been fundamental to my success. I also extend my thanks to my friends for their continuous love, encouragement, and belief in me throughout this endeavor. Finally, I am grateful to everyone who, in one way or another, contributed to the completion of this thesis. Your support has been invaluable, and I deeply appreciate it.

List of Acronyms

Below is the list of acronyms that we used in this thesis.

- **AAA** - Arrange-Act-Assert
- **ACEIoT** - African Center of Excellence in Internet of Things
- **AI** - Artificial Intelligence
- **AOP** - Aspect Oriented Programming
- **API** - Application Programming Interface
- **AST** - Abstract Syntax Tree
- **B2B** - Back-to-Back Testing
- **CA** - Custom Approach
- **CMU** - Carnegie Mellon University
- **CoAP** - Constrained Application Protocol
- **CYPS** - Crop Yield Prediction System
- **DE** - Development Environment
- **DLE** - Device Life Expectancy
- **E2E** - End-to-End
- **FR** - Functional Requirement
- **GA** - Genetic Algorithm
- **GT** - Ground Truth
- **HA** - Hybrid Approach
- **HITL** - Human In The Loop

- **IoT** - Internet of Thing
- **JSON** - JavaScript Object Notation
- **LLM** - Large Language Model
- **MBT** - Model-Based Testing
- **MQTT** - Message Queuing Telemetry Transport
- **MSLA** - Multi-Stage LLM Approach
- **NFR** - Non-Functional Requirement
- **NLP** - Natural Language Processing
- **NTT** - Nippon Telegraph and Telephone Corporation
- **OTA** - Over-The-Air
- **OUT** - Object Under Test
- **PE** - Production Environment
- **PIT** - Mutation Testing System
- **PS** - Primary Study
- **QA** - Quality Attributes
- **RQ** - Research Question
- **RUCM** - Restricted Use Case Modeling
- **SDK** - Software Development Kit
- **SE** - Software Engineering
- **SE** - Staging Environment
- **SLR** - Systematic Literature Review
- **SQL** - Structured Query Language
- **SSLA** - Single-Stage LLM Approach
- **SUT** - System Under Test
- **TC** - Test Case
- **TE** - Testing Environment

- **TS** - Test Scenario
- **UCS** - Use Case Specification
- **UCTM** - Use Case Test Model
- **UI** - User Interface
- **UML** - Unified Modeling Language
- **UR** - University of Rwanda
- **UX** - User Experience
- **V&V** - Verification and Validation
- **VIT** - Vellore Institute of Technology
- **WIMP** - Where Is My Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

The Internet of Things (IoT) continues to evolve, with many systems being deployed daily across various domains. These systems consist of networked devices that collect and transmit data, enabling a wide range of services for end-users [101]. Unlike traditional software systems, which primarily operate within controlled environments, IoT systems are inherently distributed and dynamic, integrating multiple layers that must work seamlessly to support real-time data collection, processing, and user interaction. Typically, IoT systems are structured into device, gateway, cloud, and application layers [26, 6, 159, 180], each responsible for specific tasks such as sensing, communication, computation, and service delivery. Figure 1.1 illustrates the key components of an IoT system. IoT systems consist of a diverse range of devices, from simple sensors to complex processors, all interconnected via the Internet to exchange data and enable various services [142, 150, 105]. These devices rely on gateways to facilitate communication with cloud platforms, where data is aggregated, processed, and analyzed before being made accessible to applications. Many studies define an IoT system as a network of connected devices, often emphasizing the device and network layers while overlooking other essential components such as gateway

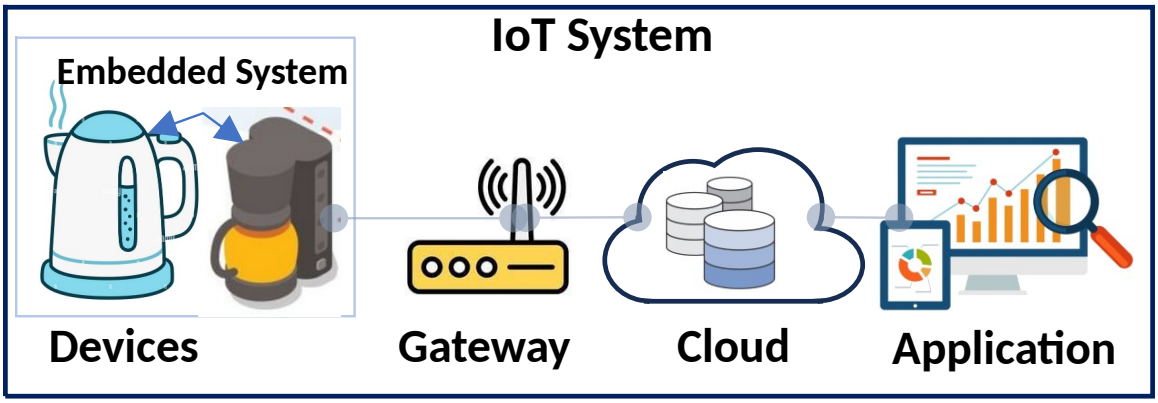


Figure 1.1: Key Components of an IoT System

and cloud layers. To eliminate potential ambiguity, in this thesis, the term IoT system specifically refers to what is commonly known as an IoT application, a system designed to manage, process, and use data collected from IoT devices [142]. Testing of IoT systems is essential to provide reliable service to the end users [66]. Lack of proper testing may lead companies to release defective systems, risking financial losses and damaging user trust due to associated risks with faulty systems [192]. IoT testing encompasses different levels, including unit testing of individual components, integration testing of interactions between components, system testing to verify overall functionality, and acceptance testing by end-users to ensure compliance with system requirements [128, 53, 189, 107]. However, unlike traditional software systems, IoT systems introduce additional challenges due to their heterogeneity, distributed execution, and real-time operational constraints [128]. As shown in Figure 1.1, each component plays a critical role in system operation, contributing to its overall complexity. The diversity of hardware components and communication protocols adds layers of difficulty to the testing process. To ensure correctness and reliability, IoT testing must adopt an end-to-end (E2E) approach, covering all layers from the physical device to the application layer [64, 68]. Failures in IoT systems can have severe consequences, including financial losses and threats to human safety. For instance, failures in aircraft sensor systems can result in catastrophic accidents, while malfunctions in medical

IoT devices can compromise patient well-being [11]. Despite the increasing adoption of IoT systems, testing of these systems remains an underexplored area, with only a limited number of studies focusing on IoT system testing [25, 163, 105, 140, 104, 74]. This thesis specifically focuses on functional end-to-end (E2E) testing of IoT systems, which ensures that data flows correctly from devices to gateways, cloud, and applications, and that the expected outcome is achieved.

1.2 Motivation

The rapid growth of the Internet of Things (IoT) has transformed the digital landscape, with approximately 18.8 billion connected IoT devices in use today and projections exceeding 40 billion by 2030. IoT now accounts for 75% of all active devices, surpassing traditional devices in both number and functionality. This explosive growth is also reflected in market value: the global IoT market is valued at over $1.1 trillion, with expectations to reach $1.56 trillion by 2029. Such expansion underscores the ubiquity and economic significance of IoT systems across domains, including smart transportation, agriculture, homes, energy, cities, and healthcare. Figure 1.2 illustrates the increasing percentage of IoT devices from 2010 to 2025 [1], emphasizing the shift toward connected systems. As IoT adoption accelerates, ensuring the reliability and correctness of these systems becomes critical.

Despite their widespread use, testing IoT systems remains a significant challenge. Functional end-to-end (E2E) testing is particularly crucial, as it verifies that data flows and interactions across the device, gateway, cloud, and application layers function as expected. However, conventional testing methods are often insufficient due to the heterogeneity of devices, distributed architectures, limited access to source code, and dynamic runtime environments. The increasing reliance on IoT systems in safety-critical and service-oriented applications makes thorough testing not just desirable but essential. Failures in smart healthcare, energy, or transportation systems can lead to severe real-world consequences. These

[1] https://explodingtopics.com/blog/iot-stats

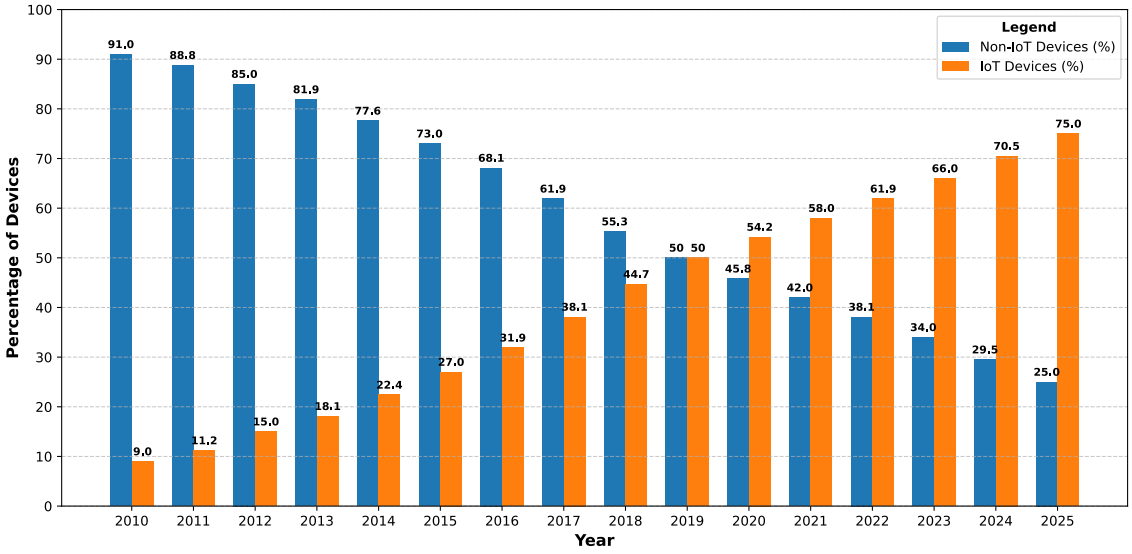


Figure 1.2: Non-IoT Devices vs IoT Devices (%) by Year (2010–2025)

challenges motivate the need for novel, automated, and effective functional E2E testing approaches tailored to the unique structure and complexity of IoT systems. Some existing studies propose testing approaches based on source code availability, while others use UML behavioral models such as activity diagrams, state charts, and sequence diagrams to generate test cases [191, 57]. However, many IoT systems rely on vendor-provided APIs, making source code inaccessible to testers. Similarly, UML behavioral models often lack sufficient detail to support automated testing effectively [191]. As a result, existing solutions remain insufficient for handling the full complexity of IoT systems. Despite ongoing research, practitioners continue to face significant challenges in conducting functional E2E testing for IoT systems. The combination of heterogeneous devices, dynamic execution environments, and the need to test across multiple layers requires new testing approaches beyond conventional approaches. Ultimately, we define our problem statement:

Problem Statement

Functional E2E testing of IoT systems remains challenging due to device heterogeneity, dynamic execution environments, and the need to test multiple layers. Traditional testing approaches often fail to systematically detect and localize bugs in real-world scenarios, necessitating a new approach for test generation and execution specific to IoT systems. Given these complexities, is it possible to automate functional end-to-end test generation and systematically execute the generated tests to detect bugs in IoT systems?

1.3 Thesis Statement and Research Questions

Thesis Statement

This thesis investigates the state of the art in IoT system testing from both academia and industry and proposes a taxonomy to guide practitioners in IoT system testing. Furthermore, it presents a framework addressing key software engineering (SE) aspects that require testing and introduces a structured approach for generating and executing functional E2E tests. This approach considers the complexity of IoT systems, which are characterized by heterogeneous components, dynamic interactions, and real-world constraints that often cause traditional testing approaches to fall short in detecting system-level bugs. We first conduct an SLR and an Industry Study to analyze existing IoT testing challenges, approaches, and tools. Based on these insights, we develop an IoT testing taxonomy, categorizing key aspects such as approaches, tools, metrics, artifacts, and testing stage and environments. This taxonomy serves as a structured reference for IoT system testing and is validated through an industry-wide survey. Furthermore, we introduce a framework for evaluating the technical SE testable aspects of IoT systems. Building on these foundations, we propose an approach for functional E2E testing, leveraging UCSs in a restricted format to systematically generate test scenarios. These scenarios are transformed into structured payloads, which serve as input for generating test cases covering multiple IoT system layers. The generated test cases are then executed on the System Under Test (SUT), capturing real-time execution data to detect bugs. To evaluate the effectiveness of the proposed approach, we conducted an empirical study on an IoT system, analyzing its performance in test case generation and bug detection. The results demonstrate that our approach improves bug detection across different layers of IoT systems while reducing the manual effort required to generate end-to-end tests, leveraging real-time execution data for enhanced bug detection across multiple layers.

This thesis answers the following research questions:

Research Questions (RQs)

- **RQ1:** How are IoT systems tested from a research perspective?
- **RQ2:** What challenges do researchers report in IoT system testing?
- **RQ3:** How do practitioners test IoT systems?
- **RQ4:** How do testers make decisions in IoT system testing?
- **RQ5:** What are the trends in IoT systems testing?
- **RQ6:** How can a taxonomy define aspects of IoT system testing?
- **RQ7:** How do practitioners evaluate the taxonomy?
- **RQ8:** How does the taxonomy improve test coverage?
- **RQ9:** What are testable technical SE aspects of IoT systems?
- **RQ10:** How complete are the testable technical SE aspects of IoT systems?
- **RQ11:** How practical is the framework for real IoT systems?
- **RQ12:** How do test generation approaches compare for E2E testing of IoT Systems?
- **RQ13:** How reliable are LLMs in E2E test generation for IoT systems?
- **RQ14:** How can functional E2E test cases be generated for IoT systems?
- **RQ15:** How can generated tests detect bugs in IoT systems?

1.4 Research Process

To answer the above RQs, we followed the research process illustrated in Figure 1.3.

This research was conducted in three phases: (1) problem identification, (2) taxonomy and framework development, and (3) approaches for IoT system testing. First, we identified key challenges in IoT system testing through two complementary studies: a systematic literature review of 83 studies published between 2012 and 2024 and an industry study involving 49 IoT practitioners. Additionally, we analyzed four years of Eclipse IoT surveys,

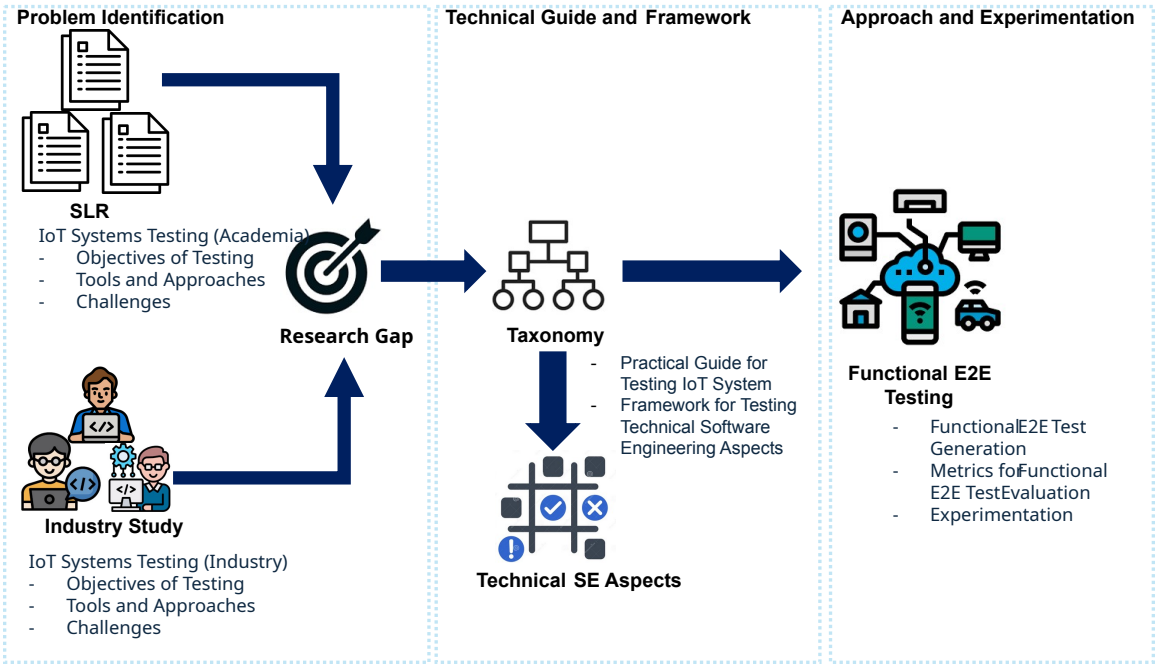


Figure 1.3: Research Roadmap

each involving at least 600 participants, to validate our findings. To address the identified challenges, we developed an IoT testing taxonomy, which was validated by over 200 practitioners. This taxonomy categorizes key aspects of IoT system testing, including objectives, tools, testers, stages, environments, objects under test, and approaches. Building on this, we proposed a framework for testing the technical software engineering aspects of IoT systems. The framework, developed in collaboration with NTT, was validated through a survey with 23 professionals and two case studies. Furthermore, we explored functional testing approaches tailored to IoT systems, focusing on functional end-to-end testing. We proposed an approach to generate test cases from use case specifications, transforming them into executable payloads. To enhance IoT-specific test generation, we introduced new keywords for writing use case specifications and developed a tool for automated test generation.

1.5 Contributions

The objective of this thesis is to propose an approach for end-to-end (E2E) functional testing of IoT systems using restricted Use Case Specifications. The key contributions of this thesis are:

- Investigating the current state of the art in IoT system testing from both industry and academia to identify existing gaps.
- Providing a practical guide for testing various aspects of IoT systems.
- Providing a framework for testing technical software engineering aspects of IoT systems.
- Proposing an approach for generating and executing tests that systematically cover all layers of the IoT system, from device to application, to detect bugs effectively.
- Defining specific metrics to assess test coverage and improve IoT system reliability.

The rest of this thesis is organized as follows. Chapter 2 provides background and definitions. Chapter 3 discusses related work. Chapter 4 presents the systematic literature review on IoT system testing. Chapter 5 discusses the industry study on IoT system testing practices. Chapter 6 introduces the taxonomy for testing IoT systems. Chapter 7 describes the framework for testing the technical software engineering aspects of IoT systems. Chapter 8 presents the end-to-end testing approach for IoT systems. Finally, Chapter 9 concludes the thesis with a summary of findings and future work.

Chapter 2

Background and Definitions

This chapter provides background on IoT systems and end-to-end testing in their context. It also includes a glossary of key terms used in this thesis.

2.1 Background

In this paper, we propose an end-to-end (E2E) test generation approach for IoT systems to test their expected functionality as outlined in the requirements. Furthermore, we identify IoT-specific metrics to assess the effectiveness of the generated tests.

2.1.1 IoT Systems

An IoT system is a network of devices designed to collect, process, and use data to provide useful services. These devices can range from small sensors, such as the DHT22 humidity sensor, to larger devices, such as smart home appliances or industrial machinery [101]. IoT systems generally consist of multiple layers, which can vary depending on the business application or use case [168, 154]. However, most IoT systems are structured around four key layers:

- Device layer: Includes sensors and actuators that collect data [144].
- Network layer: Transfers data between devices and the cloud or edge systems.

- Cloud layer: Stores and processes the collected data.
- Application layer: Uses the processed data to provide solutions, such as analytics or control mechanisms, to users.

Figure 1.1 shows an example of a typical IoT system. An IoT system collects data or sends commands to devices via the network or gateway layer. The data is then stored in the cloud or a backend server for further processing and analysis. The application layer uses this data to deliver various services to users. The complexity of IoT systems arises from the interactions between these layers, the diversity of devices, and the interoperability of communication protocols.

2.1.2 End-to-End Testing

From a traditional software engineering perspective, testing is the process of running software with the intent of finding software bugs (errors or other defects) [12]. It involves identifying issues, ensuring reliability, and validating software behavior under different conditions. Testing an IoT system goes beyond assessing the software; it requires checking each layer and the system as a whole [69] with specific objectives such as ensuring correct functionality, performance, or security. For IoT systems, testing is particularly challenging due to the complexity of interactions between different layers (i.e., devices, networks, cloud services, and applications). The heterogeneity of IoT devices, the distributed nature of IoT systems, and the use of diverse technologies and protocols further complicate the testing process. Figure 2.1 illustrates an example of the complexity of IoT systems.

Functional E2E testing in IoT systems focuses on verifying the entire system against its specifications [128]. It ensures that each component and interaction behaves as expected and that individual devices, protocols, and applications perform their intended tasks correctly.

Functional E2E testing checks the complete flow of data and interactions across all layers of the IoT system. The objectives include ensuring seamless communication (e.g.,

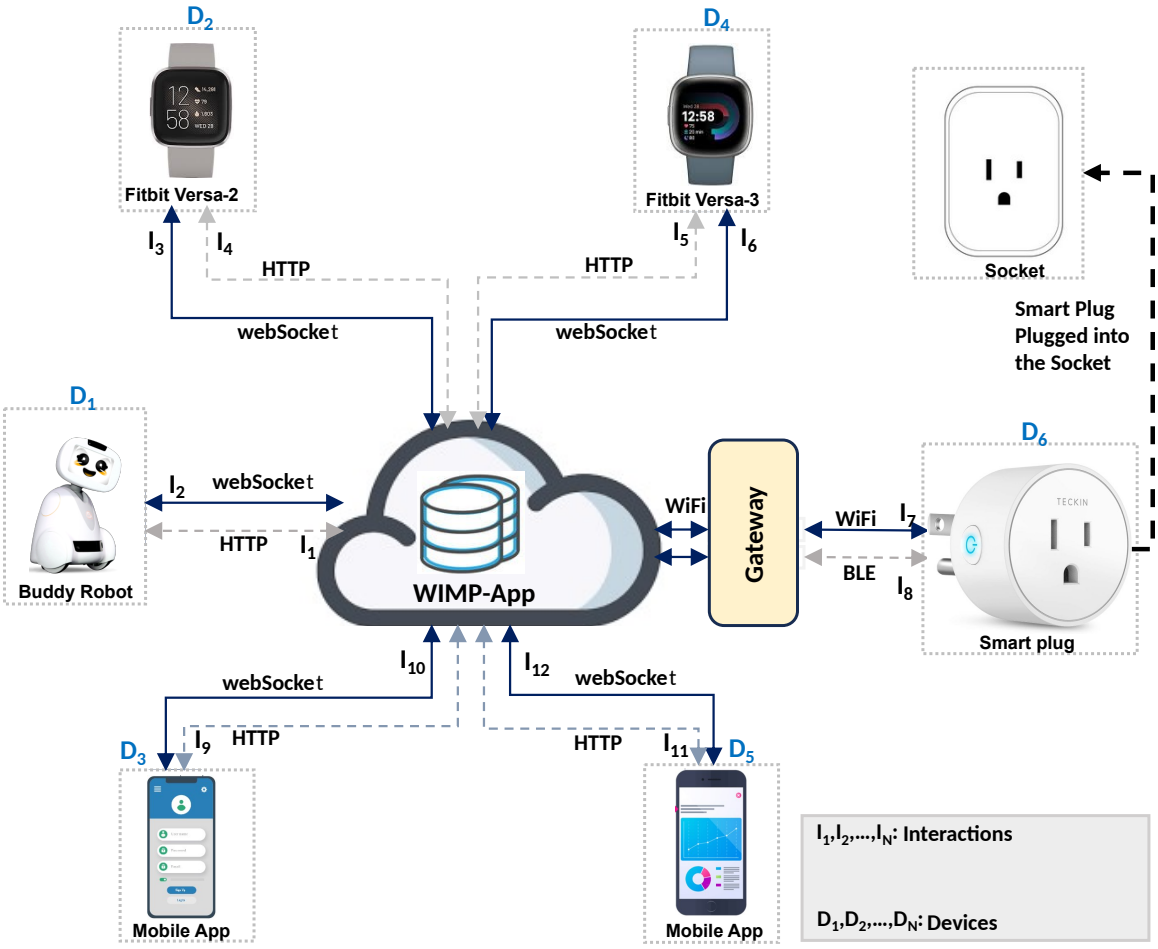


Figure 2.1: Interactions in IoT System

device → cloud → application → user interface), verifying functionality under real-world conditions, and identifying failures in data flow and processing. This process ensures that the entire system operates cohesively to meet user expectations and requirements.

2.1.3 Metrics for End-to-End Testing

Software metrics are quantifiable indicators used to assess, measure, and understand software quality and reliability [46]. These metrics enable testers to gather data on various testing procedures and devise strategies to enhance their efficiency. Metrics are a crucial component of any software development organization, helping to improve software quality [170]. Unlike traditional systems, IoT system testing requires specific metrics to address

its unique characteristics.

2.2 Definitions

This section defines key terms throughout the thesis to ensure clarity and consistency. These terms are fundamental to understanding the concepts presented in this thesis.

- **IoT System**. According to IEEE, an IoT system is a network of interconnected entities or objects that exchange information and interact with the physical world through sensing, data processing, and actuation [77].
- **Testing** is the activity of executing a system, subsystem, or component under specific preconditions with specific inputs so that its actual behavior can be compared with its required or expected behavior [56]. Testing involves the actual execution of the system under test (SUT) with the primary goal of finding defects so that they can be fixed.
- **System testing** is a phase in the software testing process to verify that a system under test functions as intended to meet specified requirements [169]. This activity involves checking the system to identify the presence of bugs and not to fix them [75].
- **A Defect**, informally known as a bug, is a flaw or weakness in the system or one of its components that could cause it to behave in an unintended, unwanted manner or to exhibit an unintended, unwanted property [56].
- **A Bug** is an error, flaw, or unintended behavior in a software system that causes it to produce incorrect or unexpected results, or to function in ways that deviate from its specified requirements. Bugs can arise due to mistakes in design, coding, configuration, or interactions with other components.
- **An error** is a human mistake that causes erroneous input or a defect [56].
- **A fault** is any abnormal system-internal condition, such as an incorrect stored data value, an incorrect subsystem state, or the execution of the wrong block of code, that may cause the system to fail [56].
- **A failure** is the event in which the system ceases to meet its requirements [56].

Chapter 3

Related Work

This chapter reviews related work in IoT systems testing, covering five key areas: Systematic Review of the Literature on IoT System Testing, IoT System Testing Taxonomy, Framework for Testing IoT Systems, and Functional End-to-End (E2E) Testing Approaches. We analyze existing studies, highlight gaps, and compare them with our research.

3.1 Systematic Literature Review on IoT System Testing

Several studies have surveyed IoT system testing, categorized into Testing Objectives, Approaches, Tools, and Challenges.

Only a few studies have addressed the quality aspects of IoT systems. In [88], the authors conducted a mapping study of quality attributes in IoT systems. Their study aimed to identify quality attributes discussed in the existing literature based on the ISO/IEC 25000 quality model [2]. They found that researchers primarily discussed performance, suitability, compatibility, usability, security, and maintainability in the context of IoT systems. Similarly, [7] provided a classification of IoT quality attributes, covering only performance, security, and privacy while suggesting the need for a more comprehensive study on all quality aspects. Another study [195] focused on the end-to-end quality of service (QoS) in IoT systems, primarily examining performance metrics and recommending additional

quality factors for assessing IoT systems. While prior research has examined various quality attributes, studies have approached them selectively rather than comprehensively. A holistic review consolidating all quality aspects of IoT systems remains necessary.

Several studies have explored testing approaches for IoT systems. A recent study [208] summarized state-of-the-art testing methods, identifying cloud-based and machine-learning-based approaches as dominant. The study categorized testing methods based on scope and objectives, covering techniques such as white-box, black-box, and gray-box testing. Other studies have examined specific areas, such as [87], which reviewed integration testing methods for IoT systems and concluded that more effective approaches are needed to address IoT-specific challenges. Model-based testing (MBT) has also been considered. In [36], the authors analyzed MBT for ensuring the quality of IoT systems, particularly in test case generation. Meanwhile, [47] described various IoT testing types and associated challenges, while [4] reviewed machine learning (ML)-based approaches for automating IoT test generation. Their study explored supervised and unsupervised learning algorithms for security attack detection, fault prediction, and anomaly detection, but it was limited to application-layer testing. From an industry perspective, [59] analyzed testing approaches for IoT healthcare solutions, proposing a framework to enhance availability. However, the study focused solely on healthcare and did not generalize findings for other IoT domains. While existing studies have provided valuable insights, there is a lack of a unified examination of IoT-specific testing approaches across different domains and layers.

Few studies have systematically examined testing tools for IoT systems. One study [208] reviewed the development of testbeds, identifying technical challenges such as concurrency and scalability, and suggesting intelligent testing with integrated technologies as a future direction. Other studies [41, 29] surveyed available IoT testing tools, focusing on testbeds, emulators, and simulators. These studies recommended further exploration of automation in testing tools and continuous integration for IoT testing. A comparative

study [18] evaluated test environments for IoT systems, emphasizing the need for test environments capable of handling geo-distributed, heterogeneous infrastructures such as edge/-fog computing. However, it excluded hardware-only testbeds, highlighting the need for comprehensive test environments that integrate both software simulation and IoT hardware modeling. Despite these efforts, a systematic review of IoT testing tools covering the breadth of software, hardware, and hybrid solutions is still lacking.

Challenges in IoT testing have been discussed in limited studies. [47] identified five primary challenges: lack of standardization, device heterogeneity, interoperability, security testing, and limitations in testing environments and tools. Similarly, [196] highlighted industry-specific challenges, such as platform diversity, software-hardware interconnection, real-time data velocity, security concerns, and scalability issues. While these studies provide useful insights into the challenges of IoT testing, they do not synthesize their findings into a broader review that links them with test objectives, approaches, and tools.

Previous studies have explored IoT testing objectives, approaches, tools, and challenges separately. However, despite the rapid evolution of the IoT domain, no existing review has investigated all four aspects collectively. This gap highlights the need for a comprehensive review of IoT systems testing from multiple perspectives.

Table 3.1 presents a comparison of recent reviews on IoT system testing, with our study listed in the last row.

3.2 Taxonomy for IoT System Testing

IoT system testing requires a structured classification to help practitioners systematically identify and understand the different aspects to be tested. A well-defined taxonomy for IoT testing provides a foundational reference, enabling practitioners to ensure comprehensive test coverage. This section introduces related work on IoT system testing taxonomies.

Several studies have proposed taxonomies for software testing to provide structured

Table 3.1: Related Work on IoT Systems Testing Review

Study	Year	Aim of Study	Research Method	Selected PSs	Focus on			
					TO	TA	TT	TC
[127]	2023	Overview of IoT testing from industry perspective	Survey and Interviews	N.A	✛	✛	✛	✚
[109]	2023	Security testing in IoT	Review	$\leq$ 2022	✛	✛	✛	✛
[208]	2022	Testing methods and testbeds in IoT	Survey	N.A	✘	✛	✛	✘
[47]	2022	Testing types and challenges in IoT	Survey	N.A	✘	✛	✘	✚
[59]	2022	Using ML to test IoT applications	Systematic Mapping	2016-2022	✘	✛	✘	✛
[14]	2022	Privacy and security, and blockchain in IoT	Review	N.M	✘	✛	✛	✘
[5]	2021	IoT device's availability testing	Survey	N.A	✘	✛	✛	✘
[18]	2021	IoT test environments	Survey	N.A	✘	✘	✛	✘
[45]	2021	Fuzzing techniques on IoT devices	Review	N.M	✘	✛	✛	✘
[178]	2021	Security testing in IoT	Review	2010-2019	✘	✛	✛	✘
[87]	2020	Methods and approaches for integration testing in IoT	Systematic Mapping	2009-2019	✘	✚	✘	✘
[88]	2020	MBT for IoT	Systematic Mapping	2009-2019	✛	✛	✛	✘
[196]	2020	IoT testing challenge from industry perspective	Survey	N.A	✘	✘	✘	✚
[36]	2019	Software testing techniques in IoT	Systematic Mapping	2008-2018	✘	✛	✘	✘
[148]	2019	Simulators, Emulators, and Testbeds for IoT	Review	N.M	✘	✘	✛	✘
[7]	2018	Methods for quality assurance in IoT	Systematic Mapping	2009-2017	✛	✛	✘	✘
[41]	2018	Testing types, tools (emulators and simulators), and challenges in IoT	Review	N.M	✘	✛	✛	✛
[29]	2018	Testbeds, Emulators, Simulators in IoT	Review	2012-2017	✘	✘	✛	✘
[131]	2018	Testing tools and techniques for IoT	Survey	N.A	✘	✛	✛	✘
[195]	2017	QoS approaches in IoT architecture	Systematic Mapping	2000-2016	✛	✘	✘	✘
[128]	2024	Testing challenges, objectives, approaches, and tools	Review	2012-2022	✚	✚	✚	✚

- **N.A**:Not Applicable; **N.M**:Not Mentioned.
- **TO**:Testing Objective; **TA**: Testing Approach; **TT**: Testing Tools; **TC**:Testing Challenge.
- ✚:Covered; ✛: Partially Covered; ✘: Not Covered.

classifications and guide testing teams. However, most of these taxonomies focus on traditional software systems and fail to address the complexities and unique challenges of IoT system testing. This section presents an overview of related taxonomies and highlights the gap that underscores the need for a dedicated IoT system testing taxonomy. This review specifically considers studies that explicitly propose taxonomies or classify aspects of IoT system testing, such as testing types, approaches, tools, or other relevant dimensions. General studies on IoT testing without a taxonomy focus were excluded. In this thesis, we use **testing aspects** to refer to any concept considered when testing a system, including the objectives of testing, testing environments, testing approaches, responsible entities, tools

and metrics, and the artifacts generated.

Villa et al. [187] discussed a taxonomy featuring nine overarching categories and 27 subcategories tailored for traditional software. This taxonomy was validated through a comprehensive survey involving IT managers and industry professionals, demonstrating its relevance and utility. However, it does not cover any aspects related to IoT system testing. Similarly, Vegas et al. [186] introduced a taxonomy for unit testing in conventional software systems, categorizing 13 testing techniques such as random testing, boundary value analysis, statement testing, branch testing, path testing, thread testing, and mutation testing. While useful for traditional software, this taxonomy lacks coverage of IoT-specific concerns such as testing environments, tools, and device interactions.

Unterkalmsteiner et al. [184] proposed a taxonomy for requirements engineering and software testing (REST) that focuses on aligning software requirements with testing strategies. The taxonomy was validated through an industry survey, highlighting its effectiveness in enhancing both requirement engineering and software testing processes. However, it is limited to a few aspects of IoT systems and does not fully support their diverse testing needs. Cheverda et al. [30] introduced a taxonomy for evaluating software quality based on metrics. This taxonomy addresses eight attributes fundamental to traditional software systems, including compatibility, portability, functional sustainability, security, usability, performance, maintainability, and reliability. However, it lacks IoT-specific testing considerations such as real-time constraints, energy efficiency, and multi-device interoperability. Khezemi et al. [89] proposed different quality attributes for IoT systems, emphasizing the need for IoT-specific quality metrics. Nonetheless, existing taxonomies do not sufficiently guide practitioners in determining what to test, how to conduct tests, where to test, and when testing should be performed.

Mubarakah et al. [130] introduced a taxonomy based on the Software Engineering Body of Knowledge (SWEBOK), emphasizing ten knowledge areas in software testing.

Although comprehensive, the study lacks details on test levels and techniques applicable to IoT systems. Makhshari et al. [114] focused on a taxonomy for categorizing IoT-related software bugs but did not address testing strategies for IoT systems. Similarly, Raibulet et al. [157] provided a taxonomy for software evaluation, attempting to categorize testing based on the aspects of "How" and "What." However, it did not consider critical IoT-related aspects such as testing stages, environments, and objectives.

Ladisa et al. [100] proposed a taxonomy for evaluating open-source software security. However, their work does not address any aspect of IoT system testing. Zander et al. [207] and Felderer et al. [52] focused on taxonomies for model-based testing, concentrating on traditional software but not considering the unique characteristics of IoT environments. Costa et al. [37], Roggio et al. [162], and studies such as [93] presented taxonomies on performance testing tools, software testing terminologies, and general testing techniques. While these taxonomies are relevant for software testing in general, their applicability to IoT is limited, as they do not account for IoT-specific testing needs such as hardware connectivity, network constraints, and dynamic environments. Coppola et al. [35] explored software metrics taxonomies, while Mubarakah et al. [130] classified software engineering tools and methodologies. However, these taxonomies do not fully address the challenges of testing IoT systems.

The ISO-29119 standard [173] provides guidelines and techniques for testing traditional software systems. However, testing IoT systems requires a broader approach that considers their distinctive attributes, such as real-time data exchange, device heterogeneity, and network variability. Firesmith [54] explored a taxonomy for software testing types. While comprehensive in its coverage of testing categories, it does not sufficiently address IoT-specific aspects such as scalability testing for IoT networks, interoperability testing across different device ecosystems, or protocol compatibility testing. Despite this limitation, the structure of Firesmith's taxonomy provides inspiration for our proposed taxonomy.

A taxonomy dedicated to IoT system testing is necessary to address these gaps. Yaqoob et al. [200] presented a comprehensive taxonomy of IoT architectures and devices, which complements our focus on testing by offering insights into the structural aspects of IoT systems. However, their work does not define a testing methodology, highlighting the need for a dedicated IoT testing taxonomy.

Finally, Usman et al. [185] explored various approaches to developing and validating taxonomies but did not propose a specific taxonomy for IoT system testing. Table 3.2 summarizes the related work on taxonomy for testing and compares these taxonomies based on the testing aspects covered. Despite prior efforts in taxonomy development for software

Table 3.2: Related Works on Testing Taxonomies in Software Engineering

Study	Aim of the Study	Year	Aspects of The Study							
			What	How	When	Where	Which	Who	Why	IoT
[100]	Taxonomy of attacks	2023	-	✝	-	-	-	-	-	No
[110]	MB Security Testing	2023	-	✝	-	-	-	-	-	Yes
[48]	IoT Testing Survey	2022	-	✝	-	-	-	-	-	Yes
[35]	Testing metrics	2022	†	-	-	-	-	-	-	No
[114]	Taxonomy of bugs	2021	-	-	-	-	-	-	-	Yes
[61]	Testing taxonomy for embedded systems	2015	†	†	-	-	†	-	-	No
[54]	Testing Types	2015	✚	✚	†	✚	✚	✓	✓	No
This	Testing IoT systems	2024	✓	✓	✓	✓	✓	✓	✓	Yes

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

testing, no existing taxonomy specific to IoT system testing exists, thus highlighting the need to establish a comprehensive IoT system testing taxonomy to guide practitioners.

3.3 Framework for IoT System Testing

Recent studies proposed several frameworks for testing various aspects of IoT systems. In [190], authors explain in detail the steps and key points of implementing automatic tests and developed various automatic client frameworks based on the Python language, connecting those frameworks to an automated testing environment. However, the proposed framework focuses solely on IoT devices rather than the entire system and does not address the technical software engineering aspects of IoT systems. Authors in [74] introduce

CT-IoT, a combinatorial testing path selection framework for IoT systems, designed to systematically identify and recommend effective testing paths. This study also proposes four coverage criteria to evaluate testing thoroughness and demonstrates CT-IoT's effectiveness through empirical studies on real-world IoT systems. However, this study did not systematically address the technical software engineering aspects of IoT systems. In [16], a framework for automated testing of IoT applications is presented, enabling user-defined experiments on agnostic IoT testbeds and virtual testbeds with adjustable network properties. While this approach addresses IoT application testing challenges, it does not specifically focus on technical software engineering aspects of IoT systems. In [22], authors focused on PatrIoT, a framework designed to support automated interoperability and integration testing of IoT systems by constructing scalable testbeds, ranging from physical to simulated environments, with predefined modules and example test cases. However, this work mainly defines principal components for the testing environment and does not delve into the technical software engineering aspects of IoT systems. In [153], Izinto, a pattern-based test automation framework for integration testing of IoT systems, is proposed. It offers generic and easily instantiable test patterns specific to IoT scenarios, validated through test cases in the Ambient Assisted Living domain. Nevertheless, the study does not focus exhaustively on testing the technical software engineering aspects of IoT systems. Authors in [13] presented AssureSense, a fault detection framework for IoT sensor data that leverages TsAssure, a novel feature extraction method, to detect subtle and hidden anomalies with improved accuracy. While effective in fault detection, its scope is confined to device-level issues and does not encompass the broader technical software engineering aspects of IoT systems. In another study [151], authors introduced RITA, an automated framework using a fine-tuned RoBERTa-based NER model to identify IoT critical objects, correlate threats, and recommend countermeasures to enhance IoT system resilience. However, this study does not consider the technical software engineering aspects of IoT systems. In [155], an

IoT Automation Framework is proposed, validating IoT systems using novel techniques such as power-based, ML-based image and sound validation, and OCR-based validation. While addressing end-to-end testing, the study does not specifically focus on the technical software engineering aspects of IoT systems. Authors in [86], authors presented a novel method for compliance testing and vulnerability evaluation of IoT system firmware, communication interfaces, and networking services using static and dynamic analysis, enabling the detection of security bugs across diverse platforms. Although this study focuses on compliance testing, it does not address the technical software engineering aspects of IoT systems. In another study [108], authors proposed IoTECS, a domain-specific language and lean simulation framework for efficient stress testing of IoT cloud systems, enabling scalable simulations of IoT and edge devices. Empirical results demonstrate IoTECS's superior performance, but the study does not investigate all technical software engineering aspects of IoT systems. In [133], a coverage criteria-based white-box testing framework for large-scale IoT applications is presented, focusing on event, functionality, and end-to-end flow coverage derived from architectural views. The framework demonstrates effectiveness in detecting subtle errors but does not address technical software engineering aspects of IoT systems. Yet, in another study [83], authors introduced a layered IoT framework emphasizing non-functional testing aspects such as security, scalability, reliability, and performance, enabling organizations to develop smart products. However, this study does not address technical software engineering aspects in detail.

Several studies also focus on the security aspects of IoT systems. In [9], authors introduced an automated penetration testing framework for smart home-based IoT devices, identifying common vulnerabilities and assessing them using Python-based tools. However, the focus remains on security testing rather than broader software engineering aspects. In another study [201], authors presentd FuzzDocs, a document-based black-box testing framework for IoT devices, which automates input generation by analyzing API

documents to enable effective security testing. While it demonstrates high accuracy and testing coverage, the study focuses on security rather than technical software engineering. In [19], authors introduced INIT, a simulation environment combining virtual and physical IoT modules, enabling comprehensive security testing by capturing traffic at all system layers. INIT focuses on security testing rather than the broader software engineering challenges of IoT systems. Lastly, authors in [198] proposed IoT-PEN, an automated penetration testing framework for IoT systems, leveraging server-client architecture and target-graphs to identify multi-stage attacks. While it effectively addresses security concerns, the study does not cover technical software engineering aspects of IoT systems. Table 3.3 summarizes and compares the closely related works. The reviewed works high-

Table 3.3: Related Work on Testing Frameworks in IoT Systems

			Framework Coverage					
Ref	**Year**	**Focus**	**D**	**G**	**C**	**A**	**E2E**	**TSEA**
[108]	2024	Lean simulation framework designed for IoT cloud stress testing	††	††	††	- -	- -	- -
[133]	2024	Coverage-based framework for testing IoT applications	- -	- -	- -	††	††	- -
[13]	2024	Framework for fault detection in IoT edge devices	††	- -	- -	- -	- -	- -
[151]	2024	Framework for designing resilient IoT applications	- -	- -	- -	††	- -	- -
[190]	2023	Framework for testing IoT devices	††	- -	- -	- -	- -	- -
[74]	2022	Framework for path selection for effective IoT testing	††	- -	- -	- -	††	- -
[155]	2021	Framework to validate IoT systems	††	††	††	††	- -	- -
[86]	2021	Framework for compliance testing of IoT systems	††	††	††	††	- -	- -
[83]	2021	Non-Functional testing framework of IoT product	††	- -	- -	- -	- -	- -
[22]	2021	Framework for Interoperability and Integration Testing of IoT	††	- -	- -	- -	††	- -
[16]	2019	Framework for testing IoT applications	- -	- -	- -	- -	††	- -
[153]	2018	Framework based on a set of test patterns specific to the IoT domain	- -	- -	- -	- -	- -	- -

✺ **D**: Device or Thing Layer; **G**: Gateway or Network Layer; **C**:Cloud or Service Layer; **A**:Application Layer; **E2E**:End-to-End or System-Level; **TSEA**: Technical Software Engineering Aspects;††: Covered; - -: Not Covered.

light advancements in IoT system testing; however, most lack a comprehensive focus on technical software engineering aspects, a gap we aim to address.

3.4 Functional End-to-End Testing for IoT Systems

Several studies have investigated system-level and component testing of complex systems, including IoT and embedded systems, focusing on performance, functionality, security, and interoperability testing. Kim et al. [90] proposed Testing as a Service (IoT-TaaS), a plug-and-test approach for IoT systems. While addressing conformance, interoperability, and semantic validation, their approach did not focus on validating the entire system's functionalities. Pedro et al. [153] introduced a pattern-based test automation approach for integration testing, focusing on specific test patterns such as periodic readings, triggered readings, actuator validation, alerts, and actions. However, their work was limited to predefined test patterns rather than comprehensive system-level testing. Leotta et al. [104] developed an acceptance testing approach for IoT systems using UML state machine diagrams to model system behaviors. Their method generated test cases from behavioral models by specifying actual data and assertions, making them executable through test scripts. While effective for component testing, this method relied on detailed behavioral models, which may not always be available.

For system-level testing, Wang et al. [191] proposed generating executable test cases for embedded systems from natural language requirements. While reducing manual effort and improving requirement coverage, their approach was designed for embedded systems rather than IoT systems. Miroslav et al. [23] introduced PatrIoT, an interoperability and integration testing approach, primarily focusing on device-level interactions rather than complete system functionality. Similarly, Hu et al. [74] proposed an approach for end-to-end device connectivity testing, introducing test metrics such as eUtility, device coverage, compatibility, and connection coverage. However, this approach primarily addressed connectivity rather than functional testing of IoT systems. Table 3.4 compares studies focusing on functional testing IoT systems.

Table 3.4: Closely Related Work on IoT Systems Testing Approaches

		Target			Inputs			Contribution			Approach	
Study	**Year**	**Systems**	**Scope**	**Functionality**	**UCSs**	**Source Code**	**UML B.M**	**Approach**	**T.T**	**TaaS**	**T.C**	**C.E**
[90]	2018	IoT	Device	- -	NA	NA	NA	††	- -	††	- -	††
[91]	2018	IoT	Device	- -	NA	NA	NA	††	- -	††	- -	††
[153]	2018	IoT	E2E	††	NA	NA	NA	- -	- -	- -	††	††
[104]	2018	IoT	UAT	††	- -	- -	††	††	- -	- -	††	††
[106]	2018	IoT	UAT	††	- -	- -	††	††	- -	- -	††	††
[17]	2019	IoT	Device	††	- -	- -	- -	††	- -	- -	- -	††
[24]	2021	IoT	Device	††	- -	- -	- -	- -	††	- -	- -	- -
[74]	2022	IoT	E2E	††	- -	- -	††	††	- -	- -	††	††
[60]	2022	IoT	E2E	††	- -	- -	††	††	- -	- -	††	††
[191]	2022	Embedded	E2E	††	††	- -	- -	††	††	- -	††	††
[38]	2022	IoT	E2E	- -	††	- -	- -	††	- -	- -	††	††
[95]	2023	IoT	E2E	- -	††	- -	- -	††	- -	- -	††	††
[197]	2024	Trad. Sys.	UAT	††	††	- -	- -	††	- -	- -	††	††

✵ **UCSs**: Use Case Specifications; **T.C**: Test Cases; **C.E**:Conducted Experiment; **T.T**:Testing Tool; **TaaS**:Testing as a Service; **Trad. Sys.**: Traditional Systems; **UAT**:User Acceptance Testing; **UML B.M**: UML Behavioral Model;††: Covered; - -: Not Covered; **N.A**: Not Applicable.

A major challenge in IoT system testing is generating and executing tests that validate system-level behaviors across multiple layers, including heterogeneous components, diverse communication protocols, and complex interactions among devices, gateways, networks, cloud services, and applications [128]. Traditional testing approaches often rely on behavioral models or source code, which are not always accessible. While system-level testing approaches such as those by Leotta et al. [104] and Wang et al. [191] have advanced IoT system validation, they remain limited in practical applicability due to their reliance on detailed behavioral models or source code, requiring significant manual effort for test generation and execution. Existing approaches for testing IoT functionalities often target specific system layers and assume the availability of source code or behavioral models. However, these models may lack sufficient detail for test automation [191], and access to source code for all IoT components is not always feasible. Unlike prior work that relies on these constraints and requires manual intervention, this thesis proposes an approach for generating and executing functional end-to-end tests for IoT systems directly from use case specifications, eliminating dependence on behavioral models or source code and enabling automated bug detection.

Chapter 4

Systematic Literature Review on IoT Systems Testing

4.1 Introduction

This chapter presents our literature review, summarizing and synthesizing the current state of IoT systems testing. The chapter examines key aspects of IoT system testing such as testing objectives, approaches, tools, and challenges. By investigating these aspects, we can understand the limitations of existing approaches and the associated challenges while testing IoT systems. Despite several reviews conducted in recent years on IoT systems testing, there was no systematic literature review (SLR) that collectively examines testing objectives, approaches, tools, and challenges. Thus, we conducted an SLR to investigate these four aspects holistically. This chapter presents our literature review, summarizing and synthesizing the current state of IoT system testing. It is guided by the following research questions (RQs):

- **RQ1:** How are IoT systems tested from a research perspective?
  - **RQ1.1**: What are the testing objectives considered?
  - **RQ1.2**: What are the testing approaches investigated?

  - **RQ1.3**: What are the testing tools investigated?
  - **RQ1.4**: What are the IoT systems used for evaluation?
- **RQ2:** What challenges do researchers report in IoT system testing?
  - **RQ2.1**: What are the testing challenges identified?
  - **RQ2.2**: How are testing challenges addressed?

By answering these RQs, we provide insights into the limitations of existing approaches and the challenges practitioners face in testing IoT systems. These insights contribute to the development of effective and specialized testing solutions for ensuring the quality of IoT systems. We followed the updated PRISMA guideline for systematic reviews [145], screening 8,294 potentially relevant studies from eight digital libraries published between 2012 and 2022. Applying inclusion and exclusion criteria, along with snowball sampling, we assessed the quality of these studies based on study design, methodology, analysis, conclusions, and implications. Ultimately, we retained 83 Primary Studies (PSs), which we analyzed to extract findings related to testing objectives, approaches, tools, and challenges. Testing in IoT systems serves multiple objectives, such as identifying defects, ensuring quality, and validating user requirements. In this chapter, we focus on testing objectives linked to the quality attributes of IoT systems. The key contributions of this chapter are as follows:

- Compilation of *47 quality attributes*, with *5 specifically related to IoT systems*, to provide a comprehensive understanding of testing objectives;
- Overview of testing approaches, including *4 levels of testing*, *10 types of testing*, and *15 testing techniques and test practices* for IoT systems;
- Compilation of *19 user testing tools* and *15 testbeds* used in IoT systems testing, highlighting their applicability at different stages of IoT development;
- Identification of *challenges encountered in IoT systems testing* and potential research directions for addressing these challenges;

- Compilation of *IoT systems used in evaluating testing techniques and approaches*, documenting their characteristics for potential future reuse.

The remainder of this chapter is structured as follows: Section 4.2 describes our research methodology. Section 4.3 presents answers to our research questions, while Section 4.4 discusses the implications of these findings. Section 4.5 outlines possible threats to the validity of our study. Finally, Section 4.6 concludes the chapter with key takeaways.

4.2 Research Method

We followed the updated PRISMA guidelines [146, 147] and Kitchenham et al. guidelines [94] to review and report our findings. We used three main phases: planning, conducting, and reporting the review. During the planning phase, we define the objective of SLR and review protocol. The objective of this SLR is defined in Section 4.1. The review protocol for conducting this SLR is defined in this section. It consists of six steps: 1. defining the research questions, 2. formulating the search query, 3. selecting the studies, 4. snowballing, 5. assessing the quality of the studies, and 6. extracting and analyzing the data.

4.2.1 Research Questions

- **RQ1.1: What Are Testing Objectives Considered?**

  Rationale: To ensure the quality of IoT systems, it is imperative to gain a clear understanding of the specific testing objectives, which can be achieved by assessing various quality attributes. While traditional system quality attributes can be applied, the complexity, heterogeneity, and distributed nature of IoT systems may necessitate the inclusion of additional attributes.

- **RQ1.2: What Are Testing Approaches Investigated?**

  Rationale: Practitioners may use several approaches to test IoT systems. We want to identify different testing approaches used to assess the quality of IoT systems. Understanding the available approaches for testing IoT systems provides valuable insights into

the current state of the field and guides future research. This knowledge can help to improve the current testing practices and identify opportunities for enhancements.

- **RQ1.3: What Are Testing Tools Investigated?**

  Rationale: The tools for testing IoT systems hold significant importance and offer valuable assistance to both practitioners and researchers. Knowledge of these tools provides insights for researchers to explore potential enhancements, while also enabling practitioners to make informed choices that align with their specific requirements. By understanding the landscape of available testing tools, the testing process in IoT systems can be improved and tailored to better meet the needs of both researchers and practitioners.

- **RQ1.4: What Are IoT Systems Used For Evaluation?**

  Rationale: Developers may have different validation and verification objectives for different IoT systems. Knowing which systems are used for evaluation enables meaningful comparisons, providing valuable insights into the strengths and weaknesses of different approaches. This information is also essential for tailoring testing approaches to specific types of IoT systems.

- **RQ2.1: What Are Testing Challenges Identified?**

  Rationale: Testing an IoT system involves assessing multiple components, which can be tested individually or in combination with other components. The complexity inherent in IoT systems introduces unique challenges when it comes to ensuring their quality and reliability. While various approaches and tools exist for testing IoT systems, it is important to identify specific challenges that may impact the overall quality of these systems. By understanding and addressing these challenges, we can enhance testing processes, approaches, and tools to improve the overall quality and reliability of IoT systems.

- **RQ2.2: How Are Testing Challenges Addressed?**

  Rationale: Different PSs focused on different testing objectives, approaches, and tools

as well as IoT systems. Our objective is to understand which challenges have been successfully addressed and to identify those that remain unaddressed. This information is valuable for practitioners seeking solutions to testing challenges and researchers wanting to address challenges.

This chapter focuses on reviewing existing studies that studied testing IoT systems. The goal is to identify and discuss the findings of these studies and their insights into various aspects of IoT systems testing. To select the relevant studies, we applied the search query as illustrated in Section 4.2.2.

4.2.2 Search Query

We write our search query using the PICO (Population, Intervention, Comparison, Outcome) framework [33]:

- Obtaining the main terms from RQs.
- Identifying the possible synonyms of the main terms.
- Applying the Boolean OR to combine possible synonyms of the main terms.
- Applying the Boolean AND to combine expressions in the previous step.

As a result, we formulated the following search query:

> (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)

4.2.3 Studies Selection

Databases Identification:

We selected 8 online digital libraries: ACM Digital Library, Compendex, IEEE Xplore, ScienceDirect, SpringerLink, Scopus, Web of Science, and Wiley. These digital libraries are the most commonly used to search for studies when conducting literature reviews in software engineering, as suggested by Dyba et al. [44]. Figure 4.1 shows the selected digital libraries.

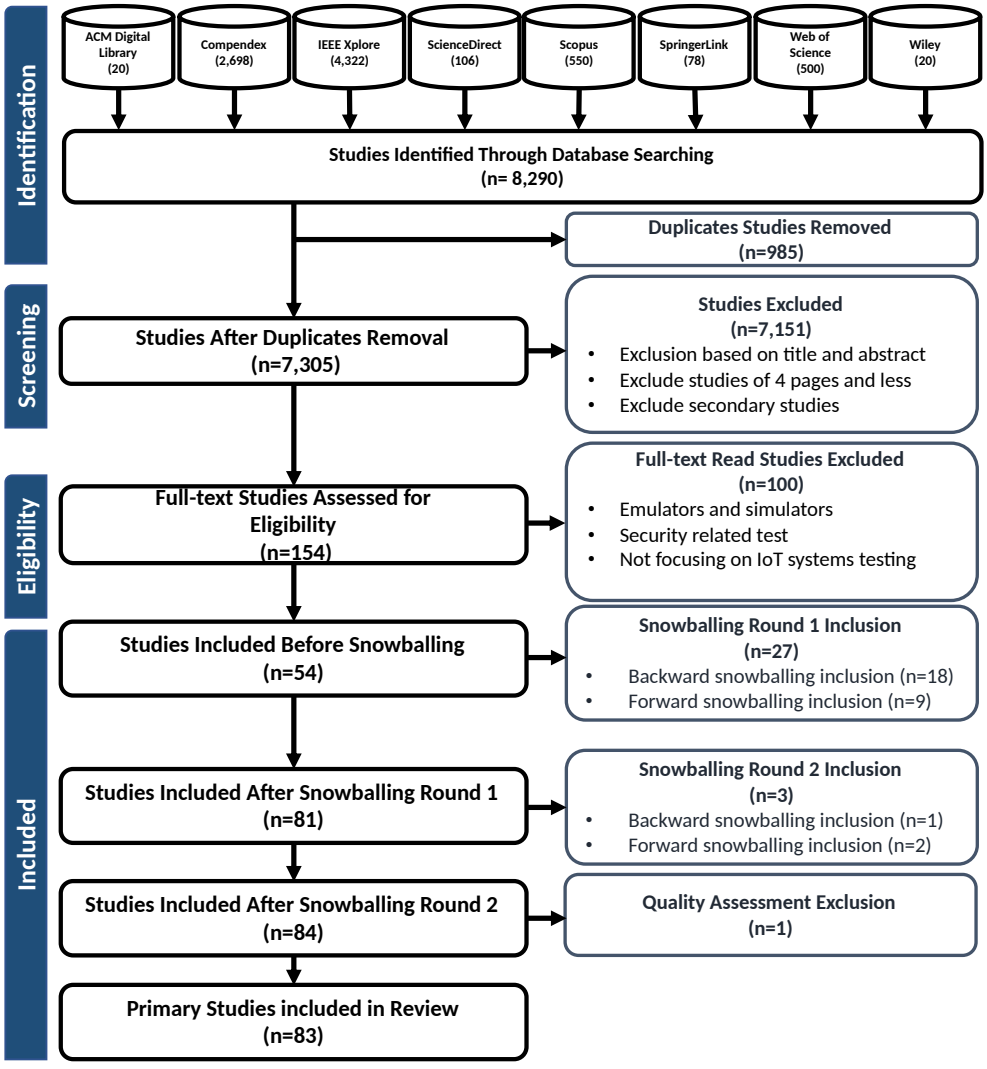


Figure 4.1: PRISMA Flow for Primary Studies Selection

We searched each of these digital libraries using our search query. Some digital libraries have limitations when performing queries. ScienceDirect accepts a maximum of eight connectors in search queries, SpringerLink does not accept the use of parentheses, ACM digital library does not accept wildcards. We customized the search query based on the specificities of each digital library. We applied online built-in filters to obtain 8,294 studies:

- Publication period: **2012-2022**
- Publication language: **English**
- Publication venues: **conferences**, **journals**, or **workshops**
- Excluded **security** testing studies.

Duplicates Removal:

We exported the bibliographic entries to a spreadsheet for analysis. We used Excel's built-in filtering and sorting capabilities and manually removed 989 duplicate studies.

Screening:

We defined a set of inclusion and exclusion criteria. We applied these criteria to select relevant primary studies and exclude irrelevant ones.

Inclusion Criteria: We considered the following inclusion criteria for PSs selection:

- The study is written in English.
- The study is published between 2012 and 2022.
- The study is published in journals, conferences, or workshops.
- The study explicitly discusses at least one aspect of IoT systems testing.
- The study has at least 4 pages.

Exclusion Criteria: We considered the following exclusion criteria:

- The study has less than 4 pages.
- The study is not a primary study.
- The study focuses on IoT security testing.
- The study focuses on emulators and simulators.

- The study has not been peer-reviewed.
- The study is a graduate thesis or project report.
- The study does not have its full text available online.
- The study does not provide enough details.

We applied our exclusion criteria in two steps based on the titles, abstracts, and full texts. We applied the first six criteria to the titles and abstracts, while we applied criteria 7 and 8 to the full texts. We chose to exclude PSs about IoT security testing despite the importance of this topic for two reasons. First, this topic deserves its own SLRs and has been already the subject of recent surveys, e.g., [109, 51, 65]. Second, security testing for IoT systems is a vast and complex topic, which would have overshadowed other objectives and increased the complexity and length of this article. We initially excluded security studies using online, built-in filters. Some studies using different, specific terms like *penetration testing* or *malware detection* remained in the retrieved primary studies. We applied the third exclusion criteria to remove those security studies manually. We excluded emulators and simulators in this SLR, as a comprehensive comparison of these tools has been previously conducted [148, 29] and we did not identify any new emulator or simulator that was not mentioned in these studies. To ensure that only relevant studies are included in our analysis, we added the last exclusion criteria to eliminate any study that discussed IoT systems testing aspects in a general sense, without providing sufficient detail on a specific objective, approach, tool, or challenge. While reading some studies, we observed that they did mention these aspects in their abstracts, introductions, or conclusions without providing enough information on the methodology or results to meet our inclusion criteria.

During the initial screening phase, we assessed the titles, abstracts, page counts, and whether qualified as a primary study or not. Two of our authors independently conducted the screening process using the defined exclusion and inclusion criteria. To ensure consistency when applying our inclusion and exclusion criteria, we compared the screening

results for 30 randomly selected studies with Cohen's Kappa [32] shown in Equation 1.

$$\kappa = \frac{P_o - P_e}{1 - P_e} \tag{1}$$

where:

$P_o =$ relative observed agreement among authors

$P_e =$ hypothetical probability of chance agreement

We calculated Cohen's Kappa and obtained almost perfect agreement (k= 0.938). This value underscores the consistency of our screening approach. Throughout the screening process, the two authors met regularly to review their results, resolving disagreements through consensus and discussion. Upon completion of the first screening process, a total of 154 studies are selected for full-text review as potential candidates for PSs.

Eligibility Assessment:

In the second round of the screening process, we used three criteria: emulators and simulators, security-related studies, and studies discussing certain aspects of IoT systems without enough details. Two authors independently applied these criteria to 154 studies by thoroughly reading them. To ensure a shared understanding of our inclusion and exclusion criteria, we compared the results of randomly selected 20 studies from the pool of 154 using Cohen's Kappa. We used these values to calculate the Kappa agreement, and we obtained a nearly perfect agreement (k=0.827). This value indicates the consistency between the two authors. We proceeded with confidence to complete the eligibility assessment process for the remaining studies. At the end of the process, we obtained a total of 54 PSs that met our eligibility criteria.

4.2.4 Snowballing

This chapter has two threats to sampling adequacy.

1. Some of the relevant studies could be published in venues that are not indexed by the

chosen databases, or studies excluded by the filters we used.

2. Some of the relevant studies could be published using some keywords that we did not include in our search query. For example, a study published on "IoT debugging", "IoT troubleshooting", or "IoT monitoring".

We conducted backward and forward snowballing in two rounds to find more studies.

Snowballing Round 1:

We considered the references of all PSs to identify additional relevant studies. For each of the 54 studies identified in Section 4.2.3, we collected the cited references, leading to a total of 3,453 studies. We removed any study retrieved in our initial search and studies not focusing on IoT systems testing, and we retained 101 potentially relevant studies. We screened the 101 potentially relevant studies by removing duplicates (52 studies) and then applied the same inclusion/exclusion criteria as before, and we found 18 additional PSs.

Similarly, we used Google Scholar to identify all the studies that cited the selected 54 studies. We identified 1,645 studies. We removed any study retrieved previously and studies not focusing on IoT systems testing. We retained 152 potentially relevant studies for further screening. We screened these 152 potentially relevant studies by removing duplicates and applying the same inclusion and exclusion criteria as before. We added 9 PSs. By the end of this round 1, we found 27 additional PSs.

Snowballing Round 2:

We considered the 27 studies found in the previous round, and we went through the snowballing process for them again. We checked all the references used in these 27 studies, and we found 5 new potential studies. We screened these 5 studies, and we obtained 1 PS based on the exclusion and inclusion criteria defined before.

We used Google Scholar to identify all studies that cited these 27 studies. We found 6 new potential studies. We applied the same exclusion and inclusion criteria and kept 2 additional PSs. After this round, we obtained 3 additional PSs. Table 4.1 summarizes the

results of this process.

Table 4.1: Backward and Forward Snowballing

Snowballing	Round	Retrieved	Included
Backward	Round 1	101	18
Forward	Round 1	152	9
Backward	Round 2	5	1
Forward	Round 2	6	2
Total		**264**	**30**

4.2.5 Quality Assessment

We devised a set of guidelines, following the recommendations of Kitchenham et al. [94], to assess the quality of primary studies (PSs). Subsequently, we formulated a quality checklist consisting of 19 questions in five categories: study design, conduct, analysis, conclusion, and implication. Each question is answered with a choice of "No," "Partially," or "Yes," and scores of 0, 0.5, and 1 are assigned to these responses, respectively. Two authors applied the checklist independently on each PS. We compared the results and resolved any discrepancies through discussion. The outcome of this evaluation is the percentage of PSs answering each question in the checklist, as shown in Table 4.2.

We calculated the quality of each study by adding the scores of all applicable questions and calculating the corresponding final percentage. The authors agreed to keep the studies with at least a 75% score. Consequently, one study, which scored less than the set threshold, was excluded. We thus obtained 83 PSs.

4.2.6 Data Extraction and Analysis

To obtain the data required to answer each research question (RQ1-RQ6), we studied each PS. We extracted the various data items described in Table 4.3, by applying Algorithm 1. The data extraction form we used is publicly available on Ptidej or on Zenodo websites.

The **Data Extraction Algorithm** (Algorithm 1) provides a structured approach for

Table 4.2: Quality Assessment Checklist

ID	Question	Percentage of PSs			
		Yes	**Partially**	**No**	**N/A**
Design					
QA1	Are the IoT systems testing or quality assurance activities clearly stated?	100.0%	0.0%	0.0%	0.0%
QA2	Are the studies clearly discussed either testing tools, approaches, quality attributes, or testing challenges of IoT systems?	100.0%	0.0%	0.0%	0.0%
QA3	Are the aims of the studies clearly stated?	100.0%	0.0%	0.0%	0.0%
QA4	Are the RQs relevant?	100.0%	0.0%	0.0%	0.0%
Conduct					
QA5	Are the components of IoT systems addressed in the studies clearly stated?	85.5%	2.4%	0.0%	12.0%
QA6	Are the experiments or case studies conducted?	85.5%	2.4%	0.0%	12.0%
QA7	Are the details of the system under test described?	85.5%	2.4%	0.0%	12.0%
QA8	Are the results of the studies validated?	85.5%	2.4%	0.0%	12.0%
Analysis					
QA9	Are the aims of the analysis clearly stated?	98.8%	0.0%	1.2%	0.0%
QA10	Are specific tools or algorithms used to analyze the data?	63.9%	3.6%	20.5%	12.0%
QA11	Are the data used in the analysis clearly stated in PSs?	81.9%	0.0%	3.6%	14.5%
QA12	Is the statistical analysis performed correctly?	81.9%	0.0%	3.6%	14.5%
QA13	Are the data and/or tools used available?	96.4%	0.0%	3.6%	0.0%
Conclusion					
QA14	Are validity threats discussed in PSs?	84.3%	0.0%	15.7%	0.0%
QA15	Are the results compared with state-of-the-art practices?	72.3%	0.0%	27.7%	0.0%
QA16	Do the results support the conclusions?	96.4%	0.0%	3.6%	0.0%
Implication					
QA17	Did PSs extend or improve the existing tools/approaches?	85.5%	2.4%	0.0%	12.0%
QA18	Did PSs discuss future research as an improvement or enhancement of proposed tools or approaches?	85.5%	2.4%	0.0%	12.0%
QA19	Did PSs discuss any solution for IoT systems testing challenges?	4.8%	0.0%	0.0%	95.2%

manually classifying studies and extracting relevant data in a systematic manner. The algorithm ensures consistency in categorizing studies based on predefined themes and facilitates the organization of extracted information.

The process begins with a *set of studies* (S) to be analyzed and a predefined *set of categories* (N), which include **Quality Attributes (QA), Testing Approaches (TA), Testing Techniques (TT), and Testing Challenges (TC)**. The algorithm guides the researcher in reviewing each study (P_i) and determining its *primary category* (N_j). If a match is found within the predefined categories, the study is assigned to that category ($K[j]$). The

Algorithm 1 Data Extraction Algorithm

Require: A set of studies S
Ensure: Studies in each category K

$K \leftarrow \emptyset$
$i \leftarrow 1$
Let $S \leftarrow Studies$
Let $N = \{QA, TA, TT, TC\}$
while $S \neq \emptyset$ **do**
    $P_i \leftarrow$ Select study from S
    category $\leftarrow$ Identify primary category of P_i
    if category $\in N$ **then**
        Assign P_i to K[category]
        Find another category for P_i
        if new category $\in N$ **then**
            Assign P_i to K[new category]
        else
            Flag P_i for discussion
        end if
    else
        Flag P_i for discussion
    end if
    Extract relevant data and update extraction form
    Remove P_i from S
    $i \leftarrow i + 1$
end while
return K

Table 4.3: Data Extraction Elements

#	Data Item	Description	RQs
1	Code	Unique Identifier of the PS.	
2	Study Title	The title of the PS.	
3	Year	Year the PS was published.	
4	Venue	Where the PS was published.	
5	Source	The source of the PS.	
6	Author(s)	The authors of the PS.	
7	Focus	The main focus of the PS.	
8	Contribution	The key contribution of the PS.	
9	Category	Category of the study.	
10	Tools	The name of the proposed tool.	RQ3
11	Layer	IoT layer studied.	
12	Approach	Approach used.	RQ2,RQ4,RQ6
13	QA	Quality attributes discussed.	RQ1
14	Challenge	Challenge discussed in the PS.	RQ5, RQ6
15	Research or Evaluation Method	The method used in the PS.	RQ4

researcher then checks whether the study belongs to additional categories, ensuring that studies contributing to multiple themes are appropriately classified. If a study does not fit any predefined category, it is flagged for discussion among the authors to determine its appropriate classification. After categorization, the researcher extracts relevant data elements and records them in a **data extraction form**. The study is then removed from the dataset, and the process is repeated for the next study until all studies are reviewed. While this process is conducted manually, following this structured algorithm ensures consistency, reduces classification bias, and enables a systematic synthesis of findings in the study.

4.3 Results

In this section, we answer our research question "RQ1: How are IoT systems tested, and what challenges do researchers report?". We divide our RQ into the following subquestions:

1. **RQ1.1**: What Are Testing Objectives Considered?
2. **RQ1.2**: What Are Testing Approaches Investigated?

3. **RQ1.3**: What Are Testing Tools Investigated?
4. **RQ1.4**: What Are IoT Systems Used For Evaluation?
5. **RQ2.1**: What Are Testing Challenges Identified?
6. **RQ2.2**: How Are Testing Challenges Addressed?

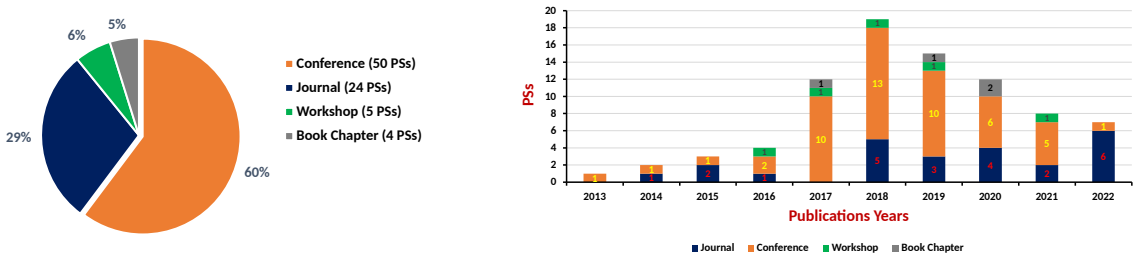
4.3.1 RQ1:How are IoT systems tested from a research perspective?

In the following subsections, we describe the results of each question and the corresponding observations. Prior to presenting answers to our research questions, we start with the preliminary findings (PF) of bibliographic data. We present the main areas of research focus in IoT systems testing and commonly used research methods.

PF1: What Publication Trends Are Observed?

- **PF1.1: Publication Trends Over the Past Ten Years** Figure 4.2 shows the trend of publications on IoT systems testing over the past 10 years. We observed a steady increase in PSs, with a peak in 2018 when 19 PSs (22.9%) were published. We observed a decrease from 2019 until the end of 2022. Most PSs are published in conferences. We did not find any PS published in 2012. The first PS was published by Reetz et al. in 2013 [271]. From 2014 to 2016, on average, 3 PSs were published each year, and at least 1 PS was published in a journal. In 2017, no PS was published in a journal. However, 10 PSs were published in conference proceedings. The majority of PSs (87.95%) were published during the period 2017-2022. Despite a slight continuous decrease in PSs since 2019, we observed that 6 PSs (7.23%) were published in journals in 2022.

  The 83 PSs were published in a total of 66 different venues, including 40 conferences, 18 journals, 5 workshops, and 3 book series, as shown in Table A.1. Among the journals, *IEEE Access* and *IEEE Internet of Things Journal* stand out with 4 PSs each, accounting for 4.8% of the total. Each remaining journal has only 1 PS. In terms of conferences, the *ACM/IEEE Conference on Internet of Things Design and Implementation* has the highest number of PSs with 3 (3.6%), and 9 other conferences have 2 PSs each. Each remaining



(a) Distribution of PSs by Study Type

(b) Distribution of PSs by Year

Figure 4.2: Distribution of PSs

30 conferences has only 1 PS.

Takeaway for PF1.1

IoT systems testing studies showed a steady increase until 2018, followed by a slight decline. The majority of PSs were published at conferences, but there has been a recent rise in journal articles.

- **PF1.2: Top Active Authors and Affiliations** The 83 PSs were written by 288 unique authors. Table A.2 shows the top active authors, who are from both industry and academia. The most active contributors to our PSs are from the industry, which may suggest that industry-specific challenges are the driving force behind the research initiatives in this field. Le Gall coauthored the highest number of PSs (7.23%) [222, 236, 277, 223, 242, 212] on *interoperability testing*, *conformance testing*, *testing framework*, and *model-based testing as a service*. Several authors published 4 PSs, including Leotta, Ricca, and Watteyne. There are multiple authors with 3 PSs, such as Ahmad, Ancona, Franceschini, Gutierrez-Madronal, Baqa, Kuemper, Medina-Bulo, Olianas, Ribaudo, and Toenjes. Some authors such as S. Ahmed, Ari, L. Badr, Baranwal, Bellekens, Bures, Bonnet, Clerissi, and others have 2 PSs each.

  Authors are from multiple countries as shown in Figure 4.3, indicating a global interest in the topic. Many countries have only one PS (1.2% of total PSs). Some countries have 2 or 3 PSs (2.4% or 3.6%). Few countries, such as China, South Korea, and Taiwan

have higher representation with 4 or more PSs (4.8% or more of the total PSs). France has the highest number of PSs (11 PSs), accounting for 13.3%, followed by the United Kingdom, United States, Germany, and Spain, each with 8 PSs (9.6%).

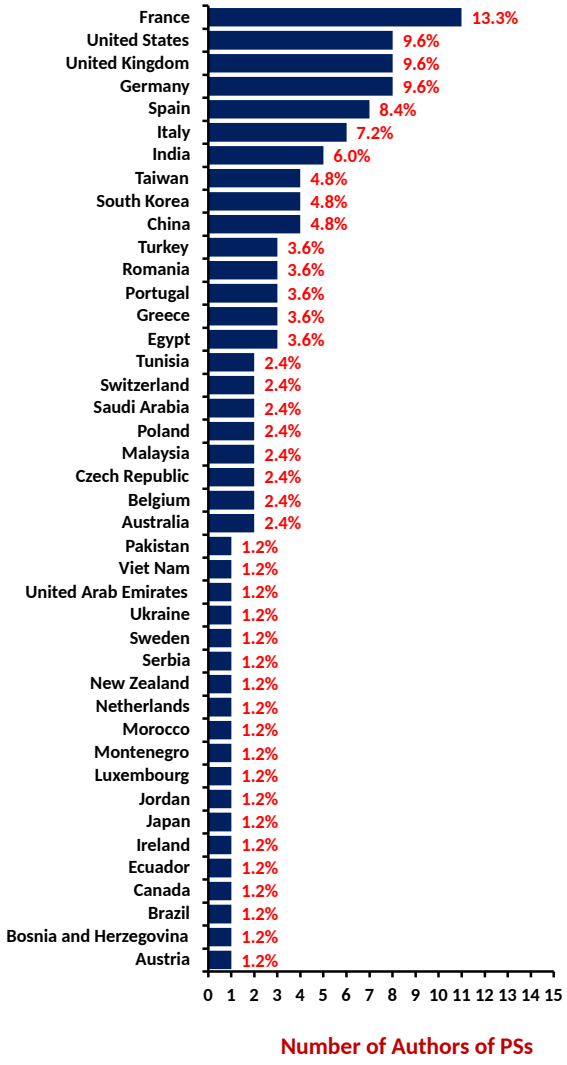


Figure 4.3: Authors' Countries

EGM in France has the highest number of PSs (7.23%). INRIA in France follows closely with 5 PSs (6.0%). University of Cádiz in Spain, Università di Genova in Italy, and Czech Technical University in Prague, Czech Republic have 3 PSs (3.6% of the total PSs) each. Several institutions, including the University of Porto in Portugal, TU Berlin in Germany, EURECOM in France, Institute of Computer Science at the University of Osnabrueck in

Germany, and University of Applied Sciences Osnabrueck in Germany, and others as shown in Table 4.4 have 2 PSs each.

Table 4.4: Affiliations of authors with at least 2 PSs

Affiliation Name	**# PSs**	**Affiliation Name**	**# PSs**
EGM, France	6	Université de Franche-Comté, France	2
INRIA, France	5	University of Porto, Portugal	2
University of Cádiz, Spain	3	TU Berlin, Germany	2
Università di Genova, Italy	3	EURECOM, France	2
Czech Technical University in Prague, Czech Republic	2	University of Osnabrueck, Germany	2
Banaras Hindu University, India	2	University of Applied Sciences Osnabrueck, Germany	2
University of Alabama at Birmingham, USA	2	Özyeğin University, Turkey	2
Ain Shams University, Egypt	2	IRISA, France	2
Al-Baha University, Saudi Arabia	2	Mandat International, Switzerland	2
University of Sfax, Tunisia	2	University of Surrey, United Kingdom	2
Sejong University, South Korea	2	University of Cantabria, Spain	2
Korea Electronics Technology Institute, South Korea	2		

✵ **PSs**:Primary Studies.

Takeaway for PF1.2

Notable contributors are Le Gall, Leotta, Ricca, and Watteyne. The most active institutions are EGM and INRIA. Prominent countries include France, the United Kingdom, the United States of America, Germany, and Spain. The findings reflect a global interest in IoT systems testing and highlight the active participation of industry experts.

- **PF1.3: Collaboration in PSs Co-authorship by country**: We considered the countries with at least two PSs. Authors of PSs in France collaborated with other authors from 7 different countries on 11 PSs. The authors of PSs in the USA collaborated with authors from three different countries. Our observation suggests that international collaboration in IoT systems testing is critical, with France serving as a prominent hub for collaboration with authors from multiple countries.

Co-authorship by affiliation: We identified 132 different institutions in 83 PSs. We considered institutions with at least 2 PSs. We assigned a unique number to each institution. Table 4.4 shows the assigned number (code) we used for the top 23 institutions. We provided the entire list of all institutions and their assigned code in the replication package. We used VOSViewer [1] package to visualize the collaboration between different institutions. EGM has 17 links. Institutions like INRIA in France, Sejong University and Korea Electronics Technology Institute in South Korea, EURECOM in France, and the University of Cantabria in Spain collaborated with EGM. INRIA has 22 links. Seventeen institutions collaborated with INRIA on 1 PS, while five institutions collaborated with INRIA on at least 2 PSs. The institutions with more collaborations are industry-based.

Co-authorship by Authors: We assessed the collaborative efforts among authors by examining those who participated in at least two primary studies (PSs). Our analysis revealed that several authors collaborated on multiple PSs. One such author is Le Gall of EGM, who collaborated with eight other authors on at least two PSs. Maurizio and Filippo, both from Università di Genova in Italy, collaborated in all four PSs [250, 264, 249, 251] they coauthored. They collaborated with six other authors from the same institution. Thomas of INRIA in France also had four PSs and collaborated with authors from IRISA (Federico and Cesar) and Sebastien (Mandat International). The analysis suggests that collaboration among authors is common, with some authors collaborating on multiple PSs.

[1] https://www.vosviewer.com/

Takeaway for PF1.3

Several researchers have extensive collaborations with authors from other countries, which may evidence the global reach of their work. Additionally, the involvement of authors from the industry in these collaborations may underscore the practical and real-world applicability of their research efforts, which may contribute to more comprehensive and impactful approaches to IoT systems testing.

PF 2: What Are Aspects of Testing Studied?

We analyzed the 83 PSs to identify the focus of each PS. Some of PSs ([231], [235], [223], [232], [267], [224], [275], [268], [264], [260], [239], [225], [288], [216], [251], [246]) focused on more than one aspect, therefore, we agreed to focus on the primary and secondary aspects for each PS and considered PSs falling under these aspects to address the RQs. Once each author completed the analysis, we calculated Cohen's Kappa and obtained almost perfect agreement (k= 0.9). We solved any disagreement through discussion among the team members. Table 4.5 shows the focus of each PS. We noticed that Some PSs discussed more than one testing aspect. For example, PSs such as [231, 251] primarily focus on testing approaches but also mention some testing tools. The primary category for each PS refers to the testing aspect that received more emphasis in the study, while the secondary category relates to other aspects mentioned in the PS with less emphasis.

Out of the 83 PSs studied, the majority (56 or 67.4%) relate to approaches for testing IoT systems. Of those 56 PSs, 50 PSs (60.2%) focused primarily on testing approaches, while 6 PSs (7.23%) discussed testing approaches together with other aspects. Another aspect discussed in PSs is testing tools with 25 PSs (30.1%). Of those 25 PSs, 21 PSs primarily focused on different tools used for testing IoT systems. Additionally, 4 PSs discussed testing tools alongside other aspects. 29 PSs (34.9%) discuss test objectives based on quality attributes. Of those 29 PSs, 8 PSs (9.6%) primarily focused on test objectives, while 21 PSs (25.3%) discussed test objectives alongside other aspects. Testing challenges

Table 4.5: Areas of Focus for IoT Systems Testing in PSs

Category	PSs	% PSs
Quality Attributes		
Primary Category	[219], [227], [280], [221], [279], [243], [257], [239]	9.6%
Secondary Category	[261], [222], [236], [229], [215], [242], [275], [284], [228], [212], [218], [226], [213], [260],[276], [248], [255], [258],[214], [241], [246]	25.3%
Testing Approaches		
Primary Category	[290], [259], [237], [270], [265], [250], [222], [236], [231], [235], [223], [229], [247], [278], [215], [252], [271], [244], [269], [263], [242], [287], [233], [275], [268], [212], [283], [218], [226],[266], [238], [210], [285], [274], [213], [282], [260], [230], [225], [255], [288], [216], [217], [258], [245], [249], [214], [251], [281], [246]	60.2%
Secondary Category	[232], [291], [267], [224], [264], [239]	7.23%
Testing Tools		
Primary Category	[261], [209] ,[277], [267], [211], [240], [289], [220], [224], [234], [262], [284], [228], [286], [272], [264], [273], [276], [248], [241], [253]	25.3%
Secondary Category	[231],[223], [268],[260], [251]	4.8%
Testing Challenges		
Primary Category	[232], [291], [254], [256]	4.8%
Secondary Category	[235], [225], [288], [216]	4.8%

are the least studied aspect, with only 8 PSs (9.64%) covering this topic. Of those 8 PSs, only 4 PSs (4.8%) primarily focused on testing challenges, while the other 4 PSs discussed testing challenges alongside other aspects.

We analyzed PSs to understand which layers of IoT systems are discussed. The most discussed layer is the *device layer* with 63 PSs, accounting for 75.90%. Fifty PSs (60.24%) discuss testing the *network layer*. Forty PSs (48.19%) discuss the *application layer* testing. Eighteen PSs (21.69%) mention the testing of the *cloud layer*. We observed that the authors of PSs used different terms to refer to describe approaches. Some authors used the term *framework* [270, 235, 247, 252, 242, 268, 264, 218, 266, 245, 246], while others used the same term to denote a tool [267, 241]. Similarly, some authors referred to it as a *platform* [232], while others used the same term to describe a tool [223, 224]. Other authors used

the term *methodology* [274, 246]. Others used the term *algorithm* [290, 259, 237, 285, 274, 213], while others used the term *test strategy* [225]. 55 PSs (66.2%) discuss testing approaches in the form of testing practices, techniques, types, and levels. 15 PSs (18.0%) provide details on testbeds, while 19 PSs (22.8%) mention end-user testing tools. We provide more details on testbeds, testing techniques, types, and levels in RQ2 and RQ3.

Takeaway for PF2

The device layer was the most discussed layer for IoT system testing. Cloud layer is the least considered in the selected PSs. Testing approaches include testing frameworks and platforms. End-user testing tools and testbeds are commonly discussed in PSs.

PF 3: What Research/Evaluation Methods Are Used?

Table 4.6 provides an overview of the research and evaluation methods used in PSs. *Experiments* emerge as the most popular choice among PSs, with forty-three of the total PSs opting for this method. *Case studies* stand out as the second most preferred method, with thirteen PSs. We observed that the findings from these studies may have limitations on the generalizability of the results. *Surveys* rank as the third preferred method, with eleven PSs. Surveys can provide valuable insights into the perceptions and opinions of IoT practitioners. However, they have potential limitations and criticisms, including the risk of bias, low response rate, and dependence on self-reported data. *Combining case studies with experiments* are also common research methods used in PSs, with eleven PSs. We observed that combining case studies and experiments can enhance the validity of the findings. *Grounded theory* is the least used research method, with five PSs: four PSs focused on testing approaches [225, 255, 245, 281], and one PS addressed quality issues [243].

Table 4.6: Research/Evaluation Methods

Research Method	PSs	#	%
Experiments	[290, 259, 270, 265, 209, 236, 277, 223, 291, 211, 240, 229, 247, 289, 215, 220, 224, 269, 234, 262, 263, 275, 284, 228, 286, 272, 212, 226, 266, 238, 210, 285, 274, 213, 273, 276, 239, 248, 288, 217, 241, 253, 246]	43	51.8%
Case Studies	[222, 235, 267, 278, 271, 244, 242, 287, 233, 283, 218, 260, 216]	13	15.7%
Surveys	[219, 231, 254, 256, 280, 221, 279, 257, 282, 230, 258]	11	13.2%
Case Studies with Experiments	[261, 237, 250, 227, 232, 252, 268, 264, 249, 214, 251]	11	13.2%
Ground Theory	[243, 225, 255, 245, 281]	5	6%
Total		**83**	**100%**

Takeaway for PF3

Experiments are the most commonly used evaluation method, with case studies and surveys also being used. Some studies combined case studies and experiments, allowing for an in-depth exploration of specific situations while also providing evidence of the generalizability of the proposed solutions. In contrast to studies relying solely on either case studies or experiments, those combining both methods yield comprehensive results.

RQ1.1: What Are Testing Objectives Considered?

To understand and evaluate the quality of IoT systems, researchers considered various testing objectives. These objectives are expressed in terms of quality attributes. We collected the quality attributes investigated in IoT systems. A quality attribute (QA) refers to a measurable or testable property of a system, that measures the degree to which the system meets this attribute. While traditional software systems have well-known quality attributes [2], no universally recommended quality attributes for IoT systems. Our objective is to identify the quality attributes of IoT systems in PSs. To understand why IoT systems should be tested, these quality attributes serve as test objectives.

We conducted a thorough analysis of quality attributes discussed in PSs to identify

various objectives considered for testing IoT systems. Table 4.7 reports all quality attributes identified in PSs. In addition to PSs discussing quality attributes as primary or secondary categories as shown in Table 4.5, We also included any quality attributes from other PSs that discuss them in general.

Table 4.7: Quality Attributes Reported in PSs

PS	A	B	C	D	E	F	G	H	J	K	L	M	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU
[219]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[261]	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-
[222]	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[236]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[223]	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[227]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[291]	-	-	-	✓	-	✓	-	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[267]	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[229]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[215]	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
[242]	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[275]	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[284]	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[228]	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
[212]	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[218]	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[226]	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[280]	-	✓	-	-	-	✓	-	✓	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	✓	-	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[221]	-	-	-	✓	-	-	✓	✓	✓	-	-	✓	✓	-	✓	-	-	-	-	✓	-	✓	-	✓	-	-	-	-	-	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-
[279]	-	✓	-	-	-	✓	-	✓	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	✓	-	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[213]	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[243]	-	-	-	-	-	✓	-	✓	✓	-	-	✓	-	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[257]	-	-	-	✓	-	-	✓	✓	✓	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	✓	✓	-	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-
[260]	-	-	-	-	-	-	-	✓	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	✓	-	-	-	-	-
[276]	-	-	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
[239]	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[248]	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[255]	-	-	-	-	-	✓	-	✓	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[217]	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[258]	-	-	-	✓	-	-	-	✓	-	-	-	✓	-	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-
[214]	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	✓	-	-	✓	✓	-	-	-
[241]	-	-	-	-	-	✓	-	✓	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
[246]	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	✓	-	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Total	**1**	**4**	**1**	**6**	**1**	**17**	**3**	**17**	**7**	**1**	**1**	**13**	**3**	**1**	**10**	**9**	**2**	**1**	**2**	**3**	**1**	**4**	**4**	**4**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**1**	**2**	**3**	**3**	**2**	**1**	**3**	**1**	**1**	**1**	**1**	**2**	**1**	**1**

- **A**: Accessibility; **B**: Availability; **C** :Device lifespan Expectancy; **D**: Energy Efficiency; **E**: Installability; **F**: Interoperability; **G**: Maintainability; **H**: Performance; **J**: Reliability; **K**: Responsiveness; **L**: Satisfaction; **M**: Scalability; **O**:Timeliness; **P**: Testability; **Q**: Usability; **R**: Conformance; **S**: Completeness; **T**: Suitability; **U**: Accuracy; **V**: Correctness; **W**: Plausibility; **X**: Robustness; **Y**: Connectivity; **Z**: Compatibility; **AA**:Sensitivity; **AB**: Relevance; **AC**: Stability; **AD**: Mobility; **AE**: Flexibility; **AF**: Extendibility; **AG**: Effectiveness; **AH**: Cost Efficient; **AI**: Portability; **AJ**: Functional Suitability; **AK**: Verifiability; **AL**: Data Integrity; **AM**: Reusability; **AN**:Validity; **AO**: Regulatory Compliance; **AP**:Safety; **AQ**: Integration; **AR**: Resiliency; **AS**: Distributivity; **AT**: Fault-Tolerance; **AU**:Dynamicity

- ✓: Discussed; -: Not discussed.

We identified 47 attributes that drive test objectives. *Interoperability* and *performance* emerged as the two most frequent quality attributes (in seventeen PSs or 20.4%) of IoT systems. *Scalability* (discussed in thirteen PSs or 15.6%) is the third most discussed quality attribute in PSs. *Usability* (in ten PSs or 12.0%) is the next most discussed in PSs. *Conformance* (discussed in nine PSs or 10.8%) is also among the top discussed quality attributes in PSs. *Reliability* is discussed in seven PSs (8.4%). *Resource utilization* or *energy efficiency* is mentioned in six PSs (7.2%). In addition to the attributes listed in Table 4.7, [246] introduces *artificiality*, and *concordance* as quality attributes for IoT systems. Consequently, we observed that testing interoperability and performance are the top test

objectives, followed by testing scalability, usability, conformance, reliability, and energy efficiency.

We observed that quality attributes such as *connectivity*, *device lifespan expectancy*, *distributivity*, *dynamicity*, and *energy efficiency* are required for IoT systems. Traditional software systems may not require these attributes. Attributes such as availability, compatibility, correctness, fault-tolerance, functional suitability, installability, interoperability, maintainability, performance, portability, reliability, resource utilization, reusability, and usability are already defined in ISO/IEC 25010 [2]. In this section, we propose the definitions for attributes not defined in ISO.

- **Accessibility**. The degree to which users can access and use the IoT system effectively.
- **Accuracy**. The accuracy of sensor data is defined as the precision in data measured in terms of the standard deviation of a data value relative to its mean [227].
- **Artificiality** refers to the extent to which sensor data is processed or created. This can include data sourced from a single sensor, aggregated values from multiple sources, or artificially interpolated values generated using algorithms [246].
- **Completeness**. The sensor data completeness refers to the degree to which sensor data values are not missing for a given time window [227].
- **Concordance** refers to a measure used to describe the level of agreement or alignment between the information provided by a specific data source and the information obtained from other independent data sources that report related or correlated effects [246].
- **Conformance**. The degree to which a system adheres to and meets the explicit requirements, specifications, or standards set forth by relevant authorities, industry norms, or regulatory bodies.
- **Connectivity**. As suggested by authors of [175], we can define connectivity in IoT as the system's ability to establish and maintain reliable and efficient connections between its various components and devices.

- **Cost Efficient**. The ability of the system to achieve optimal functionality while minimizing resource usage.
- **Data Integrity**. The degree to which data remains accurate, consistent, and unaltered throughout its lifecycle, safeguarding against errors or corruption.
- **Device Lifespan Expectancy**. As suggested by the authors of [102], we can define IoT device lifespan expectancy as the period during which the device is expected to remain operational and perform its intended functions reliably before it may need replacement or significant maintenance.
- **Distributivity**. Distributivity refers to the characteristic of an IoT system in which its components, located across networked devices, possess the capability to effectively and efficiently communicate, coordinate, and exchange data through message passing. This allows the system to appear to its users as a single, coherent system [132, 177].
- **Dynamicity**. Dynamicity refers to the property of the system's architecture that allows components and connectors to be created, interconnected, or removed during the system's operation. It signifies the system's ability to adapt, evolve, and reconfigure itself in real-time, often in response to changing conditions or requirements, without the need for significant disruptions or downtime [27].
- **Effectiveness**. Effectiveness is a measure of how well the system achieves its outcomes.
- **Energy efficiency**. Energy efficiency is the ability of IoT system to achieve the desired results or functionality using less energy resources [149].
- **Extendibility**. The degree to which a system can be expanded or enhanced to accommodate new features, functionalities, or changes.
- **Flexibility**. The system's capability for devices to undergo changes in both hardware and software configurations, as well as adaptations in their waveform [279].
- **Mobility**. Capability of devices or nodes within the system to move or change location while maintaining seamless connectivity and functionality.

- **Plausibility** refers to a measure that determines whether the information received from a data source aligns with the probabilistic knowledge of what it is supposed to measure, evaluating whether the received data makes sense about its expected purpose [246].
- **Regulatory Compliance**. The extent to which a system adheres to and satisfies the established laws and regulations relevant to its operations.
- **Relevance**. The degree to which the data collected and processed by the system aligns with the intended goals and requirements.
- **Resilience**. The capability of IoT systems to withstand disruptions and crises, recover from emergencies and near-catastrophes, and adapt effectively to a dynamic and ever-changing environment.
- **Robustness**. The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.
- **Satisfaction**. The extent to which user needs are satisfied when the system is used in a particular operation [219].
- **Responsiveness**. The system's ability to handle a request within a specified or required time interval [219].
- **Safety**. The extent to which a system minimizes the risk of harm to users during its operation or in the occurrence of unexpected events.
- **Scalability**. System's ability to handle and accommodate an increasing number of devices efficiently, without compromising performance, reliability, or responsiveness.
- **Sensitivity**. The degree to which the output changes with respect to change in input parameters [280].
- **Stability**. Stability in an IoT system refers to the consistent and reliable output of sensors over time [280].
- **Suitability**. The extent to which a sensor is appropriate or well-suited to fulfill the specific needs and demands of a particular application. The sensor suitability defines

whether the data value expected from the application lies within the measuring interval of the sensor [227].

- **Testability**. The extent to how easy is to design and conduct tests for the system [219].
- **Timeliness**. The ability of the system to deliver and process data at the right time.
- **Validity**. The accuracy and correctness of the data collected by sensors.
- **Verifiability**. System's capability to provide evidence or confirmation that it behaves and performs as specified.

Takeaway for RQ1.1

The top test objectives are interoperability, performance, scalability, conformance, usability, reliability, and energy efficiency. Specific objectives for IoT system testing include connectivity, energy efficiency, device lifespan expectancy, distributivity, and dynamicity.

RQ1.2: What Are Testing Approaches Investigated?

In [49], the authors proposed a framework for developing IoT systems. The framework consists of five layers. The device layer and the application layer require five phases: analysis, design, coding, testing, and installation. The platform layer (cloud layer) encompasses four phases: design, coding, testing, and installation. The infrastructure layer (network) involves the analysis phase. In our study's second and third research questions (RQ2 and RQ3), we used these phases to elucidate the specific phases where the approaches and tools are used.

According to the International Organization for Standardization (ISO), a testing approach "*is a high-level implementation choice of a testing strategy*" [160]. It defines how testing will be carried out and includes various elements, such as level of testing, type of testing, technique of testing, and testing practices. ISO also defines the testing level as "*one of a sequence of test stages, each of which is typically associated with the achievement of*

particular objectives and used to treat particular risks" [160]. The following are common levels of testing: unit/component testing, integration testing, system testing, system integration testing, and acceptance testing. ISO further defines testing type as a specific category of testing focusing on specific quality attributes, such as functional testing, usability testing, and performance testing. Testers can carry out testing activity at a specific level or across multiple levels. For example, conducting performance testing at both the unit test level and the system test level. The testing technique is defined as "*a procedure used to create or select a test model, identify test coverage items, and derive corresponding test cases*"[160]. For example, software testing uses specific test techniques such as equivalence partitioning, boundary value analysis, decision table testing, and branch testing. In this section, we discuss testing approaches (testing levels, testing types, and testing techniques and practices) investigated in PSs.

Table 4.8: Testing Levels

Category 3: Testing Levels			
Name	**Focus**	**Phase**	**Contribution**
Unit Testing [282, 258]	Individual components	C, T	Identification of part of the system causing the failure.
Integration Testing [282, 258]	Component integrations	T	Testing the interface between various IoT components.
System Testing [282]	Entire system	T	Use of simulation tools to simulate the environment.
Acceptance Testing [250, 264, 251]	User-system interaction	T	Use of existing testing tools: Appium, Sikuli, etc.

✺ **C**: Coding; **T**:Testing.

We identified several testing levels in PSs. Table 4.8 provides an overview of these testing levels, which are further explored in this section. Unit testing plays a role in testing IoT systems by isolating individual components, such as sensors, smart objects, communication layer, and application, testers can trace and identify the root causes of system failures [260].

Unlike integration testing in traditional software systems, which focuses on testing the interaction between different modules, integration testing in IoT systems focuses on testing the interfaces between IoT components or layers, such as smart objects (IoT devices) and gateways [282, 258].

System testing is used to *"test the system with its actual operational environment with different scales and scenarios"* [282]. It ensures that the system's functionality and performance meet the desired requirements. System testing includes testing the system as a whole rather than individual components or layers. If the real-world environment is not available, simulation tools can be used to replicate the testing environment.

Acceptance testing, which is user-centric, focuses on evaluating IoT systems from the end-user perspective. It tests the services provided with user involvement [282]. The primary objective is to verify that the system delivers the expected services in the user environment. Analysis of PSs revealed that some approaches involve testing multiple levels [282, 260, 258], while others focus on one level, such as integration testing, which examines the interface between various layers to ensure the proper functioning of IoT systems or acceptance testing [250, 251] for validation.

We also identified several testing types in PSs. Table 4.9 provides the summary of identified testing types.

The analysis revealed few testing types in the PSs. Maintainability testing and portability testing are absent in the PSs. Certain testing types specific to IoT systems, such as connectivity testing and distributivity testing, are also missing. Among the discussed testing types, interoperability testing was the most prominent (six PSs). Conformance testing ranked second (three PSs). Functional testing, performance testing, and scalability testing received attention in two PSs each. Other testing types like compliance testing[2], data integrity testing, compatibility testing, reliability testing, and usability testing are addressed

[2] *Conformance* and *compliance* are often used interchangeably, but they are slightly different. **Conformance** refers to adhering to a specific standard, specification, or set of requirements. **Compliance** goes beyond conformance and involves meeting legal, regulatory, or industry-specific obligations.

Table 4.9: Testing Types

Category 2: Testing Types			
Name	**Focus**	**Phase**	**Contribution**
Performance Testing [260, 258]	Different application domains	Analysis	Testing in health and medical, smart cities, and agriculture domains. [258]
Functional Testing[247], [220]	Device testing	Testing	FSM-based test cases [247] Compatibility testing tool [220]
Conformance Testing [236, 283]	Client environment dependencies	Analysis	Testing interface between components [236] ML-based detection for reactive system failures [283]
Scalability Testing [260, 242]	Adapter for protocols	Design	Validation and verification based on test report
Compliance Testing [224]	OneM2M library	Design	Inter-component communications
Interoperability Testing [291, 267, 218, 248, 241, 246]	Integration	Analysis Design	Testing interface between components Compatibility among packets and input used
Data Integrity Testing [260]	Definitions	NP	Data integrity in IoT systems
Compatibility Testing[260]	Definitions	NP	Compatibility in IoT systems
Reliability Testing[260]	Definitions	NP	Reliability in IoT test environment
Usability Testing[260]	Definitions	NP	Usability of IoT systems for different users

✵ **NP**: Not Provided.

in one PS each.

Finally, we identified many testing techniques considered for IoT systems. Table 4.10 shows the summary of identified testing techniques. The authors of [55] proposed a list of software testing techniques that can be leveraged for testing the application layer of IoT systems. The author of [69] presented different testing techniques specifically tailored for IoT systems. Automation testing is the most researched technique (6 PSs), but full automation of IoT systems testing remains elusive and debatable. IoT system test automation currently focuses on semi-automation, where some layers are tested manually. Three PSs discussed fault testing for anomaly detection. Three other PSs discussed fuzzy testing, which involves generating large volumes of random test data. Two PSs covered

pattern-based testing, two PSs addressed standard-based testing, two PSs explored verification and validation, two PSs examined simulation-based testing, and two PSs investigated coverage-based testing. We identified one PS for each of the following testing categories: formal verification, continuous testing, requirement-based testing, data validation testing, log analysis, and knowledge-based testing. We considered *fuzzy testing, automated testing, and model-based testing* as testing practices [69], which are also part of testing approaches.

Table 4.10: Testing Techniques and Best Practices

Category 1: Testing Techniques and Practices				
Name	**Focus**	**Phase**	**Strengths**	**Weaknesses**
Fuzzy Testing [278, 252, 266]	Network protocol	Design, Coding	Unlocking the potential of AI algorithms in testing	Not suitable for E2E testing
Data Validation Testing [213]	Data validity testing	Testing	It supports abstract-level testing	Not suitable for complex scenario testing
Log Analysis [281]	Bug detection	Design	It supports log mining	Not suitable for cloud-based applications
Knowledge-Based Testing [274]	Knowledge Mining	Design	Leveraging ML algorithms to improve IoT systems testing	Needs a lot of processing resources
Pattern-Based Testing [268, 285]	Event pattern identification	Design	Generic and validated IoT test patterns	Not suitable for fuzzy logical systems
Coverage-Based Testing [235, 291]	Smart contracts	Design	It offers Testing as a Service	Limited to commercial apps only
Simulation-Based Testing [225, 216]	Devices	Coding	No requirements for real devices	Requires real-time user interaction
Fault Testing [263, 212]	Anomaly detection	Testing	Using ML algorithms for complex cases	Needs a lot of processing resources
Automated Testing [236, 247, 271]	Network and devices	Design	It uses an intermediary model to validate IoT services	Not suitable for E2E testing
Verification and Validation[267, 244, 214]	Scenario analysis	Analysis	It supports textual notation and UML modelling	Requires clear criteria for test cases
Requirement-Based Testing [269]	Network	Analysis	Experimental results	Not suitable if requirements are not clear
Standard-Based Testing [239, 245]	Network Protocols	Analysis	Implementation-Independent	Not suitable if requirements are not clear
Continuous Testing [231]	DevOps	All	It proposed best practices based on open-source tools	Suitable for simple scenarios only
Formal Verification and Model-Based Testing [244]	Formal verification and MBT	Design	It supports abstract-level testing	Supports only partial user scenario testing due to resource constraints
Model-Based Testing[244, 212, 283]	Testing as a service	Design	It enables testing IoT data and platforms	Limited use case scenarios

- **MBT**: Model-Based Testing; **E2E**: End-to-End.
- **Level of Automation**: None of the proposed techniques or approaches are fully automated.
- **Targeted Application Domain**: Most of the approaches and techniques proposed in PSs are generic and can be applied to various IoT systems.

Takeaway for RQ1.2

Approaches for interoperability testing are prominent, followed by conformance and functional testing. Automated testing is well-researched, but there are no fully automated approaches.

RQ1.3: What Are Testing Tools Investigated?

Testing tools in software engineering are software applications that automate or facilitate the testing process. The purpose of these tools is to help the testers verify various aspects of IoT systems, such as the functional correctness and performance of IoT systems. They can simulate various scenarios and environments to validate the behavior of IoT systems under different conditions. Testing tools help to ensure the reliability and quality of IoT systems. We analyzed PSs to identify the tools proposed for testing IoT systems. We identified two categories of tools: *test execution tools* and *test environment tools (or testbeds)*. Test execution tools help to execute test cases, while test environment tools provide the environment for testing.

Below we report the summary of test execution tools in PSs.Several tools have been identified for testing IoT systems, covering different layers, methodologies, and testing levels across the software development lifecycle.

- **Héctor** [215] – A Python-based tool designed for testing IoT devices and networks. It employs a white-box testing approach and utilizes REST APIs for integration and system-level testing during the testing phase.
- **Node-RED** [264] – An open-source tool written in Node.js, primarily targeting application-layer IoT testing. It follows a black-box testing approach and supports REST APIs for acceptance and system-level testing, mainly used in the design phase.
- **Apache JMeter** [264] – A Java-based tool focused on load and performance testing of IoT applications. It supports both SOAP and REST APIs for unit testing, making it

particularly useful during the coding phase.

- **Eclipse IoT-Testware** [239] – A Java-based framework supporting multiple IoT layers. It offers both white-box and black-box testing capabilities and is commonly used in the design phase for testing IoT applications.
- **PatrIoT** [218] – A Java-based tool that focuses on distributed IoT applications. It primarily tests IoT devices and networks using REST APIs for integration testing in the testing phase.
- **Tsung** [226] – An Erlang-based tool designed for stress testing IoT systems. It supports SOAP-based API testing and is widely used during the testing phase.
- **SemTest** [223] – A tool used for semantic compliance and interoperability testing of IoT systems. It uses a black-box testing approach and is applied during the testing phase.
- **MATTER** [264] – A framework that supports executable test cases for IoT applications. Implemented in Python and Java, it follows a black-box testing methodology and is mainly used in the design phase.
- **F-Interop** [291] – An IoT testing platform designed for device-layer interoperability and performance testing. It follows a black-box approach, supports REST and XML-based Rspec APIs, and is utilized for system and acceptance testing in the testing phase.
- **IoT-TaaS** [242] – Provides Testing-as-a-Service (TaaS) for IoT devices. It uses a black-box testing approach and is primarily used for integration testing in the testing phase.
- **Izinto** [268] – A pattern-based test automation framework written in Java and Node.js. It is used for testing known patterns only.
- **ICAT** [220] – A black-box testing tool designed for IoT device compatibility testing. It is mainly utilized in the testing phase.
- **Robot Framework** [220] – A Python-based tool widely used for automated testing of IoT applications. It follows a black-box testing approach and supports REST APIs for acceptance and system-level testing in the testing phase.

- **Selenium** [231] – A popular testing framework that supports multiple programming languages, including Java, JavaScript, Python, Ruby, and C#. It is used for black-box testing of IoT applications, specifically for acceptance and system-level testing.
- **IoT-TEG** [287] – A testing tool for event generation in IoT systems. Implemented in EPL and Java 8, it follows a black-box approach and supports REST APIs for unit testing.
- **Smartesting CertifyIt** [212] – A model-based testing tool for enterprise IT applications. It uses UML, OCL, and BPMN for designing and validating IoT test cases.
- **Stryker Mutator** [249] – A mutation testing tool that supports JavaScript, TypeScript, C#, and Scala. It follows a black-box testing and is primarily used in the design phase.
- **IoT-CIRTF** [259] – A Java and Python-based tool designed for test case prioritization and selection in IoT device testing. It supports both white-box and black-box testing approaches and is primarily used for integration testing in the coding phase.
- **VectorCAST** [260] – An embedded systems testing tool that supports C and C++. It is designed for testing IoT devices and applications using both white-box and black-box methodologies. It is applied at unit and integration testing levels in the coding phase.

Although we identified several tools, their application to IoT systems testing is limited to a few qualities, such as scalability and connectivity. Our findings emphasize the need for more comprehensive testing tools that encompass a wider range of qualities of IoT systems. Several testbeds support multiple programming languages ([261, 209, 277, 211, 284, 286, 273, 276]), with Java being the most popular (used in 8 testbeds), followed by C, C++, and Python (used in 3 testbeds). *Testbeds As A Service* (TaaS) ([253, 284, 234, 240]) offer cost-effective testing services. It enables users to request testbeds and necessary resources, submit testing details, and download testing reports upon completion, leading to shortened test environment setup time. While most testbeds focus on device and network layers ([253, 228, 284, 234, 215]), some testbeds support three layers, including device, cloud, and application layers ([261, 209, 240]). ContikiOS is commonly used in testbeds

([234, 228, 276]), followed by Kafka ([261, 215]). One PS reported the use of a proprietary graphical operating system for users to access and configure IoT devices through a web-based interface ([209]). We did not find hardware and software specifications of certain testbeds, particularly for cloud-based testbed ([277, 240]) or for on-premise installation testbed ([262, 286, 272]).

For improved clarity and ease of replication, we summarize the testbed identified below. Based on their characteristics, the IoT testbeds can be classified into different categories as follows:

- **Large-Scale IoT Testbeds**
  - **Two-Tier Fog Testbed** – Handles streaming data efficiently with high availability.
  - **FIESTA-IoT** – Fault-tolerant, supports federated experiments.
  - **FIT IoT-Lab** – Large-scale open-access IoT testbed with wireless IoT device support.
  - **HATBED** – Affordable, remote debugging, and software profiling.
  - **IoT Bed** – Cost-effective testbed for users.
  - **JOSE** – Quick setup time, supports various IoT layers.
  - **LinkLab** – Scalable, multi-user, multi-site testbed with remote development.
  - **Smart Santander** – Smart city deployment with guaranteed dependability.
  - **UiTIoT** – Focuses on reliability, availability, and effectiveness.
- **Small-Scale IoT Testbeds**
  - **AssIUT IoT** – Enables user access and configuration of IoT nodes through GUI.
  - **FogTestBed** – Allows users to request resources and submit experiments online.
  - **IoTier** – Supports easy integration with container orchestrators.
  - **LocURa4IoT** – Designed for flexible indoor localization experiments.
  - **EAWN Testbed** – Focuses on energy-aware protocol testing in real devices.
  - **SD IoT Testbed [276]** – Provides a simulation environment for real-world testing.
- **Cloud-Based Testbeds**

  - **Two-Tier Fog Testbed** – Uses Kafka, Hadoop, and Spark for cloud-based analytics.
  - **FIESTA-IoT** – Cloud-based experimental environment.
  - **FIT IoT-Lab** – Supports embedded OS with cloud-based access.
  - **AssIUT IoT** – Uses Windows Server, IIS, ASP.NET, and MS SQL.
  - **LinkLab** – Cloud-based on *Alibaba Cloud*.
  - **Smart Santander** – Cloud-based IoT infrastructure.
  - **SD IoT Testbed** – Supports various IoT OS like ContikiOS, TinyOS, and FreeRTOS.
  - **UiTIoT** – Uses OpenStack, DeltaQ, and QOMET for cloud computing.
- **On-Premise Testbeds**
  - **HATBED** – Supports non-invasive software profiling and remote debugging.
  - **IoTier** – Focuses on containerized deployments.
  - **LocURa4IoT** – Indoor localization with on-premise setup.
  - **EAWN Testbed** – Energy-aware protocol testing in real devices.
  - **JOSE** – Supports Java, C, and JavaScript with an on-premise deployment.
- **Smart City & Industrial IoT Testbeds**
  - **Two-Tier Fog Testbed** – Smart city applications with edge computing.
  - **FIESTA-IoT** – Smart city experiments.
  - **Smart Santander** – Extensive *smart city* testbed with sensor networks.
  - **FIT IoT-Lab** – Used in *smart factory* settings.
- **Energy & Environment Monitoring Testbeds**
  - **EAWN Testbed** – Energy-aware testing of IoT protocols.
  - **SD IoT Testbed** – Supports energy-aware sensors (RPi, Intel Galileo, BME280).
  - **Smart Santander** – Energy-efficient sensor nodes and communication interfaces.

Our analysis revealed that existing testbeds mainly focus on specific layers of IoT systems, particularly network (13 PSs) and device layers (11 PSs). Most discussed testbeds are cloud-based (10 PSs), providing scalability, flexibility, and cost efficiency. There

is a significant emphasis on IoT testbeds for smart cities, with five testbeds identified [261, 277, 286, 272, 273]. Seven testbeds are generic and applicable to any application domain.

The lack of hardware and software specifications for certain testbeds, particularly cloud-based and on-premises installations, may hinder transparency and replication of experiments. Existing testbeds mainly focus on specific IoT layers, such as network and device layers, potentially leaving other layers underrepresented in testing. Cloud-based testbeds dominate the landscape, favored for their scalability, flexibility, and cost efficiency. Moreover, the emphasis on IoT testbeds for smart cities may highlight the significance of urban applications. However, the relatively small number of generic testbeds may indicate a need for more versatile solutions applicable across various IoT domains. Addressing these implications could enhance the effectiveness and inclusivity of IoT systems testing efforts.

Takeaway for RQ1.3

Several tools for traditional software testing, such as Selenium, Robot Framework, Apache JMeter, and Stryker Mutator, are used in IoT systems testing. Specific tools to test IoT systems, such as PatrIoT, ICAT, IoT-CIRT, and Héctor, are found in PSs.

RQ1.4: What Are IoT Systems Used For Evaluation?

We compiled a catalog of diverse IoT systems used to evaluate various approaches proposed in PSs. Each IoT system has a distinct focus and may include different components, covering areas like Ambient Assisted Living (AAL), smart street systems, smart parking, smart manufacturing, smart agriculture, and more. Some are mobile apps for Android and iOS, while others provide data through web interfaces. The source code for the evaluated systems is not publicly available, except for *DiaMH*. The following IoT systems have been used in evaluation studies, focusing on different applications and domains.

• **DiaMH** [250, 264, 249, 251] – A Diabetes Mobile Health IoT system that monitors

glucose levels, alerts patients and doctors when values exceed a threshold, and regulates insulin dosing. It includes wearable glucose sensors and insulin pumps connected to smartphones, which serve as intermediaries to a cloud-based healthcare system for data processing and insulin dosage recommendations.

- **Care Receiver Monitor in AAL** [268] – A monitoring system where a caregiver observes a care receiver in a controlled environment. Various sensors collect health and environmental data, which is sent to a central server that maintains health records and triggers predefined actions and alerts.
- **4 Android apps, 4 iOS apps** [220] – A scenario involving one cloud server, one Wi-Fi access point, eight app versions (two Android apps on two Android OS versions and two iOS apps on two iOS versions), four gateway versions, and six sensor versions.
- **Smart Street System, AMQ Online Product** [218] – A smart street system using an active messaging queue (AMQ Online) to enhance urban living through smart lighting, traffic monitoring, and environmental sensors.
- **GSM** [259] – A system providing efficient IoT device connectivity. It includes two datasets: one with 80 requirements and 100 test cases for IoT device connection efficiency and another for Mobile IoT (MIoT) with 51 requirements and 41 test cases.
- **Sensing App** [270] – A simple IoT system using temperature and light sensors to generate small amounts of data.
- **Smart Parking IoT System, Smart Manufacturing System** [235] – A smart parking system managing 1840 devices to optimize parking space allocation. The smart manufacturing system integrates 161 devices, including sensors and cameras, to monitor the manufacturing environment, with remote access to data through web servers.
- **Smart Agriculture Applications** [227] – Uses an IoT setup with an LM35 temperature sensor, an Arduino UNO, and an internet-connected laptop. Data is transmitted to ThingSpeak via Arduino and analyzed in real-time using MATLAB.

- **Temperature Monitoring System** [244] – A system with four sensors measuring ambient temperature, storing the data in a database for future analysis.
- **Smart Home** [263] – A simulated smart home integrating 11 apps that automate 17 IoT devices, including:
  - **Motion-Activated Lights** – Turns lights on when motion is detected and off when no motion is present.
  - **Smoke Alarm** – Sounds an alarm and unlocks doors when smoke is detected.
  - **Temperature Control** – Maintains room temperature between 70-80 degrees by controlling heating and cooling.
  - **Water Leak Detector** – Sounds an alarm and closes the water valve if leak is detected.
  - **Welcome Home** – Unlocks doors and activates the coffee machine when user arrives.
  - **Secure Patio** – Sends a text message if unexpected movement is detected.
  - **Energy Saver** – Closes windows if the heater or air conditioner is running.
  - **Secure Home** – Locks doors and closes windows when the user is away.
  - **Intruder Detector** – Sends a notification if suspecious movement is detected.
  - **Alarm Safety** – Turns on lights when an alarm is triggered.
  - **Morning Air** – Opens and closes windows at predefined times.
- **SeRGIo** [216] – A mobile sensing system designed for geospatial IoT applications through participatory sensing, involving academic and industrial partners for data collection.
- **Smart Mobility** [214] – An IoT system assessing pollution levels on urban cycling routes using smart bicycles, a central data server, and smart poles.
- **ODAA (Open Data Aarhus)** [246] – A city-wide open data portal, providing access to datasets such as traffic data from 449 sensors reporting vehicle counts and speeds.
- **Smart Parking Application** [261] – A smart parking system using IoT to efficiently manage parking spaces, reduce congestion, and help users locate available spots.

We noticed that some authors did not specify the name of the system they used in their

experiment or did not make their source code publicly available. They simply mentioned that they conducted experiments, without further details. The absence of such details may impact result reproducibility and potentially hinder the progress of other researchers. We believe that sharing this information can be beneficial to other researchers in the field of IoT systems testing.

Takeaway for RQ1.4

Several IoT systems have been used to evaluate the proposed approaches. Most of these systems are not publicly available, thus affecting the reproducibility of the results reported in PSs.

4.3.2 RQ2:What challenges do researchers report in IoT system testing?

RQ2.1: What Are Testing Challenges Identified?

We identified the following testing challenges in the PSs.

- *Large number of heterogeneous devices* [235, 260]. If IoT systems have hundreds, or even thousands, of heterogeneous devices, a challenge is identifying which devices should be tested to detect faults. Another challenge is testing *different communication protocols*. Some devices face power issues because the *availability of energy and network* cannot be guaranteed [256].
- IoT devices generate *data in different formats*, making it difficult to create a universal method for collecting and analyzing data from devices [260]. The *lack of APIs for some IoT devices* can also pose a challenge for collecting and analyzing data in IoT systems.
- Access to *real devices* to test IoT systems, reproducing IoT bugs, fault localization, and testing *diverse technologies* are also challenges [254, 260].
- *Limitations of existing tools, and approaches*, in testing aspects of IoT systems [225].
- The *tight coupling between hardware and software* creates a testing challenge, as defects

in either component can impact the overall functionality of IoT systems [260].

- Testing *non-standard compliant devices* is hard because they do not adhere to standard protocols and specifications [239], and cannot be tested in standardized scenarios.
- *Testing in real-world scenarios* with the deployment of real devices is costly [225, 288].
- *Lack of testing methodologies* introduces a challenge to compare different IoT devices or to develop testing frameworks that apply to all IoT devices [288].
- Testing IoT is challenging when the devices behave differently in *non-repeatable scenarios* because of environmental changes or unpredictable device behavior, leading to unpredictable results and difficulty in identifying issues [288].
- *Lack of uniform communication interface and synchronization* in a hybrid environment leads to another challenge. The system under test cannot distinguish between simulated and real-life behavior. Synchronization of testing involves combining data generated by simulated entities and data generated by real entities [216].

Takeaway for RQ2.1

Testing IoT systems has several challenges, including device heterogeneity, difficulties in collecting and analyzing diverse data, tight coupling between hardware and software, non-standard compliant IoT devices, costly real-world and non-repeatable scenarios, and limitations of existing approaches and tools.

RQ2.2: How Are Testing Challenges Addressed?

We reviewed the PSs to understand how they addressed IoT testing challenges. Some of the challenges identified can be addressed by some solutions proposed in the PSs. There is no solution proposed to tackle challenges such as testing in real-world scenarios, non-repeatable scenarios, unavailability of devices due to lack of energy or network, lack of APIs for some IoT devices, testing diverse technologies, and tight coupling between hardware and software. We highlighted the addressed test objectives putting the most addressed

challenges at the top, and the ones that remain mostly unaddressed at the bottom. Several challenges in IoT system testing have been identified, and various solutions have been proposed in primary studies (PSs) to address them.

- **Large number of heterogeneous devices** [235, 260] – Authors in [251] proposed an acceptance testing approach using a UI at the system level based on test scenarios. They assumed that if the system provides the expected functionality at the system level, then the devices should work as expected. In [235], a combinatorial testing path selection framework for IoT systems, called CT-IoT, was proposed to systematically identify and recommend testing paths for effective IoT system testing. Additionally, [287] introduced a solution based on a test event generator to test multiple connected IoT devices, making decisions based on real and complex data.
- **Different communication protocols** – Authors in [218] proposed an open architecture testing framework that allows the integration of various devices by adding communication adapters between test cases and devices. They also suggested introducing configurable simulated devices, composed of predefined building blocks, to minimize challenges posed by different communication protocols.
- **Limitation of existing tools, approaches, or lack of testing methodologies** [225, 288] – Some studies have suggested online solutions such as testing-as-a-service, while others have proposed specific tools and frameworks for IoT testing. [241] discussed how F-Interop can provide an extensive experimental platform for IoT systems, offering online interoperability, conformance, and performance testing. Authors in [236] proposed an automated and scalable online conformance testing approach for IoT applications. Additionally, various tools [250, 264, 249, 251] and frameworks [268, 218] have been introduced to enhance IoT testing.
- **Non-standard compliant devices and different data formats** [239, 260, 239] suggested

the use of Eclipse IoT-Testware, an open-source standards-based solution, to ensure protocol conformance, robustness, and security. Other studies [236, 224, 242] recommended using oneM2M for IoT conformance testing. This approach involves standardized conformance testing mechanisms, automated and scalable conformance verification, and test triggering mechanisms based on standardized test interfaces.

- **Lack of a uniform communication interface and synchronization in a hybrid environment** [216] – Authors in [216] proposed an AI-powered proxy-based synchronization mechanism that operates over multiple dimensions, including space and time. Within the space dimension, the approach ensures consistency between data sensed at a specific location, while in the time dimension, it maintains consistency between data sensed at a specific time. To address synchronization issues, the study suggested (a) proxy-based synchronization to intercept and forward messages between real and virtual environments and (b) space isolation to reduce synchronization requirements by isolating specific geographic zones for either real or simulated local entities.
- **Availability of real devices and testing in real-world scenarios** [254, 260, 225, 288] – While no specific solution was proposed in PSs for real-world scenario-related challenges, [234] introduced IoTbed, a testbed that provides IoT devices on-demand, enabling users to rent devices as needed. The model offers monetary incentives to device owners, encouraging them to make their devices available for testing.
- **Non-repeatable scenarios** [288] – No explicit solutions were found for non-repeatable scenarios caused by IoT system dynamics. However, some studies [212, 283] proposed model-based testing, while others [238] introduced runtime verification of IoT systems using complex event processing.
- **Availability issues due to energy and network constraints** [256] – No PS specifically addressed the energy-related availability challenges of IoT devices. However, for connectivity issues, [218] suggested a solution for testing the connectivity of various devices.

- **Lack of APIs for IoT devices** – No PS has proposed a solution for the absence of APIs in IoT devices.
- **Testing diverse technologies** [254, 260] – No PS has discussed a solution for handling diverse IoT technologies in testing.
- **Tight coupling between hardware and software** [260] – No PS has explored solutions for addressing the tight coupling between hardware and software in IoT systems.

Takeaway for RQ2.2

No specific PS focused on a particular challenge for testing IoT systems. However, we found that solutions proposed in certain PSs can partially or fully address some of the identified challenges.

4.4 Discussions

4.4.1 Results Overview

This section discusses the results and explores the implications of the lessons learned.

- Our findings indicate that researchers have devoted less attention to specific objectives of testing IoT systems such as testing connectivity, resource usage, device life expectancy, correctness, and timeliness. Objectives such as testing interoperability, compatibility, performance, reliability, scalability, verification, data integrity, data validity, usability, regulatory compliance, conformance, integration, and fault tolerance, can be associated with testing approaches. Although some PSs discussed other objectives, they did not mention any specific approach to test them. We observed that functional suitability testing was the only objective assessed through various levels of testing (unit testing, integration testing, system testing, and acceptance testing); however, PSs did not discuss the testing levels for other objectives. We observed that only 14 out of the 47 quality attributes are associated with the testing approaches and techniques discussed in the PSs.

We could not find a reason why other quality attributes are not associated with testing approaches. However, we believe that there could be three possible reasons. One possible reason could be the rapid growth of the IoT domain. Keeping up with this rapid growth can make it difficult for researchers to focus on all testing objectives simultaneously. It is possible to expect other studies to focus on techniques and approaches for other objectives. Another reason could be the researchers' priorities. If specific objectives are considered more critical by researchers, testing studies may be biased toward these objectives. A last possible reason is that some of these objectives pertained to traditional software systems and can still be applied in the same way to IoT systems. Our findings highlighted research opportunities to explore and develop testing approaches for the overlooked objectives.

- Testing approaches encompass testing levels, testing types, testing techniques, and industry practices. We identified that all the levels of testing found in traditional software systems apply to IoT systems, although the context may exhibit slight variations. In IoT systems, unit testing may also involve the testing of each IoT layer in isolation. This shows the distinct nature of this approach in the context of IoT. Comparing the recommended testing types, techniques, and practices for IoT systems [69], we can suggest that further exploration is necessary to investigate how to apply these types, techniques, and approaches to IoT systems.
- While observing various testing tools, we found that some are tailored to different levels of testing. Nevertheless, when considering testing types, techniques, and industry practices, the availability of such tools is relatively scarce. Among the observed testing tools, Selenium, Robot Framework, and Apache JMeter, widely used in traditional software systems have been adopted to test the application layer of IoT systems. In terms of testbeds, many of them are used for testing the device layer and network layer. Conversely, test-environment tools for the application layer are scarce.

- Many challenges identified are specifically related to testing the device layer. The lack of standards could be one of the causes of these challenges. If IoT devices adhere to the same standards, it would address various issues such as different data formats, device APIs, communication interfaces, etc.

4.4.2 Implications for IoT Researchers

The lessons learned from this chapter indicate that testing IoT systems is a new and growing field. The number of PSs has been growing since 2013, with most of them being published in 2018. However, the number of PSs has been decreasing since then. This decrease could be attributed to the effects of COVID-19, making it difficult for researchers to conduct some experiments, especially in fields such as agriculture or transport. Most active researchers in PSs are industry-based rather than academia. The industry has more resources available for research activities related to IoT systems. To ensure that their IoT systems are reliable, secure, and meet customer expectations, the industry can be motivated by commercial pressures.

PSs predominantly emphasized device testing over other layers of IoT systems. Experiments are the primary evaluation method used in IoT systems testing studies. PSs that used experiments provided understandable results supported by factual evidence. PSs highlight various quality attributes in IoT systems that are also used in traditional software systems. PSs also report new quality attributes for IoT systems such as connectivity, energy efficiency, device lifespan expectancy, distributivity, and dynamicity. Traditional software testing approaches are used to test some quality attributes of IoT systems, with interoperability, conformance, and functional testing being prominent. Additionally, testing tools for testing traditional software systems, are used in IoT systems testing. However, testing IoT systems still poses certain challenges, offering new avenues for exploration.

4.4.3 Implications for IoT Practitioners

PSs reported that IoT systems testing focuses more on interoperability, performance, scalability, conformance, usability, reliability, and resource utilization. Moreover, on top of traditional quality attributes [2], PSs also reported new quality attributes like connectivity, energy efficiency, device lifespan expectancy, distributivity, and dynamicity. Practitioners can use traditional software testing approaches and tools to test those attributes common to traditional software systems. However, future research may develop specialized approaches and tools tailored to IoT systems testing, addressing the unique attributes of these systems.

4.4.4 Limitations of this chapter

There are several limitations to our study. We focused on the testing objectives, approaches, tools, and challenges discussed in the published studies between 2012 and 2022. We acknowledge that IoT systems testing is extensive and evolving. Many other testing approaches (testing types, techniques, and practices) may exist beyond the ones discussed in PSs. We did not address security testing approaches and tools in our research, as we decided to study security testing as a separate topic. Additionally, the tools we identified do not include emulators and simulators, as other studies [148, 29] have focused on them.

4.4.5 Discovering Connections and Key Observations

Figure 4.4 connects our results. The key observations are:

- Out of the 47 possible objectives identified in PSs, only 14 objectives are addressed by the testing approaches found in PSs. Among the remaining 33 objectives, there is no discussion in PSs on how to address them for IoT systems. We found no information explaining how the proposed testing approaches and tools achieved the remaining test objectives. Although many of them can be handled similarly to traditional systems, specific aspects like connectivity, distributivity, dynamicity, and device life expectancy need attention, as they are unique to IoT systems and not be addressed by traditional

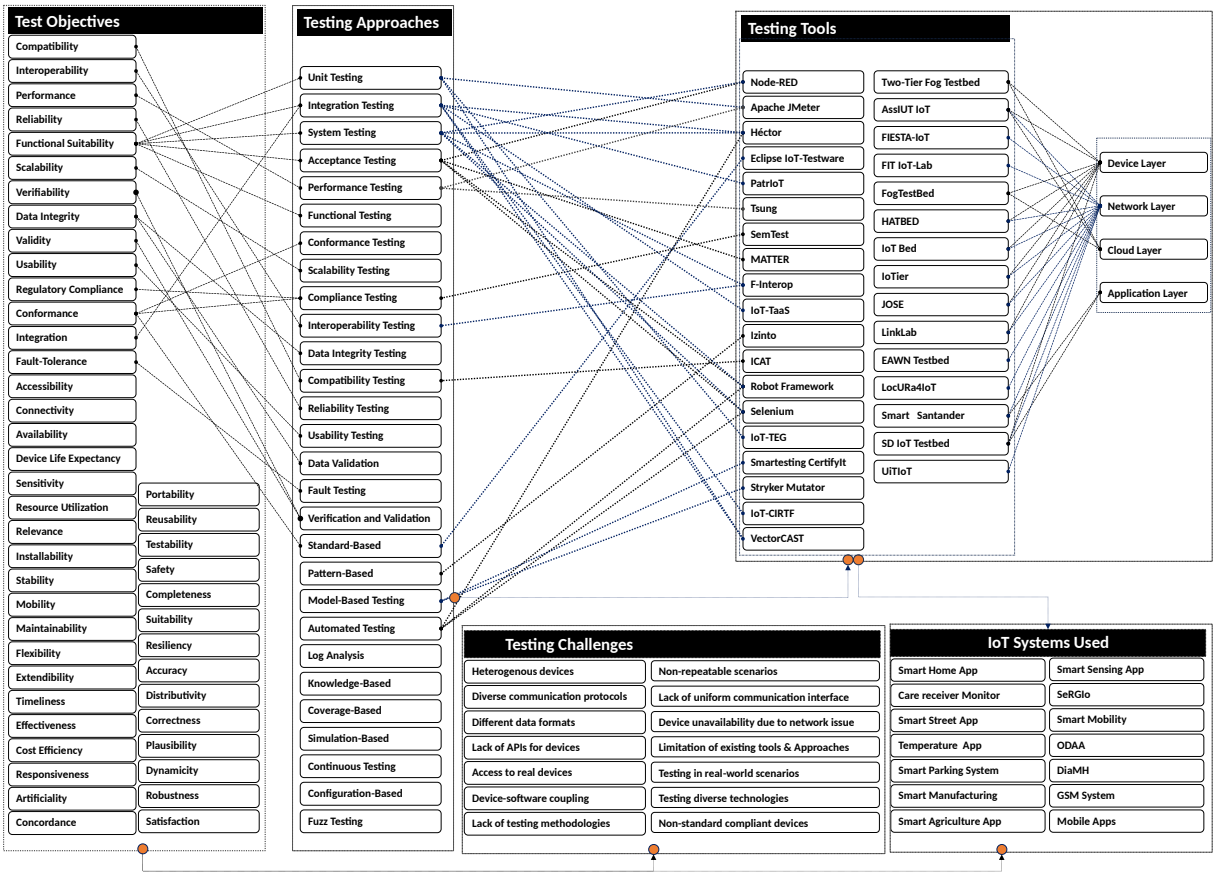


Figure 4.4: Results Overview

approaches.

- Connectivity is crucial for any IoT system. Testing for connectivity and distributivity should be regarded as essential testing types for IoT systems; however, they are not present in PSs. Moreover, for certain testing types like scalability, no tools are found in PSs, despite their importance, given the potential expansion of IoT systems with the addition of new devices.
- IoT-specific tools are proposed either on-premises or offered as a service in the cloud. Additionally, tools commonly used in traditional software systems, such as Selenium, Robot Framework, and Apache JMeter, are of equal importance for testing IoT systems. We believe that IoT system testers can also leverage other tools like WebdriveIO or Appium for automated testing of web or mobile user interfaces. In Figure 4.4, each

challenge is presented within its box, except for the limitation of existing tools and approaches, which are grouped in one box to reduce the size of Figure 4.4, resulting in a total of 14 boxes for the challenges. The limitations of existing tools, approaches, and lack of testing methodologies are grouped in one entry. In the same table, access to real devices and testing in real-world scenarios are also combined as one entry. Additionally, non-standard compliant devices and different data formats are grouped. Individual studies may not have fully addressed specific challenges. However, various PSs discussed approaches that could potentially resolve these challenges. We summarized the proposed approaches found in the PSs that could be used to address some of these challenges.

- Many IoT systems have been proposed to evaluate various approaches, but most of these systems are neither available nor accessible online.
- Some authors did not disclose the name of the system they used in their evaluation. For example, in [220], the authors provided descriptions for various *mobile apps* without specifying their names. We highlighted this in Figure 4.4 because it is important, even though no specific name was provided.

4.5 Threats to Validity

There are threats to the construct validity, internal validity, external validity, and conclusion of this chapter.

- **Construct Validity**. Construct validity relates to sources investigated and data collection. It pertains to the selection of PSs and how we extracted data from these PSs in relation to the RQs.
  - *Missing relevant studies*. Incompleteness is a threat. Our search relied on titles and keywords, which may have resulted in the omission of relevant studies. The accuracy of our search depends on how well digital libraries organize and categorize papers. To mitigate this threat, we used seven databases, which are reputable and most comprehensive digital libraries for literature reviews [44]. We used two rounds of snowballing

to find more potentially relevant studies.

  - *PSs selection bias*. During the selection of PSs, we may have excluded relevant studies. The subjective nature of manually conducting the selection process could contribute to this biased selection. To mitigate this selection bias, we defined the purpose of the study and the research questions in advance, following the PRISMA guideline, and established clear inclusion and exclusion criteria.
  - *Data extraction bias*. The data extraction process involved extensive manual efforts, which may be subject to personal bias. To minimize this bias, we defined the data collection form. Two authors followed the agreed form and independently conducted the data extraction. We used interrater agreement and several discussion sessions involving all authors to reach a common agreement. Another threat arises from the inconsistent terminologies in PSs. We discussed each of these inconsistencies and agreed on a unified vocabulary.

- **Internal Validity**. Internal validity concerns the methods used in the study and related conclusions.
  - *Scope of the review*. The research questions do not cover all aspects of IoT systems testing. We may have missed some test objectives that are not based on quality attributes. We missed any test objective, test approach, and tool related to the security testing of IoT systems. We recommended other studies that focused on other aspects of IoT systems testing not covered in this chapter.
  - *Completeness of the review*. Our study focuses on IoT systems testing objectives, approaches, tools, and challenges. Our PSs may not provide sufficient details on all these aspects. To minimize this threat, we included only relevant PSs that are specific to each research question.
  - *Method of the review*. The methodology to conduct research may introduce various

threats, including the potential for bias, which can undermine the reliability and validity of the findings. To mitigate this threat, we followed the updated PRISMA guidelines and carefully selected relevant studies while removing irrelevant ones. All authors defined and reviewed the inclusion/exclusion criteria.

- **External validity**. External validity refers to the generalizability of our findings to all IoT systems testing. This review focuses exclusively on academic research. Industry practices may not be included if they were not reported in academic publications. We reviewed studies published within a specific timeframe, which may limit the extent to which the findings can be generalized. However, we consider this review to be valuable for both academia and industry practitioners, and we will conduct an industrial study to complement it.
- **Conclusion validity** concerns the degree to which the conclusions drawn from the data extracted are reasonable and valid. We ensured the validity of our conclusions by carefully analyzing the data extracted from PSs. To enhance validity, we conducted multiple discussion sessions wherein we collectively drew and cross-verified our conclusions against the extracted data. Therefore, our conclusions solely rely on the findings obtained from PSs.

4.6 Conclusion

This chapter aimed to understand how IoT systems are tested in terms of objectives, approaches, tools, and challenges from the literature. We conducted a detailed review of 83 PSs, compiling 47 quality attributes, with 5 of them specifically related to IoT systems, helping us understand the objectives of testing IoT systems. We provided an overview of testing approaches including 4 testing levels, 15 testing types, 15 testing techniques and practices for IoT systems. We presented the compilation of 19 user testing tools and 15 testbeds for IoT systems testing, highlighting their usage at different stages of IoT systems development. We summarized the challenges encountered in IoT systems testing and

highlighted potential research directions to effectively address the emerging and futuristic challenges associated with testing IoT systems. Our study has the following implications for researchers and practitioners:

- It highlights the objectives (based on quality attributes), approaches, tools, and challenges of IoT systems testing. Furthermore, it identifies avenues for further research due to the limitations of existing approaches and tools, challenges faced by testers, and untested quality attributes.
- It serves as a guide for practitioners to understand different testing approaches and tools.

The findings of this chapter have been published in Transactions on Software Engineering [128]. In the next chapter, we shift our focus to understanding IoT system testing from the perspective of industry practitioners.

Chapter 5

Industry Study on IoT Systems Testing

5.1 Introduction

Building on the previous chapter, where we investigated IoT system testing through a systematic literature review (SLR), this chapter investigates the state of IoT systems testing from an industry perspective. We explore the challenges, tools, approaches, and artifacts reported by practitioners to understand how IoT testing is conducted in real-world settings. Despite several studies providing state-of-the-art IoT testing approaches in academic literature, they may not fully capture the industry's perspective. While researchers may focus more on the theoretical foundation of IoT systems testing, practitioners bring hands-on expertise. Their different perspectives complement each other and help bridge the gap between theory and practice [62]. From our previous chapter, we identified only a few reviews focusing on the industry perspective, but their focus is limited to either testing challenges or testing methods. Some of these reviews were conducted several years ago. However, the field of IoT is dynamic. The technology, methodologies, and challenges associated with IoT systems testing are evolving. Therefore, older reviews may not capture the latest advancements, emerging trends, and current practices in the field. Understanding the main challenges IoT practitioners face when testing IoT systems can provide invaluable

orientation for new research.

This chapter fills the gap in the academic literature by exploring the perspectives of IoT practitioners on the challenges, tools, approaches, and artifacts related to IoT systems testing in real-world settings. By gathering this information, we provide insights into the actual needs and experiences of industry practitioners. Consequently, we study the approaches and tools to test IoT systems in the industry. We focus on testing approaches, tested quality attributes, tools that practitioners use to test IoT systems, and the main challenges faced by practitioners when testing IoT systems. To achieve our objective, we use three methods to obtain facts from IoT practitioners: (1) an industry survey with 49 practitioners about testing IoT systems, (2) interviews with 9 practitioners about testing decisions, (3) data analysis of four surveys conducted by EclipseIoT[1] with IoT systems developers. This chapter is guided by the following research questions (RQs):

- **RQ3**: How do practitioners test IoT systems?
- **RQ4**: How do testers make decisions in IoT system testing?
- **RQ5**: What are the trends in IoT systems testing?

To ensure the quality of our subquestions, we follow **SPIDER [34], a framework for formulating qualitative or mixed-methods research questions (RQs) focusing on *samples*. We also followed best practices, in particular those provided by [1, 116].**

In the survey with IoT practitioners, we aim to answer RQ3: How do practitioners test IoT systems? We divide this question into the following subquestions:

1. RQ3.1: What are the tools and approaches used by practitioners for IoT systems testing in the industry?
2. RQ3.2: What are the quality attributes considered when testing IoT systems by practitioners?
3. RQ3.3: What are the artifacts recommended by practitioners for IoT test automation?

[1]Open source community for IoT. https://iot.eclipse.org/

4. RQ3.4: What are the challenges faced by IoT practitioners when testing IoT systems?

We also conduct interviews with IoT practitioners to answer RQ4: How do testers make decisions in IoT system testing? We divide this question into three subquestions:

1. RQ4.1: How do testers choose the right testing approaches, levels, coverage, and metrics?
2. RQ4.2: How can researchers contribute to overcoming testing challenges in the IoT?
3. RQ4.3: How are testing artifacts automation prioritized in IoT systems?

We analyze data from surveys conducted by EclipseIoT to answer RQ5: What are the trends in IoT systems testing? We device this question into three subquestions:

1. RQ5.1: What are the practitioners' top challenges when testing IoT systems?
2. RQ5.2: What are the top IoT communication protocols and technologies?
3. RQ5.3: What are the top IoT cloud platforms available to the testers?

By answering these subquestions, we can recommend to both practitioners and academia, the available testing tools, testing approaches, testing metrics, test coverage, testing artifacts, IoT quality attributes, tested layers, and testing challenges for IoT systems. We present a summary of our findings from four different perspectives: 1. Testing focuses more on the device, network, and application layer. Integration testing is the most considered testing level, whereas acceptance testing is the least considered. Test coverage is the top metric for IoT system testing, and the choice of metrics varies based on the project. 2. Model-based approach is popular for IoT system testing. IoT system testing is still manual or semi-automated, whereas the adoption of white-box testing is low. Node-RED is the most used tool in testing IoT systems, while AWS[2] IoT is a popular cloud platform for testing IoT devices. 3. Log analysis is the main approach to analyze the root cause of bugs. 4. Top challenges in IoT systems testing include lack of standards, security, connectivity, and lack of reference architecture. Test case generation and standard approach for

[2]https://aws.amazon.com

IoT systems testing are the top-recommended research focus.

The main contributions of this chapter are:

1. Identify the main challenges faced by IoT practitioners when testing IoT systems to guide future research.
2. Compile the top-quality attributes considered by practitioners for IoT systems.
3. Identify testing tools, testing approaches, testing metrics, test coverage, testing levels, and most tested layers in IoT systems by practitioners.
4. Provide top artifacts to consider for IoT test automation.
5. Summarize top IoT protocols, technologies, and IoT middleware.
6. Discuss lessons learned to guide future work.

The rest of this chapter is organized as follows: Section 5.2 describes the multi-method study. Section 5.3 presents findings and answers to our subquestions, while Section 5.4 reports our observations based on the analyzed data. Section 5.5 presents possible threats that could affect the validation of our answers. Finally, Section 5.6 concludes this chapter.

5.2 Research Method

The objective of this chapter is to *analyze the current state of IoT systems testing from the viewpoint of industry practitioners and identify the key challenges affecting the testing process*. To achieve our objective, we target practitioners from the industry. We mainly target IoT systems testers or quality assurance engineers for IoT solutions. We equally target other professionals in the IoT industry, including IoT developers, IoT project managers, product owners, and maintenance engineers. We use the data from our own survey and data from the survey conducted by EclipseIoT on developers' concerns. Both the EclipseIoT survey and our own survey are important for this research. Conducted annually, the EclipseIoT survey enables us to gain a comprehensive understanding of persistent challenges since 2019. The EclipseIoT survey, with a large participant pool of approximately 600 developers, provides breadth, while our survey with 49 participants provides depth. While the

EclipseIoT survey primarily focuses on developer concerns, our study takes a broader approach, encompassing testing tools, approaches, and quality attributes considered in IoT. By combining both surveys, we enhance the comprehensiveness of our research. Fig. 5.1 shows the steps of our multi-method study.

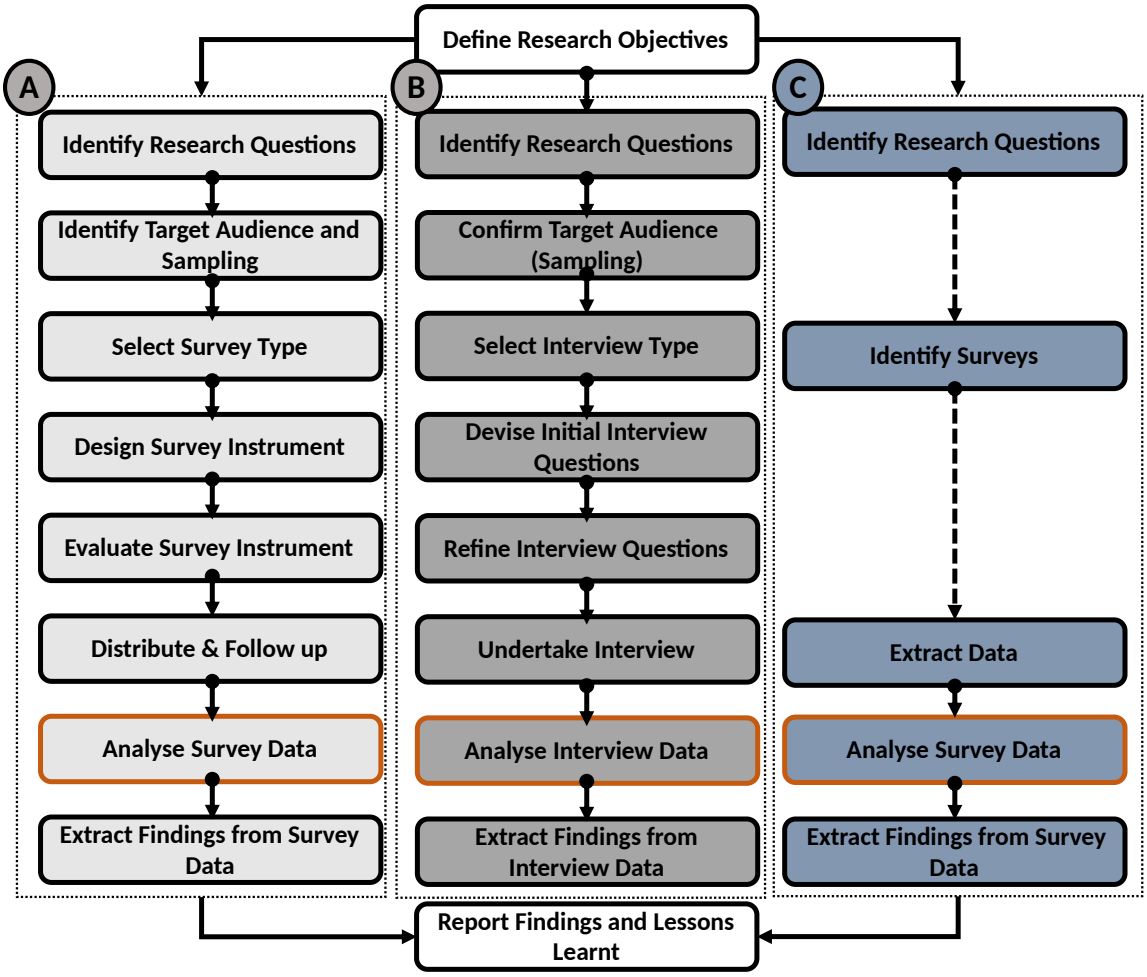


Figure 5.1: Research Methodology

We define subquestions for each method based on the main objective. Table 5.1 summarizes the defined subquestions. We start with a primary survey to answer four RQs ($RQ3_s1$, $RQ3_s2$, $RQ3_s3$, and $RQ3_s4$). Based on survey answers, we conduct interviews with IoT practitioners, who willingly accepted to participate. We analyze the interview responses to find the answers to three RQs ($RQ4_i1$, $RQ4_i2$, and $RQ4_i3$). We also analyze four surveys

Table 5.1: Research Questions (RQs)

S/No	RQs	Rationale
$RQ3_s1$	What are the tools and approaches used by practitioners for IoT systems testing in the industry?	We want to learn how IoT systems are being tested in the industry to complement what has been reported by academia.
$RQ3_s2$	What are the quality attributes considered when testing IoT systems by practitioners?	With the lack of standards in the IoT industry, we seek more insights into quality attributes applicable to IoT systems and possible quality attributes overlooked when testing IoT systems.
$RQ3_s3$	What are the artifacts recommended by practitioners for IoT test automation?	The recommendation from IoT practitioners will help the research community to focus on the most needed aspects of IoT test automation.
$RQ3_s4$	What are the challenges faced by IoT practitioners when testing IoT systems?	The feedback from industry practitioners will help the research community search for solutions to real challenges.
$RQ4_i1$	How do testers choose the right testing approaches, levels, coverage, and metrics?	We want to know how decisions are made when deciding on which approach to use, layers to test, levels to test, test coverage, and test metrics.
$RQ4_i2$	How can the research community contribute to overcoming testing challenges in IoT?	We seek to hear from practitioners the research focuses to find solutions to their challenges.
$RQ4_i3$	How is test artifacts' automation prioritized in IoT systems?	We want to know why practitioners recommended the automation of some test artifacts while others are overlooked.
$RQ5_e1$	What are the practitioners' top challenges when testing IoT systems?	We want to know the top challenges of IoT practitioners for guiding the research community in understanding their needs and priorities.
$RQ5_e2$	What are the top IoT communication protocols and technologies?	We seek more insights on the most popular communication protocols and technologies applicable to IoT systems, which could impact the testing process.
$RQ5_e3$	What are the top IoT cloud platforms available for testers?	While conducting interviews, some practitioners mentioned that they test IoT solutions using IoT cloud platforms. We seek more insights on publicly available IoT middleware which could be used for testing IoT systems.

① $RQ3_s$: Primary survey related Research Questions
② $RQ4_i$: Interview Related Research Questions
③ $RQ5_e$: EclipseIoT survey related Research Questions

conducted by EclipseIoT from IoT practitioners between 2018 and 2022 to find answers to three RQs ($RQ5_e1$, $RQ5_e2$, and $RQ5_e3$). We relate the answers from each category and draw a conclusion.

In the process of analyzing the data to find the answers to our RQs, we use the steps presented in Fig.5.2.

The rest of this section is organized as follows: Section 5.2.1 focuses on the primary survey. This section is organized as follows: Section 5.2.1 focuses on the target audience and

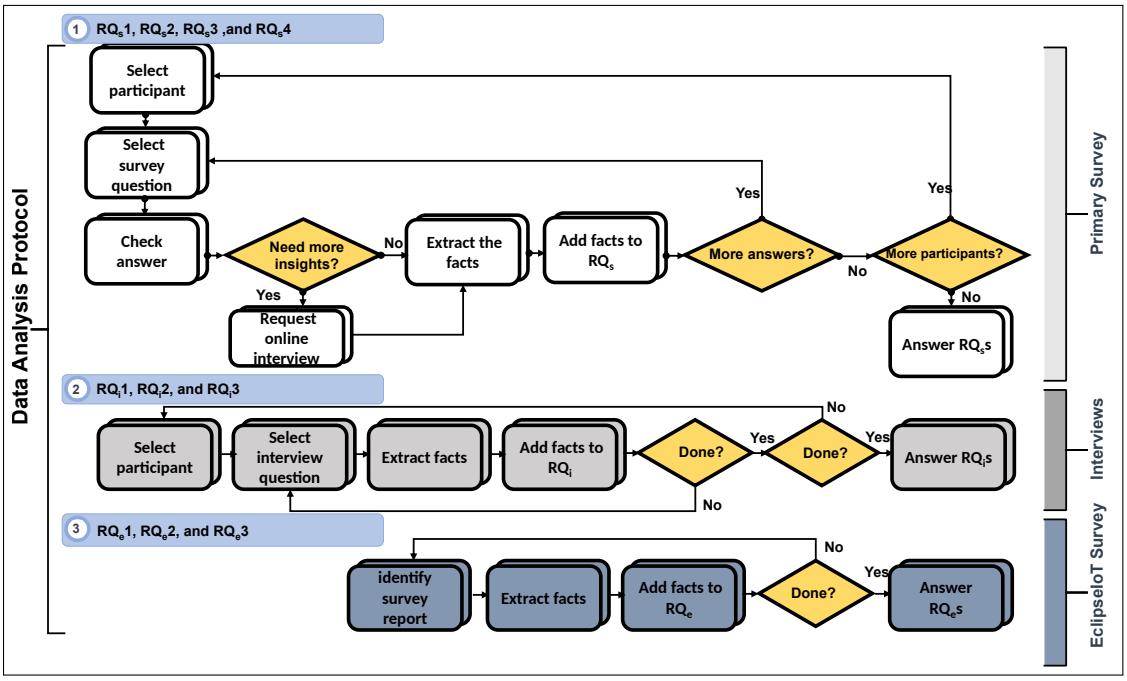


Figure 5.2: Data Analysis Process

sampling. Section 5.2.1 explains the type of survey we used. Section 5.2.1 describes survey design. Section 5.2.1 describes survey instrument evaluation. Section 5.2.1 focuses on how we collect and analyze answers from the survey. Section 5.2.1 focuses on answers to our questions and conclusions. Section 5.2.2 focuses on interviews. This section is organized as follows: Section 5.2.2 shows the practitioners who participated in our interviews. Section 5.2.2 explains the choice of the type of interview we used. Section 5.2.2 outlines how we devise the interview questions. Section 5.2.2 summarizes the refined interview questions. Section 5.2.2 provides details of how we conduct interviews. Section 5.2.2 describes how we analyze interview data. Section 5.2.2 focuses on the answers to interview-related RQs. Section 5.2.3 focuses on EclipseIoT survey data analysis. This section is organized as follows: Section 5.2.3 shows the surveys we identified from the EclipseIoT community. Section 5.2.3 explains how we extract data from those surveys. Section 5.2.3 describes how we analyze the extracted data. Section 5.2.3 focuses on answers to our three RQs. Section 5.2.4 relates the answers from primary surveys, interviews, and EclipseIoT surveys.

5.2.1 Primary Survey

A survey is one of the empirical investigation methods which is used to collect data from a large population [63]. We aim to collect both quantitative and qualitative data from the industry to obtain evidence of the state of the art of IoT systems testing from the perspective of practitioners. We follow the steps defined in [63] as shown in block A of Fig. 5.1. Surveys are an appropriate empirical strategy to gather data from IoT industry practitioners (e.g., about methods, tools, techniques, approaches, challenges) and to extract insights into the state of the art from the participants [156]. The scope of the survey includes various aspects of IoT systems testing from testing tools, approaches, quality attributes, testing levels, testing layers, test automation status, and IoT architecture tested. The rest of this section explains other steps of the survey process.

Identify Target Audience and Sampling

We want to reach out to many practitioners as possible. We target people who are involved in IoT systems testing. Depending on the company organization, those professionals may be in different positions or job titles. We use judgment sampling to select the sample through the guidance of an expert [63], and we target the followings roles:

- IoT systems developers;
- IoT systems testers;
- IoT systems quality assurance engineers;
- IoT project managers;
- IoT product owners.

We include in our form the option for the participant to write any other position or role not mentioned in the list.

Select Survey Type

Several types of surveys exist based on deployment methods.

- **Online Surveys**: A survey method that uses the internet to collect data, typically using a web-based survey platform.
- **Paper or Mail Surveys**: A traditional survey method, where participants receive a printed questionnaire and return it by mail or in person.
- **Telephone Surveys**: Participants are interviewed by telephone, either by a live interviewer or using an automated system.
- **In-person Surveys**: Interviewers visit participants at their locations to collect data.

Each of these survey methods has its own strengths and weaknesses, and the best method for a particular study depends on various factors, including the RQs, the target population, and the resources available. An online survey is becoming more popular, especially when it is used together with social media. We want to use this type of survey to reach out to many professionals.

Design Survey Instrument

Designing a survey instrument refers to the process of creating a survey questionnaire or form to collect quantitative and qualitative data from participants. Table 5.2 shows the questions we used in our survey. Our survey consists of three main sections. The first two sections consist of close-ended questions, whereas the third section includes open-ended questions. The following are the three main sections of our survey questionnaire:

- Practitioners' background information (Quantitative data).
- IoT testing facts from the provided options (Quantitative data).
- Understanding of practitioners in IoT systems (Qualitative data).

We want to collect information about the practitioners, including the address of affiliated organizations. We also capture job titles or positions, and years of experience in IoT. Regarding IoT systems testing facts, we ask about the tools used, top challenges, quality attributes tested, metrics used, etc. We provide different options for the practitioners to select from when answering those questions. The last section targets the understanding of

Table 5.2: Survey Questions

No	Survey Question
A. Demographic Information	
1	What is your job position? [Choose the best position that fits you]
2	How many years have you been working in IoT Systems?
3	What is the address of the institution or company where you work?
4	What are the top two biggest challenges you are currently facing while testing IoT systems?
B. IoT Testing Facts	
5	On which IoT layers (device, network, cloud, or application) of your focus when testing your IoT systems
6	How do you test your IoT systems?
7	On which test level of your focus when testing an IoT system?
8	What is the architecture (based on a number of layers) of the IoT systems you tested?
9	What are the top two test artifacts that you recommend for automation for IoT system testing?
10	What are the types of test coverage (Line of code, branch, requirements, etc.) of your focus while testing an IoT system, and why?
11	What are the quality attributes do you test before deploying an IoT system?
C. Understanding IoT Testing	
12	What is your approach when you want to test a new IoT system, and why?
13	What are the tools that you use for testing IoT systems, and for which purpose?
14	Do you use any frameworks/tools (if any) to conduct End-to-End automated tests for IoT systems? If yes, give the name of the framework.
15	What are the metrics (if any) that you use while testing an IoT system, and why?
16	What are the open-source tools do you use to track the root cause of the problem when an IoT system has an error or a bug?
17	Has your testing activity been affected by the lack of standards, reference architecture for IoT systems, or the use of different communication protocols? If yes, what was the impact?
18	What are the tools previously used in testing the embedded system that are still used to test the IoT systems?
19	Did you ever perform white-box testing for an IoT System? If yes, how did you perform it, and on which layer?
20	Did you use a model-based approach while testing IoT systems? If yes, which tools did you use?

① Group A and B offer practitioners a selection of options, along with the flexibility to incorporate their own answers.
② Group C consists of open-ended questions, allowing for unrestricted responses

practitioners in IoT systems testing. The survey form can be found here.

Evaluate Survey Instrument

After designing the survey instrument, we evaluate it to find if there are any flaws. To validate our questionnaire, a preliminary evaluation is conducted by two researchers. The purpose is to check the completeness and understandability of the questionnaire and determine how to distribute the questionnaire. During survey pretesting, we share the survey questionnaire with 3 IoT experts to test it before sending it to other participants.

Collect and Analyze Survey Data

We delete any data provided during survey pretesting and send the questionnaire to several professionals. We start data collection on July 1st, 2022. We finish data collection on September 30, 2022. We use Google Forms to design and generate the invitation link for the practitioners. We start with technology companies that have IoT projects, and we identify their practitioners and ask them to participate. We identify those companies through online websites including Facebook, LinkedIn, and Twitter. From those websites, we get the contacts of practitioners and the companies they work for. We contact the companies to get more practitioners to participate. We also consider different IoT research labs practitioners, and IoT practitioners from some alumni groups to participate. We follow the steps in the first part of Fig. 5.2 and Algorithm 2 to extract data from the responses provided by the participants. We record the extracted data in Excel. To statistically analyze the collected data, we use Excel to count different types of feedback (responses) in the survey, calculate percentages of the different responses, and generate tables or visualizations of the calculated results.

Extract Findings

We use the extracted facts to answer and draw the conclusion for the following RQs: $RQ3_s1$, $RQ3_s2$, $RQ3_s3$, and $RQ3_s4$. The answers to these RQs are presented in Section 5.3.1.

5.2.2 Interviews

An interview is defined as an "*interchange in which one person attempts to elicit information or expressions of opinion or belief from another person or persons*" [111]. We conduct online interviews with practitioners who agreed to provide more details about their survey answers. The initial purpose of the interviews was to get clarification and possibly resolve contradictions among survey answers. However, based on the responses we received, we

Algorithm 2 Data Analysis Algorithm

```
1: function DataAnalysis(N, K)
2:     N ← 49, K ← 20
3:     for p ← 1 to N do
4:         for q ← 1 to K do
5:             if q is not clear then
6:                 Request online interview
7:                 if Interview is confirmed then
8:                     Update Interview Participants
9:                 end if
10:            end if
11:            Extract Facts from q
12:            Add Facts to Appropriate RQ3_s
13:        end for
14:    end for
15:    return RQ3_s Answers
16: end function
```

ended up defining three additional RQs. we use the interviews to get clarification on survey answers, but also to answer the following additional RQs: $RQ4_i1$, $RQ4_i2$, and $RQ4_i3$. We follow basic steps derived from [202] as indicated in block B of Fig. 5.1. With the research objective and RQs explained, the rest of this section explains the remaining steps in an interview study.

Sampling

Table 5.3 shows the details of the participants whom we interviewed. Nine practitioners among the 49 agreed to participate in an online interview.

Select Interview Type

Interviews come in various types, depending on the number of participants and the means to interview them. For example, one type of interview is a one-on-one, in-person interview while another is the panel, video interview [111]. We use a structured interview, which is based on a fixed set of pre-determined questions [202], in order to gather information. We define open-ended questions to give the respondent the opportunity to explore the questions

Table 5.3: Interview Participants

No	Profession	Years	Country
P1	IoT project manager	12	Lithuania
P2	IoT project manager	12	USA
P3	Senior IoT solution developer	8	UAE
P4	IoT project manager	7	South Korea
P5	IoT product owner	6	USA
P6	IoT solution developer	5	USA
P7	IoT solution developer	4	Italy
P8	IoT solution developer	4	German
P9	IoT solution developer	3	Pakistan

① **Years**: Years of experience in IoT

from multiple perspectives, and this allows us to gather a variety of information about the research subject. We combine the use of one-on-one and video interviews.

- **One-on-One Interview**: This is the most common type of interview, where the interviewee meets with the interviewer to answer questions and provide information.
- **Video Interview**: A video interview is a type of interview that is conducted remotely, usually through a video conferencing platform such as Zoom, Skype, or Google Meet.

Devise Initial Interview Questions

We design the interview questionnaire and test it with the authors of this paper to check the time it takes to complete. We also check the language suitability, and potential sources of bias to verify if it produces enough relevant data to answer the RQs. We use the feedback from this pilot interview, to improve our interview design.

Refine Interviews

We use the feedback from the previous task to improve our interview design. Table 5.4 shows the questions included in our interview design. Our interview design consists of eight open questions. All the questions are designed based on the answers received from the survey.

Table 5.4: Interview Questions

No	Interview Question
1	What can be done to overcome testing challenges?
2	Why some test artifacts must be automated ahead of others?
3	Why do you consider some test coverage and not others?
4	What levels, and layers, the quality attributes are tested?
5	What testing approach do you use? Why?
6	How are test metrics selected for user acceptance?
7	How are bugs identified in IoT systems?
8	What impact is caused by the lack of standards, reference architecture, and multiple communication protocols?

Interview

We conduct an online interview with each participant. With their consent, we record the answers, and we later analyze them to find the answers to the concerned RQs.

Analyze Interview Data

We follow the steps in the middle part of Fig. 5.2 and Algorithm 3 to analyze interview data and extract the facts for RQs.

Extract Findings

We use the extracted facts to answer the following RQs: $RQ4_i1$, $RQ4_i2$, and $RQ4_i3$. The answers to those RQs are presented in Section 5.3.2.

Algorithm 3 Interview Data Analysis Algorithm

```
1: function InterviewDataAnalysis(K, Q)
2:     K ← 9, Q ← 8
3:     for i ← 1 to K do
4:         for j ← 1 to Q do
5:             Extract Facts from j
6:             Add Facts to Appropriate RQ4_i
7:         end for
8:     end for
9:     return RQ4_i Answers
10: end function
```

5.2.3 EclipseIoT Surveys

During our interview sessions, we learn that the EclipseIoT Working group publishes IoT developers' surveys every year since 2018. The purpose of each survey is to provide essential insights into IoT and edge computing industry landscapes. Those surveys present several findings including IoT developers' concerns, IoT communication protocols, IoT security technologies, connectivity protocols, IoT middleware, IoT programming languages, edge computing artifacts for IoT, IoT development tools, open-source databases for IoT, IoT strategies, and key industry focus areas. We use the insights from those interviews to find answers to the following RQs: $RQ5_e1$, $RQ5_e2$, and $RQ5_e3$. Both research objectives and RQs are already defined. We focus on the remaining steps of the C block in Fig. 5.1.

Identify Surveys

We identify four different surveys published by the EclipseIoT Working group in 2019 [78], 2020 [79], 2021 [80], and 2022 [81], with ***1717, 1652, 662, and 910*** participants respectively.

Extract Data

We use an Excel spreadsheet to extract data manually from each of those four surveys. We double-check the extracted data to ensure its correctness.

Data Analysis

We follow the steps in the last part of Fig. 5.2 and Algorithm 4 to extract facts from survey data to answer the following RQs: $RQ5_e1$, $RQ5_e2$, and $RQ5_e3$.

Algorithm 4 Data Analysis Algorithm

```
1:  function DataAnalysis(N, K)
2:      N ← 49, K ← 20
3:      for p ← 1 to N do
4:          for q ← 1 to K do
5:              if q is not clear then
6:                  Request online interview
7:                  if Interview is confirmed then
8:                      Update Interview Participants
9:                  end if
10:             end if
11:             Extract Facts from q
12:             Add Facts to Appropriate RQ3_s
13:         end for
14:     end for
15:     return RQ3_s Answers
16: end function
```

Extract Findings

We use the extracted facts to answer $RQ5_e1$, $RQ5_e2$, and $RQ5_e3$ and draw some conclusions. The detailed answer for each of those questions is presented in Section 5.3.3.

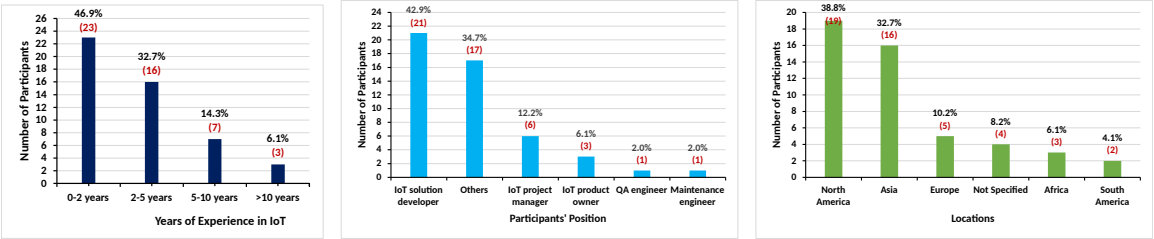
5.2.4 Findings Analysis

We relate the answers to our subquestions from each method, and we draw some conclusions based on our findings. We also relate the findings of this multi-method study with the findings in the existing literature. The relationship between some of our answers and corresponding conclusions is presented in Section 5.3.4.

5.3 Results

We analyze the extracted data to find insights to answer our research question "RQ2: How are IoT systems tested, and what challenges do practitioners face?". In the next subsections, we present the answers to each subquestion of our research question.

5.3.1 RQ3: How do practitioners test IoT systems?



(a) Experience In IoT (b) Participants' Positions (c) Participants' Locations

Figure 5.3: Survey Participants

Fig. 5.3 shows the demographic information of the participants. The highest number of survey participants are IoT solution developers, with 21 participants (42.9%). Regarding the professionals who are dedicated to IoT testing, we have identified only one participant (2.0%) as an IoT systems QA engineer. However, 17 participants (34.69%) did not provide their roles. Based on this finding, we can conclude that the developers of IoT systems are also involved in testing those systems. North America has the most participants, with 19 professionals (38.8%). Asia comes next with 16 participants (32.7%), followed by Europe with 5 participants (10.2%). Africa and South America have 3 and 2 participants respectively. Four participants (8.0 %) did not specify the location of their affiliated companies, since they are independent consultants and we allowed them to be anonymous if they do not wish to disclose such information. Twenty-three participants (46.9%) have between 0 and 2 years of work experience in IoT. Sixteen participants (32.7%) have between 2 and 5 years of work experience in IoT. Seven participants (14.3%) have between 5 and 10 years

of experience in IoT, whereas 3 participants (6.1%) have more than 10 years of work experience in IoT. At least 10 of our participants have more than 5 years of experience in IoT, and we believe that the answers to this survey are reliable. In the next section, we present the answers to our survey research questions.

- **How Are IoT Systems Tested (RQ3.1)?**
  - **Testing Tools for IoT Systems**. Testing tools are software packages that help in testing and evaluating the quality of IoT systems. These tools are designed to automate the testing process. ISO/IEC/IEEE 29119-1 describes some of the areas covered by testing tools such as test case generation, test case execution, test data generation, static analysis, test environment implementation, and maintenance [3]. We focus on the tools that are used to evaluate the quality and functionality of IoT systems. We exclude test environment tools such as testbeds, emulators, or simulators. We analyze the data provided by the participants to extract different tools available for testing IoT systems. Node-RED, a programming tool for wiring together hardware devices, APIs, and online services [138], is the most popular among the participants with 35.5%. Other tools such as Wireshark, ThingSpeak, and Apache JMeter are also popular. Selenium, which is also popular for automating web applications for testing purposes [166], is used to test IoT systems and many other tools we reported as shown in Table 5.5.

    Nine participants (29.0%) indicated that they do not use any tool when testing IoT systems. They test IoT systems manually through navigation. i.e., users performing tests by entering information into a system under test and verifying the results [3].

    We observe that Node-RED is popular among IoT practitioners to create and execute Node-red flow simulating IoT devices and testing the communication flow in the system infrastructure [171]. Practitioners mentioned other tools, such as JTAG Dongle, Cucumber, and Mocha, that are not identified in the literature.
  - **Tools Applicable to Embedded Systems**. Some tools used in embedded systems are

Table 5.5: IoT Systems Testing Tools

Tool Name	**# P**	**% P**
Node-RED	11	35.5%
Wireshark	5	16.1%
ThingSpeak	3	9.7%
Apache JMeter	2	6.5%
SmartThings	2	6.5%
Zetta	1	3.2%
Apache Kafka	1	3.2%
Selenium	1	3.2%
JTAG Dongle	1	3.2%
Gemoc	1	3.2%
Tricentis Tosca	1	3.2%
AccelQ	1	3.2%
ThingBoard	1	3.2%
Tcpdump	1	3.2%
Cucumber	1	3.2%
TestRigor	1	3.2%
JEst	1	3.2%
Mocha	1	3.2%

① **#P**: Number of participants; **%P**: Percentage of participants.

also being used to test some aspects of IoT systems. Node-RED, Tessy, and Wireshark are good examples with 4.0%, 8.0%, and 4.0% respectively. 22 participants did not provide their responses to this question, while 19 participants do not know any embedded system testing tool that can be also used for IoT systems. *We observe that Node-RED and Tessy could be used in embedded systems. Arduino IDE was not identified in the existing literature. However, the number of participants using those tools is small and we can not take any conclusion. Therefore, further studies are needed.*

– **Metrics for IoT Systems**. A metric is a standard unit of measurement that quantifies results. In the context of IoT, metrics used for evaluating or testing IoT systems are termed IoT metrics [137]. To effectively control the test processes, metrics can be used to monitor them [3]. We observed that test coverage (i.e., the number of requirements

covered by executed tests [3]), is popular with 17.2%. The success rate of executed test cases (pass or failure), and response time come next with 13.8% each. Performance is the fourth with 10.3%. Functionality, code coverage, connectivity, and the number of active clients follow with 6.9% each. Many other metrics are also reported including latency, packet loss, number of faults, feedback from customers, number of connected devices, etc. as shown in Table 5.6.

Table 5.6: IoT Testing Metrics

Metric	**# P**	**% P**
Test coverage	5	17.2%
Success rate (Pass/Fail)	4	13.8%
Response time	4	13.8%
Performance	3	10.3%
Functionality	2	6.9%
Code coverage	2	6.9%
Connectivity	2	6.9%
No of active clients	2	6.9%
Scalability	1	3.4%
Quality	1	3.4%
Reliability	1	3.4%
Packets loss	1	3.4%
Latency	1	3.4%
Periodic reading	1	3.4%
Data transmission frequency	1	3.4%
Effectiveness	1	3.4%
Efficiency	1	3.4%
Number of connected devices	1	3.4%
Customer feedback	1	3.4%
Security	1	3.4%
Accessibility	1	3.4%
No of faults	1	3.4%

① **#P**: Number of participants; **%P**: Percentage of participants.

We observe that test coverage, response time, and code coverage are also identified in the existing literature [74, 23, 103, 39]. In the literature, we identified other metrics such as the number of defects detected, mutants killed, and requests per second that

are not mentioned by practitioners in this chapter. We have come to the conclusion that there are a lot of metrics for testing IoT systems. This means that the must-have list of metrics for IoT systems needs to be studied in depth.

– **Test Coverage Considered for IoT Systems**. Test coverage is an indication of the degree to which the test item has been reached or "covered" by the test cases, including both the breadth and depth. It is often expressed in terms of the percentage of code tested or the percentage of requirements tested [172]. We received 45 responses for the metrics, and one participant is allowed to provide more than one coverage. In the answers we received from the participants, we identified three main coverages: functional coverage, requirements coverage, and code coverage. Code coverage is popular at 82.2% because most IoT developers test the application layer as shown by the data received on IoT layers tested. Requirements coverage is popular at 60.0%, whereas functional coverage is mentioned at 55.6%. *We conclude that only three test coverages are popular among practitioners. Other techniques, such as path coverage, branch coverage, and statements coverage, are not mentioned by practitioners. Many practitioners mentioned code coverage, but this is only applicable to the application layer since they focus more on this layer. For user acceptance test, requirements coverage and functional coverage could be considered for any IoT system.*

– **Testing Approaches for IoT Systems** The testing approach is defined as a high-level test implementation choice, typically made as part of the test strategy design activity. Typical choices made as test approaches are test level, test type, test technique, test practice, and the form of static testing to be used [3]. IoT practitioners reported different approaches for testing IoT systems. Many of them mentioned that they do not have any systematic approach to testing IoT systems. Component-based testing is popular among the reported approaches, with 22.9%. The verification and validation approach is the next with 11.4% followed by integration, interoperability, and

security with 8.6% each. Model-based, Test driven development and simulation-based approaches are also popular with 5.7% while others, such as automation, end-to-end, and low-level coding are only mentioned by 2.9% of the participants as shown in Table 5.7.

Table 5.7: IoT Testing Approaches

Testing Approach	**% P**	**# P**
Component-Based Approach	22.9%	8
Verification & Validation Approach	11.4%	4
Integration Based Approach	8.6%	3
Interoperability Based Approach	8.6%	3
Security Based Approach	8.6%	3
Simulation-Based Approach	5.7%	2
Test Driven Development	5.7%	2
Model-Based Approach	5.7%	2
Low level Coding Approach	2.9%	1
Automation Based Approach	2.9%	1
End-to-End Approach	2.9%	1
Usability Based Approach	2.9%	1
Scalability Based Approach	2.9%	1
Did not mention	34.3%	12

① **#P**: Number of participants; **%P**: Percentage of participants.

Twelve participants (34.3%) reported that they do not use a specific approach when testing IoT systems. They navigate through the system features, to verify if the system works as expected.

We observe that component-based is the most popular approach among the participants. We also observe that participants confuse testing approaches with testing types or targets. verification, validation, security, interoperability, integration, end-to-end, or low-level coding, are not known as testing approaches, and thus show the need for IoT testing taxonomy. In the literature, we identified more approaches such as pattern-based, mutation-based, and fault injection approaches, not mentioned by the practitioners. We can conclude that many practitioners may use some testing approaches,

but they do not know the names of those approaches.

– **Testing Levels for IoT Systems**. Testing approaches can be on one or more levels, depending on the scope of the test and its objective. So, different test levels are defined, as follows [3]: **Unit Testing**: Testing of individual hardware or software units or groups of related units [172]. It consists of isolating each part of the system and showing that individual parts fit its requirements and functionalities. **Integration Testing**: Software and/or hardware components are combined and tested to check the interaction between them and how they perform together [172]. **System Testing**: Testing a complete, integrated system to check the system's compliance and behavior within the specified requirements [172]. **Acceptance Testing**: Formal testing conducted to determine whether a system satisfies its acceptance criteria and to enable a customer, a user, or other authorized entity to determine whether to accept the system [172] Table 5.8

Table 5.8: Testing Levels in IoT Systems

Testing level	**# P**	**% P**
Integration	32	65.3%
Unit	18	36.7%
System	18	36.7%
Acceptance	8	16.3%

① **#P**: Number of participants; **%P**: Percentage of participants.

shows testing levels considered for IoT systems. As for traditional software systems, unit testing, integration testing, system testing, and acceptance testing are adopted for IoT systems [176]. Among our participants, integration testing is the most popular testing level, followed by unit, and system testing. Acceptance testing is given less consideration among the four levels of testing by practitioners. *We observe that integration testing is the most popular testing level, followed by unit and system testing among many practitioners.*

– **Layers for IoT Systems Testing** We can define the IoT layer as a fundamental constituent of an IoT system. Basic IoT architecture consists of 3 layers: device (also known as edge or perception or sensing or thing) layer, network (also known as transport or gateway or communication) layer, and application layer [115]. Some architectures include processing as an additional layer and a business layer.

1. **The Device layer** of IoT architecture also known as the sensing layer or perception layer includes devices, sensors, and actuators that collect data from their surroundings and control things at the edge [73]. For example, a temperature sensor takes temperature readings inside a refrigerator.
2. **The network layer** is responsible for connecting to other smart things, network devices, and servers. Its features are also used for transmitting and processing sensor data [168].
3. **The application layer** is responsible for delivering application specific services to the user [168].
4. **The processing layer** is also known as the middleware layer. It stores, analyses, and processes huge amounts of data that come from the transport layer. It can manage and provide a diverse set of services to the lower layers. It employs many technologies such as databases, cloud computing, and big data processing modules [168]. For example, the cloud stores and processes the incoming data to generate alerts in real-time and, when possible, reduce the total amount of data stored [73].
5. **The business layer** derives information and decision-making analysis from data.

Although some authors mentioned human or user layer [50], industry practitioners did not mention it. We analyze the extracted data to understand which layers are being considered for IoT system testing. Testing IoT systems involve testing different layers. However, we observed that the application layer is the most tested layer with 63.8%, followed by the network layer, and device layer with 55.3% and 51.1% respectively.

The business layer and cloud layer are less tested layers because many IoT systems may not have those layers. *We observe that the participants focused more on the application layer, followed by network and device layers.*

- **Architecture of IoT Systems** IoT system architectures consist of 3, 4, or 5 layers [168, 134, 115]. The participants mainly indicated that the systems tested consisted of 3-layer and 4-layer architecture. 5-layer architecture is less adopted by the participants. *In the literature, 3-layer architecture is popular. However, we believe that a five-layer architecture can help both scholars and practitioners to better address the complexity of IoT systems.*
- **IoT System Testing Automation** Test cases can either be run manually by a human test executor, or they can be executed by a test automation tool [3]. In automation testing, a test is executed by a test automation tool. We analyze the data to understand to what extent IoT testing is automated. None of the participants performed fully automated testing for IoT systems. 59.2% of the participants performed semi-automated testing, while 40.8% performed manual testing only. *We conclude that the participants do not perform fully automated tests for IoT systems. In the existing literature, we did not identify any approach or tool for automated IoT systems testing. This could be one of the research areas that need further studies.*
- **White-Box Testing in IoT** White-box testing, which is a method of testing internal structures of IoT systems, is not popular among participants. We did not focus on black or gray box testing because we want to understand to what extent the source code and internal structure of the device are tested. Only 14.8% of participants performed white-box testing at some layers (application and device layers). 85.2% of the participants did not perform white-box testing for IoT systems. *We observe that white-box testing is considered by some practitioners for application and device layers testing. We did not identify white-box testing information in our previous literature review study.*

– **Model-Based Testing Adoption** Model-based testing (MBT) uses models to generate test cases systematically and automatically [3]. ISO/IEC/IEEE29119-1 states that "by using MBT tool support environments, test cases can be quickly generated from the model and automatically executed. Thus, MBT can support improved testing beyond that supported by natural languages and manual execution". In the existing literature, model-based testing is a popular testing approach for IoT systems. In this chapter, we want to understand practitioners' perspectives. Among 25 participants who responded to this question, 12 (48.0%) used a model-based approach when testing IoT systems. *We observed that model-based testing is a popular approach among IoT practitioners.*
– **IoT Bugs Root Cause Analysis** We observe that no specific tool is popular for the participants to identify the root cause of a bug in IoT systems. Among 27 participants, some rely on logs monitoring and analysis to understand the root cause of problems or manual detection by checking each component. Participants suggested some tools used including Nagios, Splunk, Arduino, Azure DevOps, and Gemoc, as indicated in Table 5.9.

Table 5.9: Bugs Root Cause Analysis Tools

Tool	**# Participants**	**% Participants**
Nagios	4	14.8%
Logs analysis	2	7.4%
Manual detection	1	3.7%
Splunk	1	3.7%
Arduino	1	3.7%
Azure DevOps	1	3.7%
Gemoc	1	3.7%

① **#P**: Number of participants; **%P**: Percentage of participants.

We do not consider the answers from 16 participants (59.3%) for this question because they mentioned SpiraTeam tool which is not used for root cause analysis but for bug tracking. *We do not observe any specific tool that is popular among the participants*

for identifying the root cause of bugs in IoT systems. We conclude that some tools for analyzing the root cause of bugs in IoT systems may exist, but remain unknown to practitioners. This calls for a joint effort between scholars and industry practitioners to develop more tools.

– **Impact of Standards, Reference Architecture, and Protocols** We found that the lack of standards and reference architecture affected many participants. 75.0% of the participants reported a negative impact for not having standards, reference architecture, and the use of multiple protocols in IoT systems. *We conclude that lack of standards in IoT can have a negative impact on IoT systems testing. We observed the same in the existing literature.*

Takeway for RQ3.1

Semi-automated testing in IoT is popular. Node-RED is the most used tool with 35.5% of the participants. The application layer is the most tested layer with 63.8% of the participants, while integration testing is the most considered testing level with 65.3% of the participants. Component-based is the most popular approach with 22.9% of the participants. Requirements coverage is the top metric considered with 17.2%, while code, requirements, and functional coverage are the most used test coverage.

- **Quality Attributes (RQ3.2)** The product quality model defined in ISO/IEC 25010 comprises eight quality attributes, as shown in Fig. 5.4. Quality attributes, included in

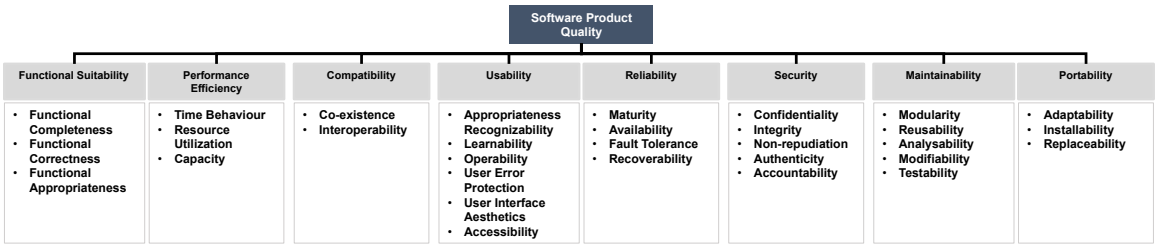


Figure 5.4: ISO/IEC 25010 software product quality model [2]

Fig.5.4, were proposed for the traditional software product. All of them, except functional suitability and maintainability, are considered for IoT systems by the practitioners. The practitioners mentioned other attributes such as connectivity and scalability, which are not included in ISO/IEC 25010 model. We observe that participants considered 14 quality attributes, as summarized in Table 5.10. While the participants did not specifically mention functional suitability and maintainability, we believe these qualities are also relevant and important for IoT systems.

Performance is the most popular attribute among participants (63.6%), while other attributes such as connectivity (50.0%), interoperability (40.9%), security (40.9%), integration (38.6%), availability (36.4%), and functionality (31.8 %) are also mentioned. Reliability (29.5%), scalability (22.7%), usability (20.5%), and compatibility (20.5%) are considered, while deployability (13.6%), compliance (9.1%), and robustness (6.1%) are less popular attributes in IoT systems. *We observe that all quality attributes for traditional software systems are considered for IoT systems. IoT systems can have more*

Table 5.10: Quality Attributes Considered for IoT Systems Testing

Quality Attribute	**# Participants**	**% Participants**
Performance	28	63.6%
Connectivity	22	50.0%
Interoperability	18	40.9%
Security	18	40.9%
Integration	17	38.6%
Availability	16	36.4%
Functionality	14	31.8%
Reliability	13	29.5%
Scalability	10	22.7%
Usability	9	20.5%
Compatibility	9	20.5%
Deployability	6	13.6%
Compliance	4	9.1%
Robustness	3	6.8%

quality attributes for measuring the quality of their specific characteristics due to their distributivity and dynamic nature. We conclude that there is a need to study exhaustively all quality attributes of IoT systems.

Takeway for RQ3.2

Fourteen quality attributes from traditional systems are also considered for IoT systems. Performance (63.6%), connectivity (50.0%), interoperability (40.9%), security (40.9%), and integration (38.6%) are the most reported quality attributes by practitioners.

- **Testing Challenges Faced by Practitioners (RQ3.3)** Testing challenges can be defined as issues or problems in IoT systems testing that calls for special effort or dedication[72]. Testing IoT systems have many challenges due to their complexity. we want to understand from practitioners' point of view the main challenges that are affecting the quality assurance of IoT systems. We observe that lack of standards, limited resources, communication protocols, and lack of reference architecture are the top four challenges. Test case generation is reported by 16.3% of the participants. Lack of testing automation tools, lack of testing environment, and adoption of model-based testing are among the top challenges reported. The participants mentioned many other challenges, as highlighted in Table 5.11.

  Many challenges reported by practitioners are also found in the literature. In the existing literature, we identified other challenges such as test coverage analysis and lack of emulators not mentioned by the participants. However, lack of standards, lack of reference architecture, diverse protocols, test case automation, and testing environment are common challenges reported by both academia and practitioners.

Table 5.11: Challenges for IoT Systems Testing

Challenges	**# P**	**% P**
Lack of standards	14	28.6%
Limited resources	12	24.5%
Communication protocols	11	22.4%
Lack of reference architecture	8	16.3%
Test cases generation	8	16.3%
Big data	8	16.3%
Test automation tool	7	14.3%
Integration testing	7	14.3%
Testing environment	6	12.2%
Model-Based testing	6	12.2%
Connectivity issues	5	10.2%
Acceptance testing	5	10.2%
Interoperability	5	10.2%
Continuous testing	5	10.2%
Dynamic nature of IoT	5	10.2%
Hardware testing	5	10.2%
Test coverage	4	8.2%
Unit testing	4	8.2%
Testing Approach	3	6.1%
System testing	3	6.1%
Regression testing	3	6.1%
Performance testing	1	2.0%
Availability of sensors	1	2.0%
Compatibility testing	1	2.0%
Adequate semantic information	1	2.0%

① **P**: Participants.

Takeway for RQ3.3

Twenty-five challenges are identified by the participants. Lack of standards is the most reported challenge, with 28.6% of the participants.

- **IoT System Testing artifacts Automation (RQ3.4)** Testing artifacts can be defined as items generated during the testing process. Good examples include test cases and test reports[112, 183, 97]. Testing artifact automation can be defined as the use of software tools or scripts to produce the test artifacts. The participants suggested different testing artifacts for automation. We observe that test cases and test data automation are the

top-recommended artifacts for automation. Other artifacts mentioned by the participants include testing reports and testing strategies. In the existing literature, we also observed that test cases and test scripts are prevalent. In the literature and industry survey, both test scenarios, defect reports, and test plan automation are not mentioned.

In the existing literature review, we identified that test case generation is among the top challenges. It is also the most testing artifact discussed by many authors [66, 118, 141, 143, 206]. We believe that test case generation in IoT systems testing is still among the top challenges for many practitioners, and thus creates the need for an automated approach to generate test cases for IoT systems.

Takeway for RQ3.4

Four testing artifacts are identified for IoT systems. Test case is the most recommended artifact for automation by many participants (90.9%).

5.3.2 RQ4: How do testers make decisions in IoT system testing?

The questions we ask interviewees are based on the answers provided during the survey. We ask each interviewee the following questions:

- **Q1**: What can be done to overcome testing challenges?
- **Q2**: Why some test artifacts must be automated ahead of others?
- **Q3**: Why do you consider some test coverage and not others?
- **Q4**: At what levels and layers the quality attributes are tested?
- **Q5**: Which testing approach do you use? Why?
- **Q6**: How are test metrics selected for user acceptance?
- **Q7**: How are bugs identified in IoT systems?
- **Q8**: What impact has the lack of standards, reference architecture, and multiple communication protocols had?

Table 5.12 shows the detailed facts extracted from interview data. We used those facts to

Table 5.12: Interviews Outcomes.[1 : **Agree**. **0**: **Disagree**]

Groups	Category	P1	P2	P3	P4	P5	P6	P7	P8	P9	Total	%
Q1 Outcomes	Improve Security	0	1	1	1	1	0	1	0	0	5	**60.0%**
	Developing Standards	1	0	0	1	0	1	0	0	0	3	40.0%
	Test case automation	1	1	0	0	0	1	0	0	0	3	40.0%
	APIs	0	0	1	0	0	0	1	0	1	3	40.0%
	Testing approach	0	1	0	0	0	0	1	1	0	3	40.0%
	Heterogeneity testing	1	0	1	0	0	0	0	0	0	2	30.0%
	Testing framework	0	0	1	0	1	0	0	0	0	2	30.0%
	Testing Environment	0	0	0	0	0	0	0	0	1	1	20.0%
Q2 Outcomes	Reduced time to test	1	1	0	1	1	1	1	1	1	8	**90.0%**
	Reduced errors	0	0	1	0	1	1	0	0	1	4	50.0%
	Improved robustness	0	1	1	0	0	0	0	0	0	2	30.0%
	Minimized efforts	0	0	0	1	0	0	0	0	0	1	20.0%
	Increased productivity	0	0	0	0	0	0	0	0	1	1	20.0%
Q3 Outcomes	Requirements check	0	1	1	1	1	0	1	1	0	6	**70.0%**
	Functionality check	0	0	1	0	0	1	1	1	1	5	**60.0%**
	Company policy	1	0	0	0	0	0	0	0	0	1	20.0%
	Project's evaluation	0	1	0	0	0	0	0	0	0	1	20.0%
	Customer expectations	0	0	0	0	1	0	0	0	0	1	20.0%
	Boost confidence	0	0	0	0	0	0	0	0	1	1	20.0%
Q4 Outcomes	Multilayers and levels	1	0	1	1	1	1	0	1	0	6	**70.0%**
	System level	0	1	0	0	0	0	1	0	1	3	40.0%
Q5 Outcomes	System navigation	0	0	0	1	1	1	0	0	1	4	50.0%
	Simulation-based app.	1	1	0	1	0	0	0	0	0	3	40.0%
	Model-based app.	0	0	1	0	0	0	0	1	0	2	30.0%
	Static analysis app.	0	0	1	0	0	0	0	0	0	1	20.0%
	Adhoc approach	0	0	0	0	0	0	1	0	0	1	20.0%
Q6 Outcomes	Case by case	1	0	1	1	1	1	1	1	0	7	**80.0%**
	Need for New metrics	0	1	0	0	0	0	0	0	0	1	20.0%
	Defects detected	0	0	0	0	0	0	0	0	1	1	20.0%
Q7 Outcomes	Log analysis	0	0	0	1	1	1	1	1	0	5	**60.0%**
	Simulation	1	0	0	0	0	0	0	0	1	2	30.0%
	IoT Device Mgmt	0	1	1	0	0	0	0	0	0	2	30.0%
	Fault injection	1	0	0	0	0	0	0	0	0	1	20.0%
Q8 Outcomes	Communication issue	0	1	1	1	0	0	1	1	1	6	**70.0%**
	Security issue	0	0	0	1	0	0	1	1	1	4	50.0%
	No impact	0	0	0	0	1	1	0	0	0	2	30.0%
	Know. of protocols	1	0	0	0	0	0	0	0	0	1	20.0%

P1: Participant 1;P2: Participant 2; etc.

find answers to the three research questions ($RQ4_i1$, $RQ4_i2$, and $RQ4_i3$).

- **Selection of testing approaches, levels, coverage, and metrics (RQ4.1)**
  - **Selection of Testing Approaches** During our survey data analysis, we learn that many

participants responded that they do not follow any specific approach while testing IoT systems. This observation prompted us to ask those who accepted to be interviewed, to elaborate more on how they test IoT systems. 44.4% mentioned that they mainly navigate throughout the system, while 33.3% mentioned the use of simulation. They also use model-based approaches and static analysis. One participant mentioned ad-hoc testing to which there is no specific or known approach used. We realize that many practitioners may lack knowledge about different testing approaches, which could explain the absence of specific selection criteria in their responses. *Some IoT developers prefer to navigate through the real systems while testing, while others use a simulation approach. Model-based and static analysis are also preferred. However, the absence of specific selection criteria highlights the necessity to develop a comprehensive taxonomy encompassing all testing approaches, which can be widely adopted by IoT professionals.*

– **IoT Layers and Levels for Quality Attributes Test** During the survey, we identified many quality attributes from the traditional systems that are being considered for IoT systems. We wanted to hear from the participants at what layer and levels those attributes are used. 66.6% of the participants mentioned that those attributes are used for many layers. Some examples include security, which is tested on the device, network, and application layers, whereas performance is mostly checked on the device and application layers. They also mentioned that those attributes can be tested at different levels, mainly unit testing and system testing. However, the decision to choose specific testing levels (unit, integration, system, or acceptance testing) is typically driven by project type and customer requirements, rather than specific selection criteria. *Some quality attributes such as security and performance are tested at multiple layers and at different levels of testing. Other quality attributes are checked at the system level when all components have been integrated. The choice of testing level is determined*

based on the nature of the project or customer requirements. We thus believe that there is a necessity to have proper guidance on quality attributes for IoT systems and how to test them.

- **Selection of Testing Metrics** We collected many metrics from the survey. Most of the participants mentioned two metrics only, and we wanted to know if those metrics are enough or not. 77.8% (7 participants) agree that the choice of metrics depends on the project. 11.1% (1 participant) used more than two metrics. *Requirements coverage, test case coverage, number of defects detected, performance, response time, and functional coverage are the top-recommended metrics for testing the entire IoT system. At the component level, code coverage is required for the application layer, while security is mostly needed for the device, application, and edge layers. We observe that many metrics can be preferred for one IoT system, but no specific metrics are common for every system. It is therefore necessary to have proper guidance for selecting the right metrics.*
- **Selection of Test Coverage** From the survey data, most of the participants do not use common test coverages. We asked those who accepted to participate in our online interview to detail why they consider those test coverages and ignore the others. 66.6% (6 participants) mentioned that they want to ensure that all the requirements have been implemented and tested, while 55.5% (5 participants) highlighted the need to check the functionality of the system as the reason. *We realized that requirements verification and functionality check are the main reasons for IoT developers to decide which test coverage to use while testing IoT systems.*
- **Root Cause Analysis** During the survey, many participants mentioned the use of SpiraTeam for managing bugs. We want to understand how developers identify the cause of the bugs. We ask the participants in our interviews to explain how they identify the cause of the bugs and the tools they use. 55.6% (5 participants) mentioned the use of

log analysis, while 22.2% rely on simulation techniques. AWS IoT and fault injection are also mentioned. *The participant P6 commented that "To understand the root cause of bugs, we analyze the logs to understand the behavior of the system and identify any abnormal or unexpected events. In our logs, we record all system activities, capturing information such as errors, warnings, and other relevant data". Log analysis is the most frequently used approach to identify the cause of bugs in IoT systems. Simulation and fault injection techniques are also used, while other practitioners use the AWS platform for monitoring IoT systems. This observation highlights the need for more tools to help practitioners to track the root cause of bugs.*

Takeway for RQ4.1

Testing metrics, test coverage, testing levels, and layers to test are decided on a project basis. Most of the participants use log analysis to identify the root causes of bugs. System navigation and static analysis are approaches that are used by practitioners for testing IoT systems. However, no specific criteria for selecting those approaches.

- **Recommended Research Community Contribution (RQ4.2)**
  - **Lack of IoT Standards and Reference Architecture** In our survey, 24 participants mentioned having been impacted by the lack of standards, lack of reference architecture, and the use of many protocols. We asked the interviewed participants to explain what kind of impact they faced. 66.7% (6 participants) mentioned communication issues between different devices, especially with different message formats, while 44.4% (4 participants) also mentioned security issues because some of the IoT devices can be compromised. *The lack of standards introduces communication and security issues in IoT systems and has a negative impact on testing. This highlights the need for standards in IoT.*

– **Actions to Overcome Testing Challenges** We ask all nine interviewees to propose research action. The interviewees are allowed to give more than one suggestion. 55% (5 interviewees) suggested security improvement in IoT, while 33% (3 interviewees) recommended focusing on the development of standards, test cases automation, API development for IoT devices, and testing approaches. Heterogeneity testing and testing framework both scored 22% (2 participants), while the testing environment got 11% (1 participant). *Security, standards, and test cases automation are among the top concerns IoT developers are still facing. APIs for smart devices and approaches for exhaustive testing are also mentioned among the concerns raised by IoT systems developers. We can conclude that security is still the main concern for many IoT developers, along with standards and test case automation.*

Takeway for RQ4.2
The participants proposed the research focus to overcome testing challenges. Improving security (55.0%) is the top recommendation. Standards (33.0%) and test case generation (33.0%) are also among the top recommendations from participants.

- **Testing artifacts Automation Prioritization (RQ4.3)** All survey participants proposed some testing artifacts to be automated. We requested the participants who accepted to have an interview with us, to explain their rationale when deciding on those testing artifacts. The participants were allowed to provide more than one reason. 88.8% (8 participants) mentioned a reduction of testing efforts (both time and money), while 44.4% specified the reduction of errors that can occur in manual processes.

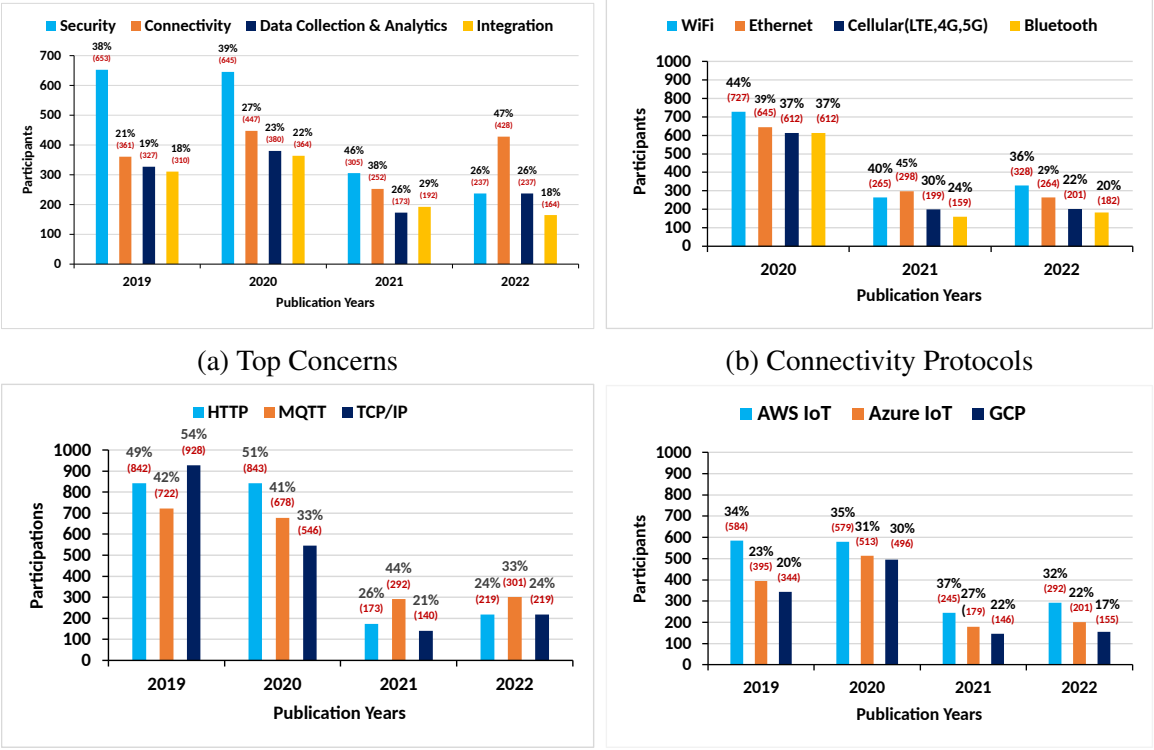
  Automation of testing artifacts such as test cases and test report results not only in reducing the efforts and resources required to test, but also reduces errors.

Takeway for RQ4.3

Prioritization of testing artifacts' automation is based on expected benefits such as reducing errors in the final system, increasing test coverage, and reducing testing effort and resources.

5.3.3 RQ5: What are the trends in IoT systems testing?

After our own survey, we also mine the insights from EclipseIoT surveys to understand the top testing challenges, communication protocols, connectivity technologies, and IoT middleware.



(a) Top Concerns

(b) Connectivity Protocols

(c) Communication Protocols

(d) Cloud Platforms

Figure 5.5: EclipseIoT Survey Statistics

- **IoT Systems Testing Challenges (RQ5.1)** Fig.5.5a shows the developers' top challenges. Security has been the most popular concern from 2019 until 2021. Although

the number of concerned developers decreases in 2022, security remains among the top three concerns. Moreover, the percentage of concerned developers on connectivity keeps growing over the years. Data collection and integration are also among the top concerns. Performance, privacy, and standards are also mentioned among other concerns [78, 79].

We believe that the continual increase in connectivity concerns highlights the challenge of finding the right technologies for IoT systems. The continual increase in integration concerns underscores the lack of APIs, and standards for many IoT devices.

Takeway for RQ5.1

Connectivity and security are the top challenges in four different surveys [78, 79, 80, 81]. Data collection and integration challenges are new challenges not identified in the literature or in our primary survey.

- **Top Communication Protocols and Connectivity Technologies (RQ5.2)**
  - **Communication Protocols** From the previous literature review study, we observed that connectivity issue is among the top challenges. We also observed the continual increase of connectivity concern in this survey since 2019. To deepen our understanding of connectivity issues, we extracted and collated data on communication protocol usage. Fig. 5.5c presents the most widely used IoT communication protocols. MQTT is the most widely used IoT communication protocol in 2021 and 2022. We observed a constant decrease in HTTP/HTTPS usage from 2020 to 2022. TCP/IP has seen a noticeable decrease since 2019 despite a slight increase of 3% from 2021 to 2022. Websocket is mentioned only in the 2019 survey [78], but its usage is low (26%) compared to HTTP (49%), MQTT (42%), and TCP/IP (54%). No data are collected on websocket since 2020.

    We observe that MQTT is the leading protocol for IoT communications because it tolerates intermittent connections and reduces network bandwidth needs [182].

– **Top IoT Connectivity Technologies** We use the connectivity data for three years (2020 [79], 2021 [80], and 2022 [81]) to understand the most widely used connectivity technologies. We did not find connectivity data in the 2019 survey [78]. Fig. 5.5b shows the trend in IoT connectivity technologies. We observed that the top connectivity technologies used in IoT are WiFi and Ethernet. Cellular (LTE, 4G, 5G) and Bluetooth are also considered, despite their continual decline in their usage over three consecutive years. Based on identified challenges, we can observe that it is difficult for developers to determine the best connectivity technology to use. The results from the 2020 survey show that WiFi is the most used at 44% followed by Ethernet at 39%. However, the results of the 2021 survey show that Ethernet is the most used with 45%, followed by WiFi at 40%. *We observe that WiFi and Ethernet are the top connectivity technologies used in IoT.*

Takeway for RQ5.2

MQTT is the most popular communication protocol in 2021 [80] and 2022 [81]. Since 2020 [79], HTTP and MQTT are the top two leading protocols. WiFi and Ethernet are the two top technologies used for connection in all three surveys [79, 80, 81].

- **Top IoT Cloud Platforms (RQ5.3)** In our interviews, interviewees mentioned the use of IoT cloud platforms for testing, monitoring, and troubleshooting IoT systems. In EclipseIoT surveys, the developers provided data on the IoT platforms they used. We analyze the data to find the available IoT platforms. Fig. 5.5d shows the top three cloud IoT platforms. Amazon AWS IoT is the most used IoT platform, followed by Microsoft Azure IoT. Google Cloud IoT Platform is the least used among the three platforms. Other platforms such as Bosch IoT suite and IBM Watson IoT platform are also used, but their usage is low [81].

IoT platforms allow interactions among connected devices with cloud applications and other devices. They support HTTP, WebSockets, and MQTT protocols. For example, AWS IoT Device Simulator enables developers to create and simulate hundreds of virtual devices, without having to configure and manage physical devices [167].

Amazon AWS IoT is the most used IoT platform for testing, monitoring, and troubleshooting IoT devices.

Takeway for RQ5.3

Three top IoT middleware (AWS IoT, Azure IoT, and GCP) are identified. Amazon Web Service (AWS) IoT is the most popular among the developers in four surveys [78, 79, 80, 81].

5.3.4 Findings Analysis Insights

We relate the answers to our research questions in this chapter. We also relate the answers with the existing literature. Table 5.13 shows the relationship between the answers within this chapter and the existing literature. *We conclude that the answers to research questions in this chapter identified common challenges, testing levels, testing layers, and testing approaches. We did not find any contradicting answers, but we observed the complementarity among those answers.*

5.4 Discussions

We observe that there are many tools reported by academia in existing literature review that are not mentioned by practitioners such as Héctor, Izinto, InterOpT, F-interop, PatrIoT[3], Fit IoT-Lab[4], and CupCarbon[5]. Other tools are reported by the practitioners but

[3]https://patriot-framework.io/
[4]https://www.iot-lab.info/
[5]https://www.cupcarbon.com/

Table 5.13: Relationship Between the Answers

Survey	Interview	EclipseIoT	Relate Answers	Relate to Prior Chapter
RQ3.1: Testing approaches	RQ4.1: Testing approaches selection	-	**Survey Findings**: • Component-based approach is more popular. • Model-based and test-driven development are also popular. **Interview Findings**: • System navigation and static analysis are two approaches not mentioned during the survey.	• Model-based approach is popular in literature and among practitioners. • System navigation and static analysis approaches are not mentioned in the literature review.
RQ3.2: Quality attributes	RQ4.1: Quality attributes testing	-	**Survey Findings:** • Fourteen quality attributes are mentioned for IoT systems. **Interview Findings:** • Quality attributes are tested at different layers (device layer, network layer, cloud layer, and application layer). • Quality attributes are tested at different levels (unit, integration, system, and acceptance testing).	• IoT layers to consider when testing IoT systems, different levels of testing, and quality attributes considered for IoT systems testing are identified in both literature and among practitioners. • This chapter shows that the quality attributes can be tested at different layers of IoT and at different levels of testing.
RQ3.3: Testing artifacts automation	RQ4.3: Automation prioritisation	-	**Survey Findings**: • Test case, test data, test report, and test strategy are testing artifacts identified. • Many practitioners recommended automation of test case generation. **Interview Findings**: • Reduction of time to test, reduction of errors, improved robustness, minimization of repetitive efforts, and increased productivity are top criteria for automation prioritization.	• Test case automation challenge is common in the literature and among practitioners. • We argue that test case generation for IoT systems deserves exhaustive study to address the needs of many practitioners.
RQ3.4: Testing challenges	RQ4.2: Overcoming challenges	RQ5.1: Practitioners' challenges	**Survey Findings**: • Lack of standards, limited resources, communication protocols, lack of reference architecture, and test case generation are top identified challenges. **Interview Findings**: • Improving security, development of standards, and test case generation are top recommended areas of focus for the research community. **EclipseIoT Surveys Findings**: • Security, connectivity, data collection, and integration are the top challenges reported by the developers. • Lack of standards can have a negative impact on security, connectivity, and integration.	• Lack of standards, lack of reference architecture, security testing, connectivity issues, testing approaches, and test case generation are common challenges in the literature and in this chapter. • Deployment, monitoring, debugging, heterogeneity test, and privacy testing are challenges identified in the literature but not mentioned in this chapter.

not reported in the literature such as ThingSpeak[6], ThingBoard[7], and JEst[8]. We argue that

[6]https://thingspeak.com/

[7]https://thingsboard.io/

[8]https://www.jest.it/tag/iot/

collaboration between academia and industry can take full advantage of the tools available from both sides and hence work together to enhance them. We also observe that the most important metrics considered for IoT systems testing such as response time, code coverage, requirements coverage, and the number of defects are the same for both practitioners and academia. We identify some metrics such as the number of mutants killed, number of requests per second, throughput, and number of defects, reported in the literature, but not mentioned by practitioners. Likewise, we identify other metrics such as the number of active clients, number of connected devices, and packet loss, mentioned by practitioners but not mentioned in the literature. We further notice that practitioners often mix up testing metrics with quality attributes. Performance, scalability, reliability, security, and accessibility are all mentioned among the metrics. However, the practitioners need the metric to assess those quality attributes instead of using them as metrics. Exhaustive studies on IoT testing metrics can help both industry and academia to choose the right metrics to consider when testing IoT systems. Testing levels such as unit testing, integration testing, system, and acceptance testing are covered by both academia and industry. Integration and unit testing are reported as the most considered testing levels by both academia and industry. Based on our previous review of the literature [67, 96], system testing is the least considered testing level by scholars, whereas acceptance testing is the least considered among practitioners. We believe that acceptance testing in the industry could be the most important test to ensure that the system is checked against its requirement specifications across all the layers. Both industry and academia suggested that the test case generation process is the most recommended task for automation. Test case automation can reduce the time required to test, reduce errors in the system, increase test coverage, improve robustness, and minimize repetitive efforts when testing. We believe that there is a need for exhaustive studies of test case automation in IoT systems for end-to-end testing. From both academia and industry, challenges are reported while testing IoT systems. Many challenges such as

deployment, debugging, monitoring, heterogeneity test, and reusability tests are reported by academia but not mentioned by practitioners. New challenges such as *data collection* and *integration challenges* are identified in this chapter but are not identified in the existing literature. Usability and availability are two quality attributes in IoT systems reported by practitioners, but not identified in the existing literature addressing IoT systems testing. However, all other quality attributes mentioned in the literature [8], are also mentioned by the practitioners. We argue that the list of identified quality attributes is not exhaustive, and therefore, a further study on IoT quality attributes could provide better guidance to both practitioners and researchers. Lastly, testing approaches, test-driven development, simulation, requirements, and model-based testing are commonly mentioned by both researchers and practitioners. Some approaches such as fault injection, fuzzy testing, fault tree analysis, pattern-based, mutation-based, and combinatorial testing are identified in the literature but are not mentioned by practitioners. Static analysis and system navigation approaches are mentioned by the practitioners, but not identified in the literature. We believe that an exhaustive study on IoT systems testing approaches is needed to find all possible testing approaches that can be used for testing IoT systems.

We learned several lessons that can guide and inform future studies. 1. Our analysis reveals a notable disconnect between the industry and academia. While various testing tools and approaches have been mentioned in the literature, it is surprising to note that many practitioners who participated in our survey were not familiar with them. This underscores the need for a good collaboration between industry and academia. Such collaboration can enable industry practitioners to adopt the most effective practices recommended by scholars, who can better address the needs of the industry. 2. There are several developer tools proposed in the existing literature, such as testbeds, emulators, and simulators. However, the availability of testing tools for end-users remains scarce. Moreover, the existing tools

for end-users address a few testing aspects, such as performance, connectivity, or interoperability. Our findings underscore the pressing need for more tools to test IoT systems, as well as an improvement for the existing tools. 3. Despite the availability of various testing approaches, it is surprising to note that many practitioners do not adhere to any approach when testing IoT systems. This may indicate that the existing approaches do not entirely fulfill their requirements. We believe that the development of effective testing approaches is necessary to cater to the needs of industry practitioners. 4. While scholars and practitioners proposed various quality attributes and metrics, our study has revealed some quality attributes that may be relevant to IoT systems not commonly known by practitioners. There is a need for a comprehensive study on quality attributes and metrics that can guide both practitioners and scholars in IoT systems testing. 5. Practitioners and scholars may use testing concepts interchangeably, which can lead to confusion due to different interpretations. For instance, testing metrics, such as reliability or availability, are sometimes referred to as testing types by some scholars and practitioners. The findings of this chapter have been published in the IEEE Internet of Things Journal [125].

Takeaway from literature and industry study

The insights from our literature review and industry study highlight that while various testing approaches exist for IoT systems, they often focus on isolated aspects rather than providing comprehensive, functional end-to-end testing. The absence of standardized testing guidelines, challenges in test case generation, and difficulties in handling device diversity, interoperability, real-time complexities, and access to real devices further complicate the testing landscape. Recognizing these limitations, the next chapter presents a taxonomy for IoT system testing. This taxonomy is designed to serve as a structured reference for practitioners, helping them navigate the complexities of IoT testing by categorizing essential aspects of the testing process. By providing a clear classification of testing objectives, techniques, and challenges, the taxonomy serves as a guide to evaluate and improve the quality of IoT systems.

5.5 Threats to validity

This section summarizes internal, external, and conclusion threats to the validity of this chapter.

Internal Validity During the survey design, we defined twenty questions and grouped them into three sections. We designed the initial survey and conducted an internal review based on our research questions. We used internal review feedback to improve the survey design before sending it out to the practitioners. Despite conducting an internal review, it is possible that some options are omitted unwillingly from the list of possible answers to some questions. To minimize this threat, we provided an option for the practitioners to provide their own answers if not listed among the provided options. On the distribution channels to reach out to the practitioners, we tried our best to reach as many practitioners as possible. Initially, we visited the websites of vendors of IoT solutions and device providers

to find information about practitioners. We also reached out to IoT research labs and requested them to distribute our survey to the practitioners connected to those labs. However, this was limited only to a few research labs. To minimize the threat of missing some practitioners, we used alumni groups of some universities. In this case, we used four alumni groups namely: Carnegie Mellon University, USA; Concordia University, Canada; Vellore Institute of Technology, India; and COMSATS University Islamabad, Pakistan. We also used three social media websites: Facebook, Twitter, and LinkedIn to get more practitioners based on their profiles. This helped us to get more IoT practitioners that we could not identify by other means. Regarding the position of participants, we were aware that some of the IoT companies may not have specific roles for IoT systems testing. To minimize this threat, we targeted more job titles that we believe at some point are involved in IoT systems testing. We knew that participants may not have enough experience and have biased answers due to the lack of knowledge. We mitigated this threat in three ways: 1. we did not provide any incentives for practitioners to participate, 2. we allowed the participants to skip the question if they do not have enough knowledge to answer that question and, 3. we conducted interviews with the participants who agreed to be contacted. We analyzed the data manually using Excel for clear data validation. There is a possibility to have some errors in our analysis. Therefore, we conducted different review sessions among team members to check the accuracy of the analysis.

External Validity We collected 49 responses from the survey participants, which might not be a good representative sample. Therefore, the results are not mineralizable. However, we believe that this is acceptable because getting answers from many IoT practitioners is not an easy task. Because, IoT is an emerging domain in its embryonic stage, some of those involved may not accept to participate in surveys and interviews. To the best of our knowledge, this is the most we could achieve in terms of the participants. We observed more participants in the EclipseIoT survey, but the information provided could

not be used for all of our research questions. We believe that our sample is still reliable that helps to provide results based on the answer of 10 practitioners having more than 5 years of experience in IoT systems. Moreover, we also have a big sample size for EclipseIoT surveys, with at least 600 participants per year.

Conclusion Validity The information from our study may show some threats to the validity of our conclusion because either some facts are not provided in the participant's responses or the participants gave contradicting or ambiguous answers. However, this threat is acceptable because we use the interviews for the purpose of mitigating and discussing the answers of participants.

5.6 Conclusion

We presented in this chapter, a multi-method study of IoT systems testing in industry with IoT practitioners. We used three methods: 1. a survey with 49 practitioners about testing IoT systems, 2. interviews with 9 practitioners about testing decisions, and 3. an analysis of the data from the four surveys conducted by EclipseIoT with IoT systems developers. We defined four research questions pertaining to IoT systems testing for the survey, three for the interviews, and three for EclipseIoT surveys. We analyzed the extracted data to find answers to the research questions. We provided the following contributions:

1. Identified the main challenges faced by IoT practitioners when testing IoT systems to guide future research.
2. Compiled the top quality attributes considered for IoT systems.
3. Identified testing tools available, testing approaches considered, testing metrics selected, test coverage used, testing levels considered, and the most tested layers in IoT systems.
4. Provided top artifacts to consider for IoT system testing automation.
5. We summarized top IoT protocols, technologies, and IoT middleware.
6. Discussed lessons learned to guide future work.

We reported that 1. testing focuses more on the device, network, and application layer. Integration testing is the most considered testing level, whereas acceptance testing is the least considered. Test coverage is the top metric for IoT system testing, and the choice of metrics varies based on the project. 2. Model-based approach is popular for IoT system testing. IoT system testing is still manual or semi-automated, whereas the adoption of white box testing is low. Node-RED is the most used tool in testing IoT systems, while Amazon AWS IoT is a popular cloud platform for testing IoT devices. 3. Log analysis is the main approach to analyzing the root cause of bugs. Log analysis provides valuable insights into a system's behavior and allows for identifying abnormal or unexpected events. Logs are records of system activities, capturing information such as errors, warnings, and other data. By analyzing these logs, developers can trace the sequence of events leading to a bug, identify the specific conditions or interactions that triggered it, and pinpoint the underlying causes [40]. 4. Top challenges in IoT systems testing include lack of standards, security, connectivity, and lack of reference architecture. Test case generation and standard approach for IoT systems testing are the top-recommended research focus.

Chapter 6

Taxonomy for IoT Systems Testing

6.1 Introduction

In our previous chapters [85, 127], we identified the lack of a structured testing guide as a major challenge in IoT systems testing, which hinders quality assurance processes. To address this gap, we propose a taxonomy that categorizes different aspects of IoT systems testing, serving as a structured guide for testers. Taxonomies have been recognized as valuable tools for improving the understanding of various testing aspects. [130, 188] suggested that a well-defined taxonomy can enhance testers' comprehension of the system by systematically organizing key elements. The taxonomy we propose in this chapter aims to serve as a technical guide, enabling IoT testers to approach testing comprehensively and systematically, reducing the risk of overlooking critical aspects. To the best of our knowledge, no prior taxonomy exists to address the specific needs of IoT systems testing.

To develop this taxonomy, we conducted a systematic review of 83 IoT testing-related primary studies (PSs) retrieved from eight digital libraries. We followed established guidelines for taxonomy development as suggested by [98, 158, 185]. Additionally, we refined and validated the taxonomy through surveys with 204 IoT practitioners and assessed its effectiveness through empirical evaluations involving two case studies with 12 testers each.

In this chapter, we answer the following the research questions (RQs):

- **RQ6:** How can a taxonomy define aspects of IoT system testing?
- **RQ7:** How do practitioners evaluate the taxonomy?
- **RQ8:** How does the taxonomy improve test coverage?

The objective of this taxonomy is twofold: 1. To offer a structured framework for organizing and understanding key dimensions of IoT systems testing, providing guidance to practitioners. 2. To serve as a reference for relevant IoT testing concepts, assisting testing teams in working more efficiently and effectively. The proposed taxonomy categorizes and organizes IoT testing aspects to improve testing, benefiting researchers, practitioners, and future developments in the field. The main contributions of this chapter are:

- Develop, validate, and refine the taxonomy through collaboration with practitioners.
- Conduct an empirical evaluation with two case studies and 12 practitioners to assess the effectiveness of the developed taxonomy.
- Provide recommendations on prioritized testing types tailored to each layer of IoT systems, informed by the insights of industry practitioners.

The rest of this chapter is organized as follows: Section 6.2 describes the methodology we used. Section 6.4.2 discusses practitioner feedback. Section 6.4 elaborates on the taxonomy and answers our RQs, while Section 6.4.3 presents the empirical evaluation findings. Section 6.3 outlines key recommendations. Section 6.5 provides an in-depth discussion. Section 6.6 highlights potential threats to validity. Section 6.7 concludes this chapter.

6.2 Research Method

The method used in this chapter comprises three phases, as shown in Figure 6.1 (i.e., taxonomy development, feedback collection, and empirical evaluation).

6.2.1 Development

In this section, we explain the steps to construct the taxonomy.

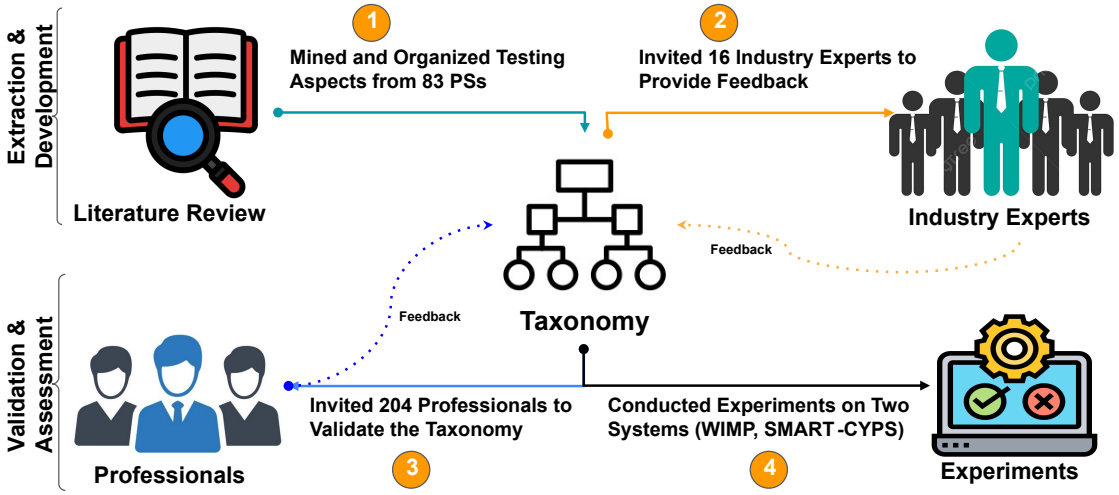


Figure 6.1: Research Method

Step 1 – Identification of Relevant Literature

We followed the Preferred Reporting Items for Systematic Reviews and Meta-Analyses statement (PRISMA) by [145] and retrieved relevant scientific articles published from January 2012 until December 2022. In this section, we have reused some data from our recent study on a systematic literature review (SLR) for IoT system testing, specifically for the identification and selection of primary studies (PSs) [85]. Specifically, we extracted data pertinent exclusively to the development of our taxonomy. We present only the key steps here to focus on the essential aspects of our methodology. Figure 6.2 summarizes the PRISMA process we used for the literature review. We started by defining the search strategy to be executed in different digital libraries. We defined the search strategy by applying the Population, Intervention, Comparison, and Outcome process (PICO) proposed by [33].

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

We selected 8 online digital libraries: ACM Digital Library, Compendex, IEEE Xplore, ScienceDirect, SpringerLink, Scopus, Web of Science, and Wiley. Those digital libraries are the most commonly used for literature reviews in software engineering [44]. We executed the search strategy and retrieved 8,294 articles from these 8 digital libraries. We removed 985 duplications and we obtained 7,305 articles. We screened these using the following inclusion and exclusion criteria.

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

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

Exclusion Criteria. We considered the following exclusion criteria:

- The article has less than 4 pages
- The article is not a primary article
- The article is not related to security testing
- The article has not been peer-reviewed
- The article is a graduate thesis or project report

• The article does not have its full text available online

We chose to exclude PSs about IoT security testing despite the importance of this topic for two reasons. First, this topic deserves its own study and has already been the subject of recent surveys, e.g., [135]. Second, security testing for IoT systems is a vast and complex topic, which would have overshadowed other objectives and increased the complexity and length of this article. We applied these criteria, and we obtained 54 articles.

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

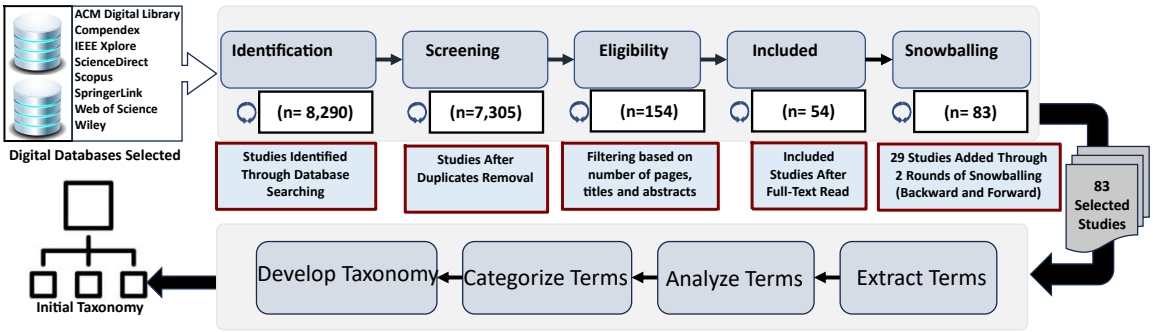


Figure 6.2: Initial Taxonomy Construction Process

Step 2 – Data Extraction and Analysis

Two researchers manually analyzed the selected studies and mined different testing aspects to develop the initial taxonomy, resolving any disagreement on the extracted aspects through discussion sessions to reach a consensus. Table 6.1 shows the form used for data extraction.

Table 6.1: Data Extraction Form

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

The studies analyzed and the data extraction form we used can be found online on ptidej website[1] and Zenodo [2].

Step 3 – Develop the Initial Taxonomy

To develop the initial taxonomy, we followed the method proposed by Usman et al. [185]. This method consists of 13 activities grouped into 4 phases, as shown in Table 6.2.

Table 6.2: Taxonomy Design Method[Adapted from [185]'s study]

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

✺ **SE**: Software Engineering.

1. *Planning.* This step consists of defining the initial aspects of the taxonomy to be developed, such as the objective of the taxonomy, the software engineering knowledge area associated with this taxonomy (i.e., the object to be classified), the data collection method, the classification structure type (e.g., tree or facet-based), and the classification procedure type (i.e., qualitative or quantitative). In this phase, we conducted six activities (B1 to B6). The knowledge area associated with the designed taxonomy is *testing* (the outcome of **B1**). The main objective of the taxonomy is *to identify and classify*

[1] `https://www.ptidej.net/downloads/replications/jss24a/`
[2] `https://zenodo.org/records/14515044`

the testing aspects reported in the existing literature (the outcome of **B2**). The subject matter of the taxonomy is *IoT system testing* (the outcome of **B3**). We chose *hierarchy* as the classification structure (the outcome of **B4**) to relate categories and sub-categories in a parent-child relationship as proposed by [99]. We selected a *qualitative* procedure to classify testing terms (the outcome of **B5**) according to [194]. We used a *systematic literature review* to identify testing aspects (the outcome of **B6**).

2. *Identification and Extraction.* We carried out 2 activities (B7 and B8). We extracted all testing terms from the *83 PSs* (the outcome of **B7**). To extract the testing terms from 83 PSs, we followed the steps in Section 6.2.1 and Section 6.2.1. Activity **B8** enabled us to control the consistency of extracted terms. We performed the following actions to categorize the extracted terms: (1) We removed duplicates, (2) whenever we identified a testing term possibly represented by another term in different studies with the same meaning, we performed terminology unification. We used a Delphi-inspired process [136], where two authors of this chapter independently define the category of the extracted terms, resolving any conflict through discussions until a consensus was reached. As a result, we identified testing aspects shown in Figure 6.3.
3. *Design and Construction.* In this phase, we described the categories of testing aspects. We identified any relevant subcategories for each testing aspect. We used extracted terms and controlled through B7 and B8 to identify and describe taxonomy dimensions and categories. We selected one dimension, named *IoT system testing* (the outcome of **B9** at the very top, making it the parent of the entire taxonomy. We identified *7 aspects (objectives, targets, approaches, timing, environment, role, and tools and metrics*, each representing a distinct category within the taxonomy, as shown in Figure 6.3 (the outcome of **B10**). We used hierarchical classification relationships to relate categories and subcategories (the outcome of **B11**). We provided a guideline through the usage scenario, and we will review the taxonomy every two years to determine if updates are

necessary. (the outcome of **B12**).

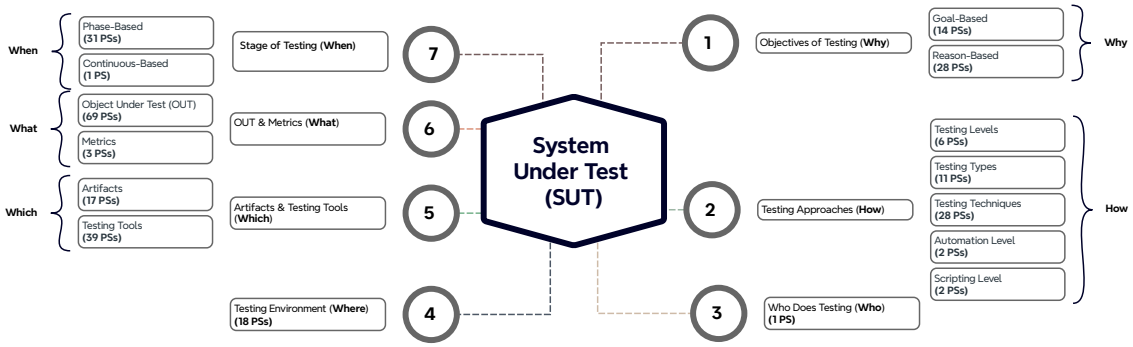


Figure 6.3: Testing Aspects From Primary Studies

4. *Validation.* Usman et al. [185] proposed various approaches for validating taxonomies, including expert opinion, case studies, experiments, and benchmarking. In this chapter, we validated this taxonomy with two rounds of surveys (i.e., expert opinion), as explained in Section 6.2.1 and 6.2.1. We also conducted an empirical study with testers (i.e., case study and experiment), as presented in Section 6.4.3.

Step 4 – Survey Round 1

We surveyed 16 industry practitioners to collect feedback on the initial taxonomy. We used the collected feedback to improve the taxonomy. We summarize the steps we used to conduct the survey as follow:

1. *Identify the Participants*. Due to limited information on the version of the taxonomy we had, we chose not to share it publicly for feedback. Instead, we identified professionals with experience in IoT systems through their LinkedIn profiles. We invited them to participate in our study and provide their feedback on the version of the taxonomy that we constructed based on a literature review. We did not specify a target number of professionals; however, we sent requests via LinkedIn private messages to 79 professionals. Ultimately, 16 practitioners voluntarily agreed to participate.

2. *Design the Feedback Collection Form.* We created a feedback collection form consisting of 20 questions grouped into three sections: demographic information, IoT testing experience, and comprehension of the taxonomy. The form can be accessed online[3].
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.

In this step, we surveyed 16 industry practitioners (12 males and 4 females) from seven countries (UAE, USA, Canada, Japan, Pakistan, India, and Rwanda) to collect feedback on our initial taxonomy. Their IoT experience varied, with 25% having 5 to 10 years, 12.5% having 3 to 5 years, and the majority (62.5%) having 1 to 3 years. Figure 6.4 presents the details of the participants.

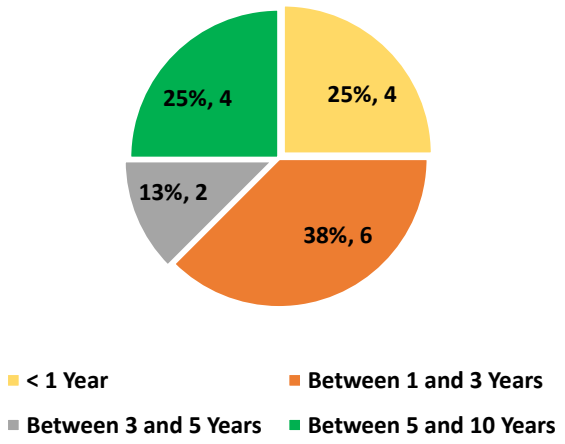


Figure 6.4: Participants'Experience in IoT

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

Step 5 – Survey Round 2

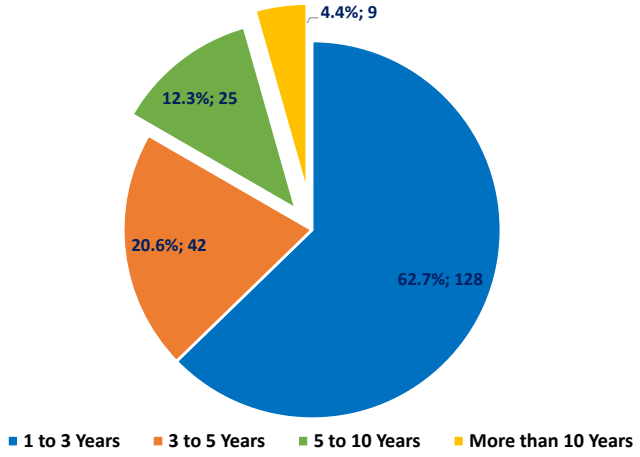
We surveyed 204 practitioners to evaluate various aspects of the developed taxonomy and provide their feedback. We used the feedback provided to improve and finalize the taxonomy. We conducted three main activities: designing the survey, distributing the survey and taxonomy, and analyzing the survey results. We followed the same steps mentioned above.

- *Design the Survey.* We used Google Forms to create our survey. We asked participants questions on completeness, understandability, helpfulness, meeting user expectations, effectiveness, overall impact for practitioners and stakeholders, and participants' opinions on the categories and subcategories. The link to the survey is online [4].
- *Distribute the survey.* We obtained an "Ethics Certificate" for research involving human subjects. At this stage, our taxonomy was more developed, and we were receptive to public feedback. We aimed to gather feedback from at least 100 professionals specializing in IoT systems. We distributed the Survey, along with a copy of the taxonomy, via various social media channels, mainly LinkedIn and Facebook IoT groups. We also contacted alumni associations from VIT[5], CMU[6], UR-ACEIoT[7]. We did not provide any monetary incentives for the participants and requested those willing to participate to read the taxonomy and ask questions, if any, before taking the survey. The survey was conducted over a period of 45 days, during which 204 practitioners agreed to review the taxonomy and provide their feedback.
- *Analyze the Survey Results.* We analyzed the survey data to finalize the taxonomy.

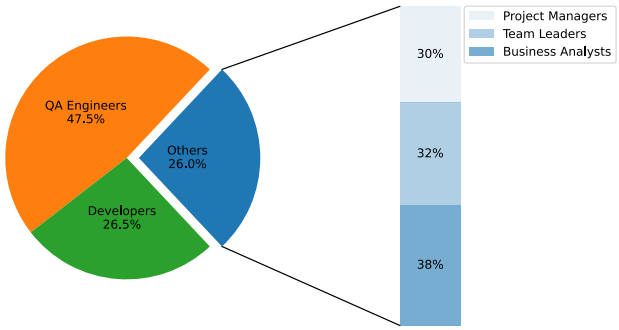
  The materials we used can be accessed from ptidej website[8]. The complete taxonomy can be accessed online [9].

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

In this step, we surveyed 204 industry practitioners, including developers and quality assurance engineers with at least two years of IoT experience, to evaluate the revised taxonomy incorporating feedback from the Round 1 survey. Figure 6.5 shows the details of the participants.



(a) Participants'Experience in IoT



(b) Participants' Professions

Figure 6.5: Participants'Details for Survey – Round 2

6.2.2 Empirical Evaluation

We conducted a case study on two separate systems to evaluate how the taxonomy can improve testers' understanding. We recruited software engineering students from the IoT lab at Concordia University for the WIMP project, and IoT students from the Center of IoT

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

6.3 Practical Guidance and Recommendations

6.3.1 Practical Guidance

Although the taxonomy offers options for each aspect, testers are not required to consider all of them. However, for each category, testers must make decisions based on what to test, the objectives, etc. Figure 6.6 illustrates how testers can navigate the taxonomy.

Figure 6.7 shows a (simplified) example for practitioners to navigate the taxonomy. Although we do not depict all the potential options outlined in the taxonomy, it shows that practitioners can select one of the four identified options for the testing environment to execute their tests. Regarding the testing approach, the scripting level is optional and requires the use of automation tools. It cannot enumerate all conceivable metrics or test automation tools, but guides practitioners in making a choice based on their applicability to their SUT.

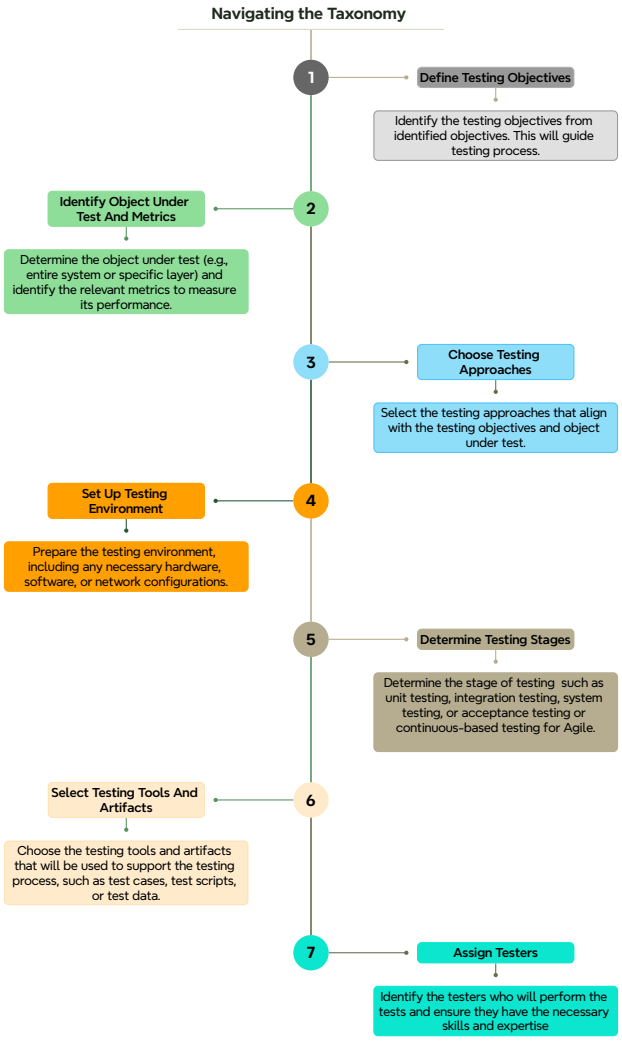


Figure 6.6: Steps to Guide the Practitioners

6.3.2 Recommendations

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

- **Comprehensive Test Planning Frameworks for IoT Systems**. From our experiment, we observed that many testers primarily focus on functionality testing, often overlooking other critical aspects. We strongly recommend the development and adoption of a

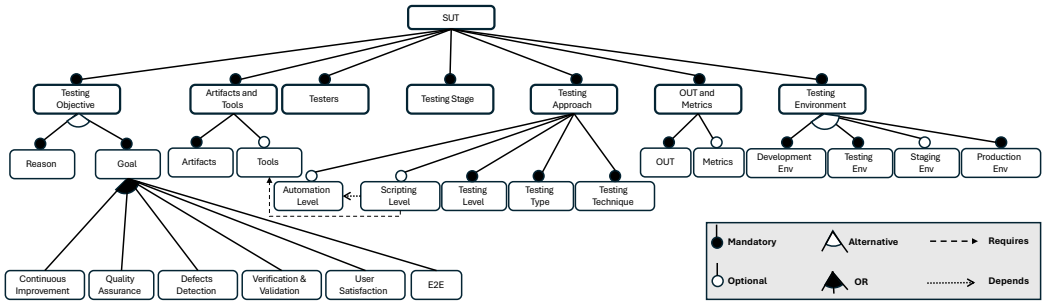


Figure 6.7: Simplified Example for Practitioners to Navigate the Taxonomy

structured test planning framework to address this limitation. Such a framework should explicitly prioritize and integrate diverse testing aspects beyond functionality, including scalability, interoperability, energy efficiency, protocol coverage, and real-time performance. By doing so, testers can ensure a more comprehensive evaluation of IoT systems.

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

- **Leveraging LLMs and AI for Test Automation**. From both the surveys conducted and the reviewed papers, we observed that LLMs and AI are often overlooked in the context of IoT systems compared to traditional systems. We recommend the use of Large Language Models (LLMs) to automate essential testing tasks such as generating test inputs, test oracles, constraints, test cases, and test scripts. These models have been effective in traditional software systems [193] and could be similarly beneficial for IoT systems, which often rely on requirements-based test case derivation. LLMs can reduce manual

effort significantly by using extensive datasets to produce context-aware testing components. Future research should focus on tailoring LLMs to address the unique complexities of IoT systems, assessing their accuracy and efficiency in real-world scenarios.

- **Applying NLP-Based Techniques for Use Case-Driven Test Case Generation**. Based on the papers reviewed and feedback from the surveys, the adoption of NLP techniques in testing IoT systems is very low. Given the limited accessibility of source code in IoT systems, we recommend using NLP techniques to derive test cases directly from use case specifications. This approach has been applied to embedded systems [57, 191] and could similarly be considered for IoT systems, where access to the source code may not be available for black-box testing. Future research should focus on refining these NLP techniques to improve their ability to accurately convert structured use case narratives into executable test scripts for complex systems like IoT systems.

- **End-to-End (E2E) Testing**. E2E testing is critical for ensuring the seamless functionality of IoT systems, encompassing all components from sensors to applications. By evaluating interactions and data flows across the entire system, E2E testing helps to uncover issues that might not be apparent in isolated testing. Future research should focus on developing or enhancing existing approaches for various aspects of E2E testing in a real environment, enhancing the ability to identify and resolve issues across the IoT system. This recommendation is based on insights gathered during this study, particularly on testing objectives. Many participants emphasized the importance of focusing on E2E scenario testing, for both functional and non-functional related aspects. Functional requirement-based E2E testing includes commonly known approaches like acceptance testing, allowing users to verify that the developed system meets their needs and expectations, ensuring readiness for deployment. E2E testing is also applicable to non-functional aspects such as performance, security, usability, and scalability. Testing individual layers alone may not require a new approach or tool. From our findings, we identified

some tools and approaches for embedded systems. These can be used for testing various devices at the device layer. Likewise, we identified approaches and tools for testing web-based, mobile, or desktop applications, and these can be used for testing the application layer. We observed limited approaches and tools for testing the entire system, referred to as E2E. Existing E2E approaches focus on specific aspects, and they are limited in scope. Bosmans et al. [20] proposed a hybrid simulation-based testing approach focusing on interactions of IoT system entities. However, testing in a real environment is desirable because IoT systems may misbehave in a real environment while behaving well in a simulated one. Clerissi et al. [31] proposed an approach based on the behavior of the SUT, such as a state machine diagram. However, for complex IoT systems, producing such a behavioral model may not be feasible or detailed enough for automation of IoT system testing. Kim et al. [92] proposed an IoT testing framework focusing on conformance and interoperability. However, there are several other aspects that are not yet explored from an E2E perspective. While emphasizing that E2E testing is crucial for ensuring the comprehensive functionality of IoT systems, it is equally important not to overlook testing individual layers or the interactions between layers. For practitioners aiming to implement E2E testing effectively, we propose a three-step approach for testing IoT systems.

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

- *Device Layer:* Functional testing verifies device functions such as sensing, processing, and actuating (e.g., sensor accuracy, actuator response, and communication protocols). Non-functional testing checks performance, security, reliability, and battery life.
- *Gateway Layer:* Functional testing focuses on data processing tasks like filtering,

aggregation, and transmission. Non-functional testing addresses reliability, scalability, and network performance.

  - *Cloud Layer:* Functional testing includes data processing and storage validation, while non-functional testing assesses aspects like availability, failover, scalability, and security.
  - *Application Layer:* Functional testing ensures user interfaces and functionalities meet requirements, while non-functional testing targets user experience (UX), security, and performance.

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

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

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

- **Generalizing IoT Testing Taxonomy for other Systems**. Using the taxonomy, the participants created numerous tests that are also applicable to traditional systems. This IoT system testing taxonomy can be adapted to other systems, such as web applications, desktop applications, mobile applications, embedded systems, and more, by omitting

IoT-specific components, providing a flexible framework for diverse systems.

6.4 Results

In this section, we present the description of the taxonomy for IoT system testing, referred to as *TaxIoTe*. We describe each aspect in *TaxIoTe* by answering some of our questions. Due to space constraints, we have omitted certain figures in this section. We mainly focus on testing objectives, tools and artifacts, testing approaches, and what to test. However, the complete set of all figures is available in the published paper.

6.4.1 RQ6: How can a taxonomy define aspects of IoT system testing?

- **What Are Testing Objectives.** We compiled and classified the objectives for testing, as depicted in Figure 6.8.

  We organized the objective of testing based on the reason and goal for testing. On *reason-based*, we identified six reasons: conducting initial testing, retesting, regression testing, patch testing, backup and recovery testing, and Go-Live testing. On *goal-based*, we identified continuous improvement, customer satisfaction, verification and validation, quality assurance, defects detection, battery drainage checking, sensor calibration checking, network coverage and range checking, and geolocation capability checking. It can also focus on end-to-end (E2E) business scenarios. E2E testing can be functional or non-functional requirements-based as shown in Figure 6.8. Due to space constraints, we are unable to provide detailed descriptions for each item in the taxonomy here, but we will provide a comprehensive description of each item on our GitHub repository and ptidej website. We suggest the following explanations for certain objectives. *Continuous improvement* in IoT system testing aims to regularly enhance both functionality and the quality of IoT systems. *User satisfaction* in IoT systems testing ensures that the system meets or exceeds user expectations. *Verification and Validation* are separate processes that both use testing as one of their principal practices. *Verification* is the process of evaluating

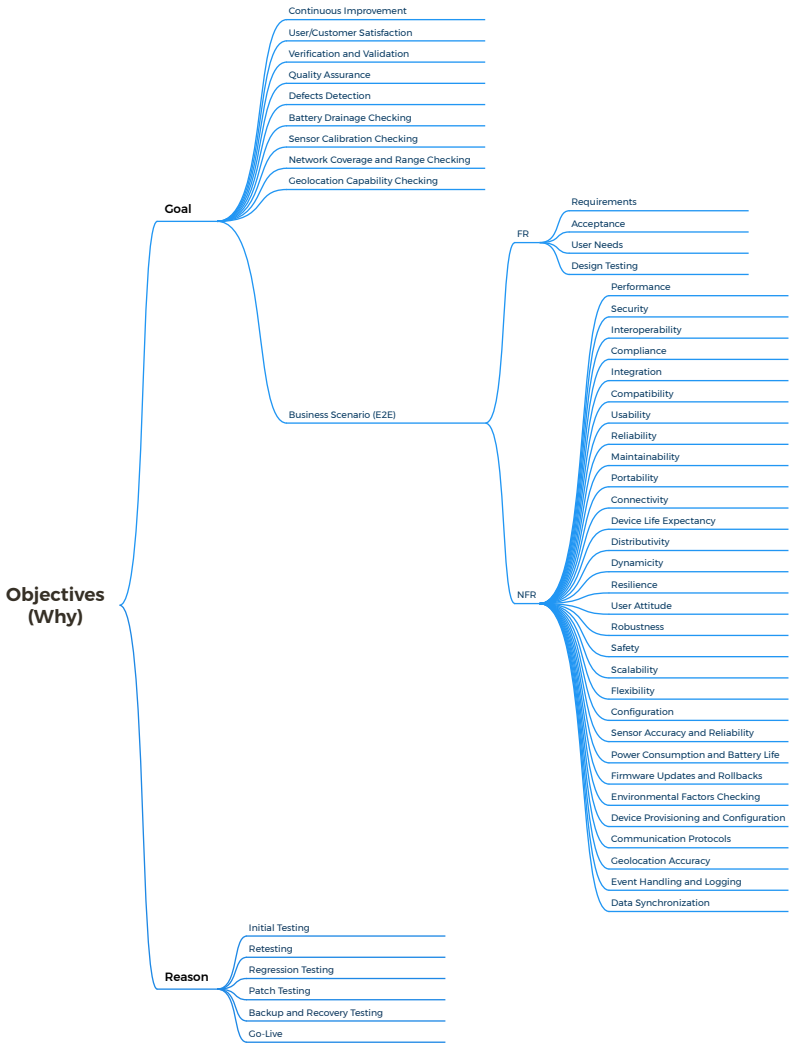


Figure 6.8: Testing Objectives

the product or system to ensure that it meets the specified requirements (i.e., building the product right) [82]. *Validation* is the process of evaluating the product or system to ensure that it meets the user's needs and expectations (i.e., building the right product). *Quality Assurance* is the process that ensures products or services meet quality standards and expectations [82]. *Defects Detection* focuses on identifying errors, bugs, faults, and failures in IoT systems. *Battery drainage checking* ensures that an IoT device's battery life meets expectations and that power consumption is optimized. This involves measuring

battery drain under various usage scenarios. *Sensor calibration checking* verifies that an IoT device's sensors are accurately calibrated to provide reliable and precise data, such as temperature, humidity, or pressure readings. *Network coverage and range checking* ensures that an IoT device can maintain a stable and reliable connection to the network within a specified range and coverage area. *Geolocation capacity checking* verifies that an IoT device can accurately determine its location and provide location-based services, such as GPS tracking or location-based alerts. For functional requirements (FR), we defined the following objectives:

- *Requirement Testing.* The objective is to verify and validate specific requirements and specifications of an IoT system. Test cases (TCs) and test scenarios are designed and executed based on the requirements of the IoT system.
- *Acceptance Testing.* Acceptance testing is owned by the customer to verify that the system works in accordance with the customer's expectations [161].
- *User Need Testing* verifies if the IoT system meets stakeholders' needs.
- *Design-driven testing* verifies if the IoT system conforms to the provided designs.

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

- *Performance Testing* is conducted to evaluate the degree to which a test item accomplishes its designated functions within given constraints of time and other resources.
- *Security Testing* is conducted to evaluate the degree to which the SUT, and associated data and information, are protected so that unauthorized persons or systems cannot read or modify them, and authorized persons or systems are not denied access to them [10].
- *Interoperability Testing* refers to the degree to which the system operates (i.e., interfaces and collaborates) effectively with specified external systems [56]. It helps to test that various devices from different vendors can communicate and understand each other.

- *Compliance Testing.* Compliance testing ensures that the SUT complies with laws, standards, or regulations.
- *Integration Testing.* It focuses on checking the interface between different components of the SUT [10].
- *Compatibility Testing.* Compatibility testing measures the degree of satisfaction to which a SUT can function satisfactorily alongside other products [10].
- *Usability Testing* checks the user behavior in using the system. Testing the UI and its navigation helps elucidate if a better design can make navigation simple and desirable, as well as prevent users from doing something they should not need to do [70].
- *Reliability Testing* Reliability testing evaluates the ability of a SUT to perform its required functions, including evaluating the frequency with which failures occur [10].
- *Maintainability Testing.* Maintainability testing is conducted to evaluate the effectiveness and efficiency with which a SUT may be modified [10].
- *Portability Testing.* Portability testing assesses how easily a SUT can transition from one environment to another and the extent of modifications required to make it functional in diverse environments.
- *Connectivity Driven Testing.* Connectivity testing is conducted to check how IoT device connects to other devices, and endpoints such as gateways [175].
- *Device Life Expectancy (DLE) Testing.* DLE testing is conducted to evaluate the time interval from when a device is sold to when it is discarded [102].
- *Distributivity Testing.* Distributivity testing is conducted to check how components of SUT can be spread across various devices in the network [177].
- *Dynamicity Testing.* Dynamicity testing checks how components and connectors can be created, connected, or removed during system execution [27].
- *Resilience.* The ability of an IoT system to recover quickly from failures, errors, or disruptions, and to continue operating effectively.

- *User Attitude.* Testing to ensure that an IoT system is user-friendly, intuitive, and meets user expectations, including usability, accessibility, and overall user experience.
- *Robustness.* The ability of an IoT system to withstand and resist failures, errors, or disruptions, and to continue operating effectively in adverse conditions.
- *Safety.* Testing to ensure that an IoT system does not pose any risks or hazards to users, and that it operates in a safe and secure manner.
- *Scalability.* The ability of an IoT system to handle increased load, traffic, data, or added devices without compromising performance.
- *Flexibility.* The ability of an IoT system to adapt to changing requirements, protocols, or technologies, and to integrate with other systems or devices.
- *Configuration.* Testing to ensure that an IoT system can be properly configured, customized, and updated and that its settings and parameters can be easily managed.
- *Sensor Accuracy and Reliability.* This involves verifying that sensors in the IoT devices accurately and consistently measure and report data. For example, temperature sensors should report correct temperature readings under various conditions.
- *Power Consumption and Battery Life.* Ensuring that the IoT devices have efficient power consumption and that the battery life meets the expected requirements.
- *Firmware Updates and Rollbacks* Verify that over-the-air (OTA) firmware updates apply successfully and devices can revert if an update fails.
- *Environmental Factors Checking.* Testing the device's performance under various environmental conditions, such as extreme temperatures, humidity, and other factors that the device might encounter in real-world use.
- *Device Provisioning and Configuration.* Ensuring that new devices can be correctly set up and configured to join the network. This includes testing the process of adding a device, configuring it, and ensuring it starts functioning as expected.

- *Communication Protocols.* Testing the implementation and performance of communication protocols specific to IoT, ensuring that protocols like MQTT, CoAP, and others are correctly implemented, efficient, and secure.
- *Geolocation Accuracy.* Verifying the accuracy and reliability of location-based services provided by the device. This includes testing GPS and other location technologies to ensure they report correct locations.
- *Event Handling and Logging.* Ensuring that the device correctly handles events (e.g., sensor triggers, and network changes) and logs relevant information.
- *Data Synchronization.* Verifying that data collected by the device is correctly synchronized with the cloud or other central systems. This includes ensuring data integrity and consistency across all platforms.

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

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

Takeaway

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

- **What Are Testing Tools and Artifacts?** We organized testing tools and artifacts for testing IoT systems, as shown in Figure 6.9.

  The identified artifacts include test cases (TCs), test data, test scenarios, test scripts, defects reports, test plans, test strategies, and traceability matrix. We define those artifacts based on ISO [82].

  – *Test Scenario.* A test scenario is defined as any functionality that can be tested (e.g., check the login functionality). One test scenario can have multiple TCs.

  – *TCs.* A test case is any single atomic test consisting of test preconditions, test inputs, expected test output, and expected test postconditions [56].

  – *Test Data.* Test data are data used to satisfy the input requirements for executing one or more TCs.

  – *Test Script.* A test script is a procedure specification document specifying one or more test procedures.

  – *Defect Report.* A defect report is a report indicating whether a specific test case has passed or failed.

  – *Test Plan.* The test plan consists of a detailed description of test objectives to be achieved and the means and schedule for achieving them, organized to coordinate testing activities for some test item or set of test items.

  – *Test Strategy.* Test strategy is a document that outlines how testing will be conducted.

  – *Test Log.* A record of the testing activities that have been performed on a software system, including the test cases that were run, the results of those tests, and any defects

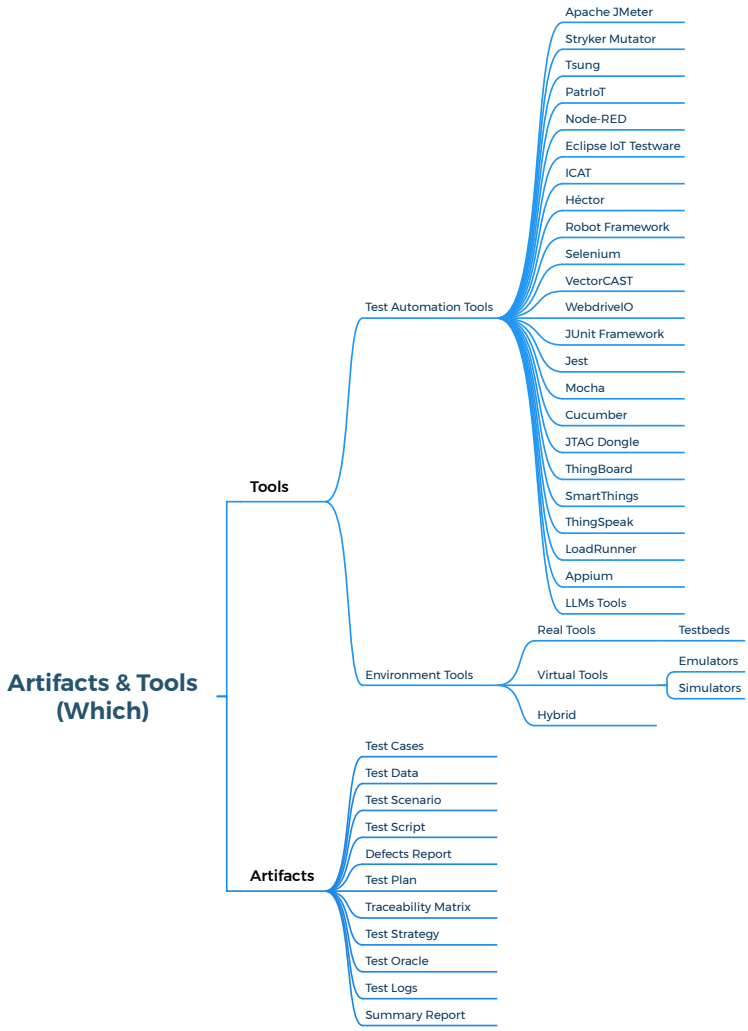


Figure 6.9: Which Tools To Use and Artifacts Produced

or issues that were identified.

– *Summary Report.* It summarizes the test execution, including the number of test cases executed and key metrics such as test coverage, pass/fail rates, and defect trends.
– *Traceability Matrix.* A tool to trace the requirements of a system to the corresponding test cases that have been created to validate those requirements [76].
– *Test Oracles.* A source for determining expected results for test cases and comparing them to the actual results of the SUT.

Testing tools are software applications or frameworks designed to support and automate various aspects of the testing process, including test case creation, execution, defect tracking, performance monitoring, and result analysis [56]. These tools are categorized into two groups: one for running tests (e.g., Selenium, Appium, and many LLM-based testing tools, such as ChatGPT, CodeX, CodeT5, GPT-4, and GPT-3 [193]), and the other for managing test environments (testbeds or simulators/emulators [42, 58]). We identified 23 tools in this taxonomy namely Apache JMeter, Stryker Mutator, Tsung, PatrIoT, Node-RED, Eclipse IoT Testware, ICAT, Héctor, Robot Framework, Selenium, VectorCAST, WebdriveIO, JUnit Framework, Jest, Mocha, Cucumber, JTAG Dongle, ThingBoard, SmartThings, ThingSpeak, LoadRunner, Appium, and LLMs Tools.

- *Testbeds.* Testbeds that consist of hardware and software components where IoT systems can be tested.
- *Virtual Tools* Virtual tools consist of real software on virtualized infrastructure [15].
- *Emulators.* Emulators are software tools that mimic the hardware and software of a real device [56].
- *Simulators* Simulators are software tools that mimic the software and environment where IoT devices operate, often for testing and development purposes [56].
- *Hybrid Environments.* Hybrid Environments comprise a mix of physical hardware and virtualized components.

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

Takeaway

The taxonomy excludes test automation tools due to their dynamic and rapidly evolving nature, focusing instead on artifacts common to all software systems. It is also important to recognize that many practitioners are not yet fully aware of LLM-based testing tools, such as ChatGPT, CodeX, CodeT5, GPT-4, and GPT-3[193], which are gaining relevance in the field.

- **Who Is Responsible For Testing?** We summarized the roles of individuals or organizations responsible for testing. We categorized the testers into 2 main groups: organization-based and role-based. Within the organization-based category, testing can be the responsibility of the following:
  - *Developing Organization.* The organization responsible for development also conducts the testing.
  - *Client Organization.* The client is responsible for testing.
  - *Contractor.* Testing is outsourced to a third party.
  - *Operator.* The operator may conduct some tests such as security and performance testing.

  The role-based classification includes:
  - *Developer.* The developer of the system can perform some, especially for unit tests.
  - *QA Engineers* ensure the quality of the system through testing and process enhancement.
  - *Infrastructure Engineer.* Infrastructure engineers evaluate various aspects of the infrastructure, including performance, security, reliability, maintainability, and distributivity, to ensure that the system meets quality requirements.
  - *Security Engineer.* Security engineers are responsible for testing the system's security and addressing vulnerabilities.
  - *System Administrator.* System administrators can conduct various tests and take care

of resource usage, backup, and recovery.

- *End User.* End users can conduct their tests to determine whether the system aligns with their needs and requirements and whether it is acceptable for use.
- *Maintenance Engineer.* Maintenance engineers may test system updates, patches, and bug fixes for stability, compatibility, and functionality.

Takeaway

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

- **What Are Testing Stages?** We summarized the stages of testing for both phase-based and continuous-based development approaches. In continuous development, testing is conducted based on Agile or DevOps methodologies.
  - *Agile Testing.* In agile-based testing, features are tested as they are developed.
  - *DevOps Testing.* In DevOps, the new build is run through a series of tests that will thoroughly check if the new build is ready for production.

  In phased development, testing occurs at various phases such as analysis, design, and coding. In many cases, testing is done in the testing phase.

Takeaway

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

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

Takeaway

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

- **What Are Testing Approaches?** We identified testing levels, testing types, testing techniques, automation, and scripting levels, as shown in Figure 6.10. We explain each category in this section based on ISO-29119 standards [173].
  - *Testing Levels.* There are six testing levels: unit, integration, system, acceptance, regression, and patch testing [176]. *Unit testing* checks individual IoT system components separately. *Integration testing* gradually tests larger subsystems as they integrate

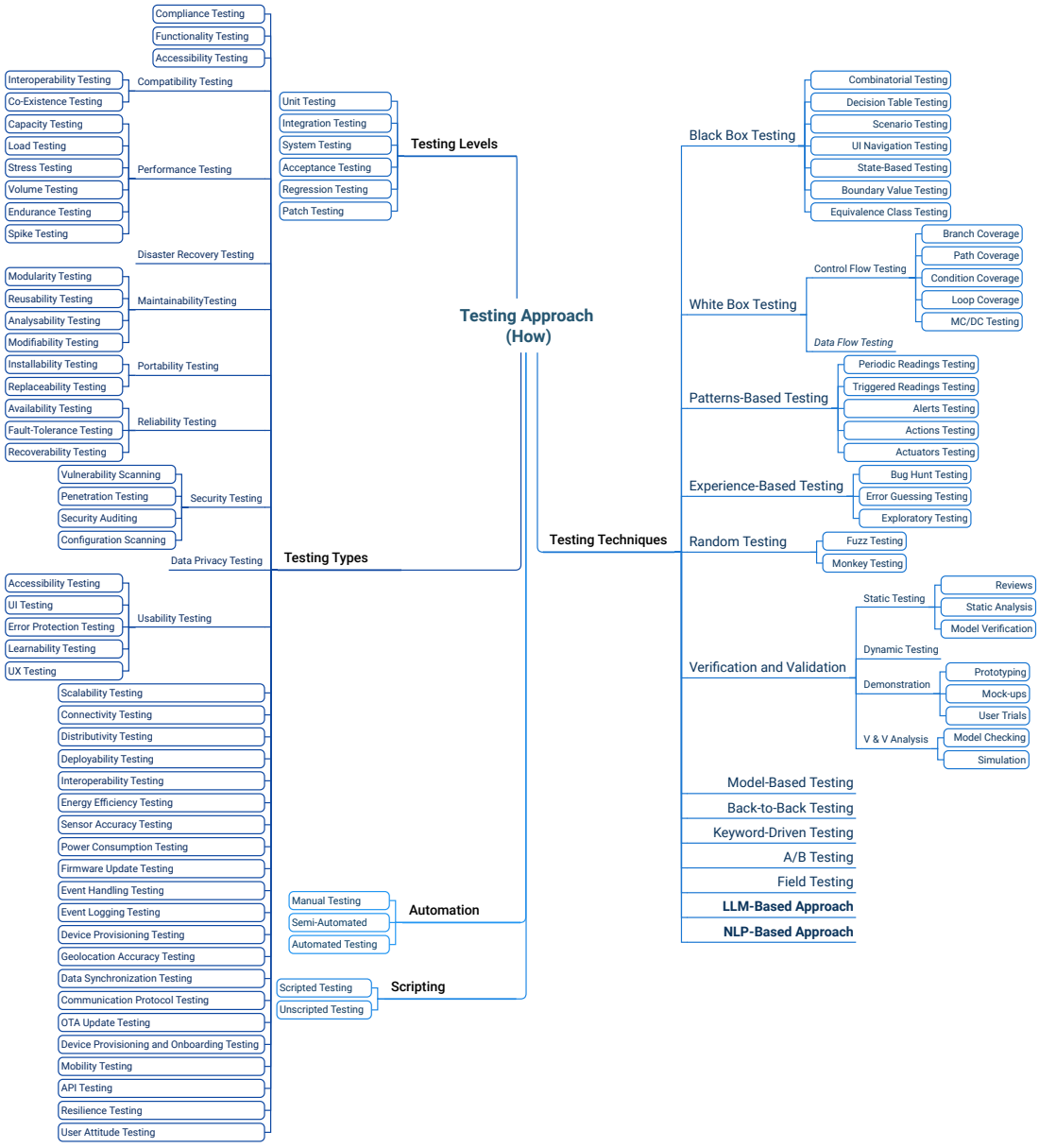


Figure 6.10: Testing Approaches

into the whole system [56]. *System testing* examines the entire system, not just individual components. *Acceptance testing* is software testing where the system is tested to determine whether it meets business requirements and is ready to be delivered to end users. *Regression testing* verifies code changes haven't introduced new defects [56].

– *Testing Types.* We identified several types of testing named after their test objectives.

The definitions for these testing types are provided in this section.

- *Testing Techniques. Technique-based testing* focuses on techniques used to assess the system functionality and quality to determine if it meets the specified requirements.
  - *Black-Box Testing.* Any testing with no knowledge of the internal structure or code of the component or system [56]. This testing is restricted to the externally visible behavior and characteristics of the item being tested. *Combinatorial testing* is a testing method that uses multiple combinations of input parameters to perform testing [82]. *Decision-table testing* is a specification-based test design technique based on exercising decision rules in a decision table [82]. *Scenario testing* is a testing technique that uses scenarios, i.e. speculative stories, to help the tester assess a test system. *User interface (UI) navigation testing* is a testing technique that focuses on verifying the navigation flow and UI elements of a SUT. *State-based testing* evaluates the behavior of the system based on different states. *Boundary-value testing* is a testing technique in which TCs are selected just inside, on, and just outside each boundary of an equivalence class of potential TCs [56]. *Equivalence class testing* divides the set of test inputs into different equivalence data classes.
  - *White-Box Testing.* White-box testing is based on an analysis of the internal structure and the code of a component or system under test [56]. *Control flow testing* such as *branch coverage, path coverage, condition coverage, loop coverage, and modified condition/decision coverage (MC/DC)*, focuses on the execution order of statements to develop the TCs. Branch coverage ensures that each branch is executed, thus ensuring that all reachable code is executed. Path coverage ensures that all possible paths are tested. Condition coverage focuses on the execution of different conditions independently of each other. Loop coverage focuses on the validity of the loop constructs. MC/DC ensures that each condition independently affects the decision outcome (i.e., ensuring that every condition within a decision determines

all possible outcomes of that decision). *Data flow testing* focuses on variables and their values across the different components of a system.

* *Patterns-based Testing.* Testing patterns are recurring, proven approaches or strategies used to design and execute effective TCs. Those patterns include *periodic readings testing* that evaluates system behavior during regular intervals. *Triggered Readings Testing* focuses on system behavior triggered by specific events. *Alerts testing* that verifies the generation and handling of system alerts. *Actions testing* that tests the actions or responses initiated by the system. *Actuators testing* that ensures proper functioning of actuators.
* *Experience-based Testing.* Experience-based testing is a testing technique solely based on the experience of the tester. Those include *bug hunt testing*, a structured search for defects in the absence of formal TCs. *Error guessing testing*, using testers' experience to guess the potential error-prone areas of the system. *Exploratory testing*, experience-based testing in which the testers simultaneously design and execute tests based on the tester's existing relevant knowledge, prior exploration of the test item [82].
* *Random Testing.* Testing techniques that are based on generating random inputs and TCs. Those include *fuzz testing* and *monkey testing*. *Fuzz testing* is an approach in which random inputs, called fuzz, are used to cause the system to fail [82, 56]. *Monkey testing* is a method that applies random inputs to a system, aiming to crash it, without predefined TCs.
* *Verification and Validation (V&V). Verification* is the process of evaluating the product or system to ensure that it meets the specified requirements (i.e., building the product right) [56]. Verification can be performed in many ways (e.g., demonstration, inspection, review, and testing). *Validation* is the process of evaluating the product or system to ensure that it meets the user's needs and expectations (i.e.,

building the right product) [56]. Validation can be performed using many techniques (e.g., 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 TCs. *Demonstration methods* focus on showcasing system functionalities. It includes *prototyping* (i.e., building a simplified version of the system), *mock-up* (i.e., a non-functional representation of the user interface), and *user trials* (i.e., involving end-users in testing). *V&V Analysis* is an analysis technique for formal verification of the system models (i.e., *model checking*) or creating simulations to assess the system behavior (i.e., *simulation*).

* *Model Based Testing (MBT).* MBT uses models to generate TCs systematically and automatically [82].
* *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 [82].
* *Keyword-driven Testing.* Test Cases (TCs) are written using keywords to represent test actions and data.
* *A/B Testing.* A/B testing is a statistical method to compare two systems and find which performs better.
* *LLM-Based Techniques.* Large Language Models (LLMs) represent a subset of machine learning models that utilize deep learning techniques and substantial datasets to generate human-like text. These models excel in various tasks, including text generation, question answering, and adapting to diverse scenarios. Recent research highlights an increasing trend in the integration of LLMs into the testing of various systems, demonstrating their versatility and effectiveness in automating and

enhancing testing processes [193].

  * *NLP-Based Techniques.* Natural Language Processing (NLP) is a domain within artificial intelligence that focuses on analyzing and interpreting human language. It uses its own rules and statistical methods for structured tasks such as information extraction and translation. By leveraging explicit linguistic rules and knowledge, NLP systems are designed to parse and understand grammar, syntax, and semantics. These systems utilize predefined rules to analyze sentence structure, identify key terms, and interpret the finer details of language. This enables NLP to discern the meaning and intent behind words expressed by humans. The application of NLP techniques in automating testing processes is notable, especially in converting requirements specified in natural language into executable tests [191, 57].
  * *Functionality Testing.* Any testing intended to verify functionality by causing the implementation of a system function to fail to identify defects [56].

- *Automation Levels.* TCs can either be run manually by a human (*i.e., manual testing*), or they can be executed by a test automation tool (*i.e., automated testing*). *Semi-automated testing* mixes automated testing and manual testing [82].
- *Scripting Levels.* Scripted testing follows a documented test script [82], adhering to pre-defined TCs and steps. In contrast, unscripted testing involves no formal preparation, documentation, or test scripts.
- *Mobility Testing.* Evaluate device performance and connectivity while in motion, simulating real-world use cases where devices are mobile (e.g., in vehicles, on drones).
- *Field Testing.* Deploy devices in actual use-case scenarios outside the lab to gather performance data in real-world conditions, assessing connectivity, power consumption, and sensor accuracy.
- *Over-the-Air (OTA) Update Testing.* This type of testing focuses on evaluating the process of remotely updating the device firmware, ensuring successful updates, and

handling failures.

– *Real-world Environment Simulation.* This is a strategy used to create test environments that mimic real-world conditions to evaluate how IoT devices perform under various environmental factors.

The taxonomy lacks some details, especially for dynamic testing and model-based testing. Although some details could be provided in other published materials such as ISTQB [174] or in ISO 29119 standards [173], we included only aspects that we identified from SLR or that were suggested by practitioners. Practitioners may consult those materials for more details if required.

Takeaway

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

- **What Are OUT And Metrics?** We identified the target item (specific IoT system layer or end-to-end) and the metrics, as shown in Figure 6.11. Items under test include end-to-end (E2E) scenarios or specific layers. E2E testing focuses on testing the entire system with all components integrated. E2E testing can be performed on the real system. Testers can also use models to represent the desired behavior of a system under test (SUT). Items under test can be specific layers such as device, network, cloud, or application layer. Metrics fall into two categories: domain-specific and general metrics, with the latter categorized into three groups: code-level, functional-based, and non-functional-based metrics as indicated in Figure 6.11. For functionally-based metrics, we classify them into two primary categories: Defect/Bug-based and Specification-based metrics.

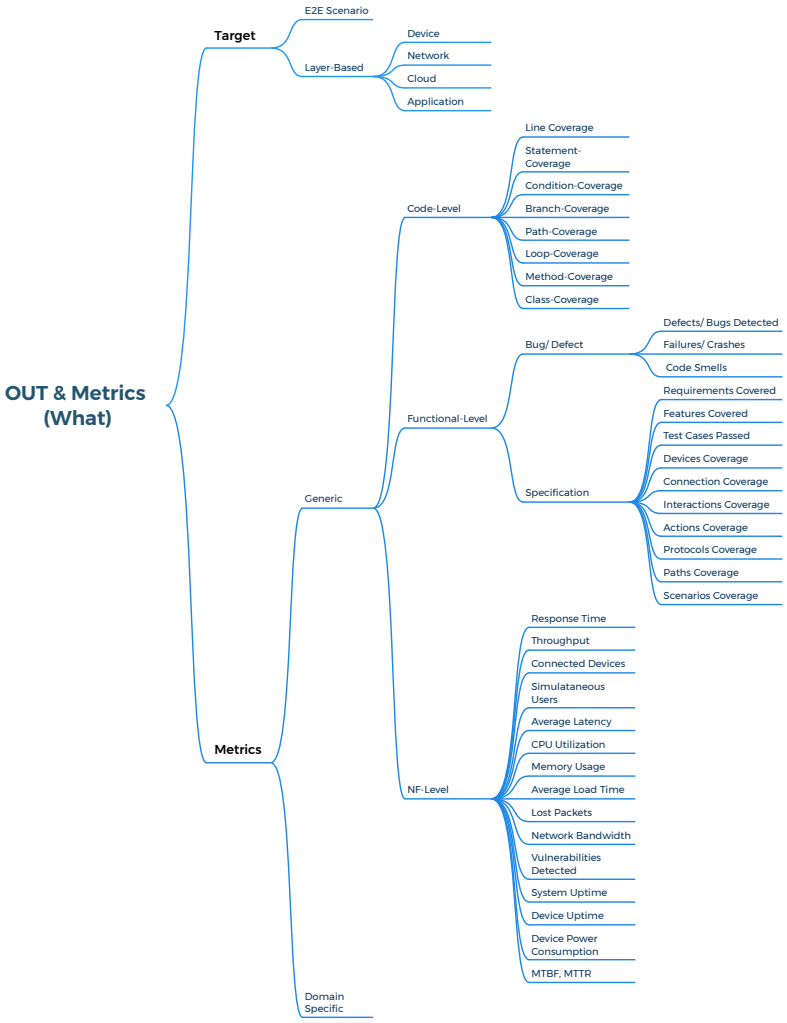


Figure 6.11: What To Test

Under Defect/Bug-based metrics, we identified three metrics: the number of bugs or defects detected, the number of failures or crashes, and the number of code smells. For Specification-based metrics, we identified the following metrics known from traditional software systems: the number of requirements covered, the number of features covered, and the number of test cases that passed or failed. However we have added IoT specific metrics such as: action/operation/command coverage, connection coverage, device coverage, interaction coverage, protocol coverage, and path coverage.

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

Takeaway

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

6.4.2 RQ7: How do practitioners evaluate the taxonomy?

We used the results from surveys to assess how the practitioners evaluate the proposed taxonomy.

- **Survey Round 1** Initially, we used the feedback provided by 16 experts to improve the initial taxonomy (i.e., round 1 of the survey). The participants evaluated the taxonomy, and we summarized their evaluation results in Figure 6.12. To ensure participant anonymity, we assigned unique identifiers (P1 to P16) to represent each of the 16 participants in this chapter. We consolidated "strongly agree" and "agree" into "agree", then aggregated their corresponding values. For instance, combining 43.75% strongly agreeing with 12.50% agreeing results in a total of 56.25% agreement.

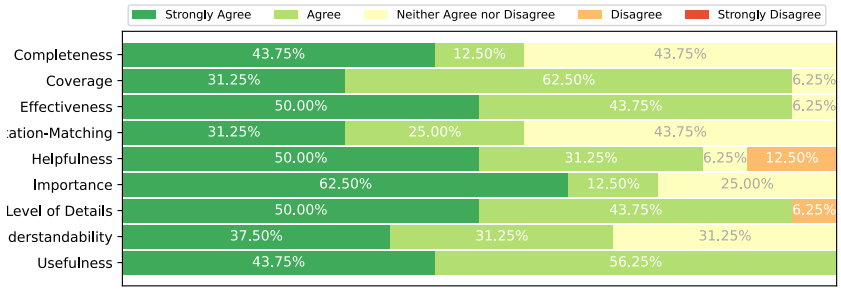


Figure 6.12: Evaluation Results - Round 1

We asked the practitioners to assess the completeness, understandability, usefulness,

helpfulness, expectation matching, importance, and effectiveness of the taxonomy on a scale of 1 (strongly disagree) to 5 (strongly agree) as shown in Figure 6.12.

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

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

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

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

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

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

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

> This taxonomy can help testers develop a comprehensive test plan that covers all the different components and elements of an IoT system, including devices, connectivity, data management, analytics, applications, security, and cloud computing. It can also help them design test cases that take into account the

> various challenges associated with testing IoT systems, such as data volume, device diversity, and complex interactions between components. (P7)

We asked the practitioners to assess the importance and the degree to which the taxonomy matched the practitioners' expectations using a scale from 1 (strongly disagree) to 5 (strongly agree). *Importance.* 75.0% of the participants agreed on the importance of the taxonomy, while 25.0% of the participants did not agree or disagree that the taxonomy is important to the practitioners.

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

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

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

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

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

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

Takeaway For Survey Round 1

The first survey round with 16 industry practitioners provided valuable insights to refine the taxonomy. Most participants found it useful (100%), helpful (81.25%), and effective (93.75%) in supporting IoT system testing. Key suggestions included adding security testing, business scenario (E2E) goals, and more granular guidance for testers. These inputs helped improve the taxonomy for broader adoption and practical use.

- **Survey Round 2** In the second round of evaluation, we used a 5-point scale ranging from 1 (strongly disagree) to 5 (strongly agree). Figure 6.13a shows the evaluation results. We ensured participant anonymity by assigning unique identifiers (P1 to P204) to represent each of the 204 participants.

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

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

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

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

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

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

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

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

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

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

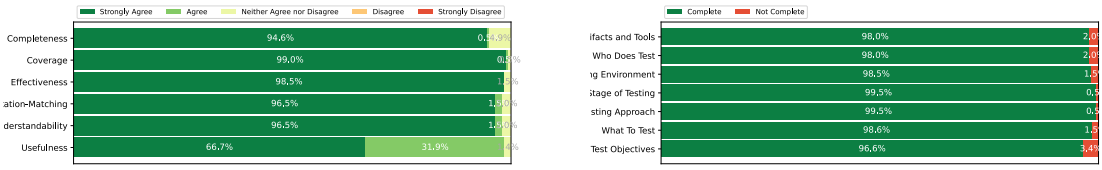
> This taxonomy is very useful, and complete, and will help the testers to understand all dimensions of testing. (P45)

We further asked the practitioners to evaluate the completeness of each dimension. We used “Yes” or “No” for each dimension. In the case of “No”, we asked the practitioners

to mention what they believed is missing. We used their feedback to improve the taxonomy. Figure 6.13b shows the evaluation results. At least more than 96% (196) of the practitioners believed that each of the categories in the taxonomy is complete.

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

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



(a) Overall Taxonomy Evaluation

(b) Completeness of each Aspect

Figure 6.13: Evaluation Results - Round 2

Takeaway for Round 2 of the Survey

The second round of evaluation confirmed ahowed strong agreement among 204 industry professionals on completeness (95.1%), coverage (99.5%), effectiveness (98.5%), expectation fulfillment (98.0%), understandability (98.0%), and usefulness (98.6%) of the taxonomy. Participants highlighted its structured approach to IoT testing, covering critical aspects such as connectivity, interoperability, security, and management. Additionally, practitioners provided valuable contributions to enhance the taxonomy, suggesting the inclusion of patch testing, Go-Live testing, backup and recovery testing, cloud environments, and additional testing phases. Their feedback further refined the taxonomy to ensure broader applicability and completeness for IoT system testing.

6.4.3 RQ8: How does the taxonomy improve test coverage?

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

- **Selection of Participants.** The participants involved in the empirical evaluation were not part of the previous surveys conducted. For each system, 12 master's and PhD students were selected as participants. These students were carefully chosen based on their affiliation with two specific labs that had access to the IoT systems used in the evaluation. The first group of students was from the Ptidej Lab at Concordia University, Canada, where the WIMP system is developed and maintained. This affiliation made them ideal candidates for the experiment. While the lab includes many students, only those with prior experience in developing or testing IoT systems were selected. Their experience was verified through an analysis of their LinkedIn profiles. The second group of students was from the University of Rwanda's Center of Excellence for IoT, where one of the authors

had access to another IoT system, SMART-CYPS. Admission to this program requires students to have at least two years of experience in IoT systems. In summary, these students were chosen because (1) they had relevant experience in developing or testing IoT systems, and (2) they had direct access to the respective systems. The primary focus of the study was to evaluate the breadth of test coverage in terms of the number of tests created with and without the taxonomy, rather than the quality of individual tests. The tests created by the students were not executable. The decision not to execute these tests on the systems was due to the need to convert them into executable scripts, which was beyond the scope of the current study. Our evaluation is solely based on the number of tests and scenarios covered by each participant.

- **Experimentation with WIMP** *WIMP Description.* WIMP[10] or "Where Is My Professor", is an IoT-based system that enables students to track the availability of their professors in real-time. The main objective of this system is to collect data from various IoT devices, such as sensors, cameras, and beacons, to provide an automated response on the professor's availability by analysing the collected data. This system has smart sensors, such as the WEMO smart plug[11], as well as devices like the Fitbit Watch[12], which serve the purpose of location tracking and play a pivotal role in the decision-making process. The core modules of WIMP are the following:
  1. Internal user management module includes user accounts, profiles, permissions, etc.
  2. Data collection module for real-time tracking of professor's availability and location by continuously collecting data from various IoT devices.
  3. Availability module to provide an automated response about professor's availability based on analysis of the data collected from various IoT devices.

  More details about WIMP can be found online [13].

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

Participants. We conducted an empirical study with 12 participants, randomly dividing them into two groups: Group 1 (G1) and Group 2 (G2). G2 received a copy of the taxonomy, while G1 did not. We collected the details of participants as recommended in [43]. G1 comprises three male and three female participants, all holding at least a bachelor's degree and possessing a minimum of three years of experience in software engineering. Additionally, two participants in G1 have prior working experience in IoT systems. G2 comprises two female and four male participants, all holding at least a bachelor's degree and having more than three years of working experience in software engineering. None of the participants in G2 have prior working experience in IoT systems. We asked each participant to study and prepare test scenarios (TSs) and Test Cases (TCs) for the three modules mentioned above, with each participant given 2 hours maximum.

Results. Table 6.4 shows the number of TSs and TCs identified by each participant. We observed that the group that used the taxonomy identified more TSs and TCs compared to the group that did not. G1 primarily focused on functional aspects, while G2 identified more non-functional aspects such as connectivity, security, performance, and compatibility on top of functional aspects. Table 6.3 shows an example of a test created by one participant for functional testing based on modified template provided by [191].

Table 6.3: Example of Test Created by Participant

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

Our empirical results show that the taxonomy can guide testers with no prior experience in testing IoT systems by helping them understand the various dimensions of testing in

IoT systems, thus creating more tests than the testers without taxonomy.

Table 6.4: Empirical Evaluation Results for WIMP

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

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

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

  Participants. We invited 12 participants and tasked them to experiment on SMART-CYPS. 8 (66.7%) were master students in the IoT program, and 4 (33.3%) were final-year students majoring in software engineering with a focus on IoT. 7 (58.3%) were female, while 5 (41.7%) were male. 6 (50%) had between 3 and 5 years of experience in SE. 2 (16.7%) had more than 5 years of experience in SE, while 4 (33.3%) had less than 3 years of experience in SE. 8 (66.7%) had between 1 and 3 years of experience in IoT-related projects. 3 (25%) had less than a year, while 1 (8.3%) had between 3 and 5 years

of experience in IoT projects. We divided 12 participants randomly into 2 groups of 6 participants each. One group with taxonomy, and the other without. The group with taxonomy was given 30 minutes to consult the taxonomy before the experiment. We asked each member of each group to conduct the experiment within a maximum of 2 hours. Each participant documented test cases and test scenarios in an Excel file, which was submitted after 2 hours.

Results. We analyzed the results as shown in Table 6.5. G1 indicates the group without taxonomy, while G2 represents the group with taxonomy. The participants with taxonomy created more test cases and scenarios than the participants with no taxonomy. They also identified more aspects compared to the group that did not have the taxonomy.

Table 6.5: Empirical Evaluation Results for SMART-CYPS

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

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

Takeaway for RQ8

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

6.5 Discussion

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

6.5.1 Alignment of Testing Objectives and Testing Types

The link between testing objectives and testing types is a critical aspect of understanding their alignment. Figure 6.8 defines the goals that testing seeks to achieve, while testing types, shown in Figure 6.10, represent the diverse methods used to accomplish these goals. The overlap between these elements is both expected and intentional. Figure 6.8 highlights the most commonly reported objectives identified through literature and surveys. Similarly, the testing types encompass a broader range, including not only tests explicitly targeting these objectives but also other forms of testing reported in the literature and surveys, regardless of whether they directly align with specific objectives.

6.5.2 Practitioners Feedback

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

6.5.3 Experimental Insights

The experiment provided valuable insights into testing IoT systems on two different platforms. Testers predominantly focused on functionality, device connectivity, and data transmission, possibly because they undervalued other aspects or were accustomed to prioritizing functionality, device connectivity, and data transmission in their testing practices. Additionally, testers struggled to create executable tests due to varying execution targets that required knowledge of different technologies and programming languages. Instead,

they concentrated on defining test steps, specifying inputs, expected outputs, and execution targets. This approach is reasonable given their limited knowledge of execution targets (i.e., test runners). Interestingly, some testers without access to the taxonomy performed well, likely due to their prior experience and the time dedicated to the experiment. However, the taxonomy proved particularly beneficial for testers with no prior experience, offering them a structured framework and a broader range of testing options. This suggests that using the taxonomy can significantly enhance testing outcomes, especially for novice testers.

6.5.4 Navigating the Taxonomy for Effective Testing

The proposed taxonomy provides a structured, seven-step navigation map (Figure 6.6) to guide testers in planning and executing comprehensive testing. The process begins with defining testing objectives (reason for testing) and identifying the object under test, which may range from individual layers (e.g., devices, networks) to end-to-end system evaluations. Testers then select appropriate testing approaches, including types (e.g., security, performance), levels (e.g., unit, integration), or techniques (e.g., random, model-based, black-box/white-box). Subsequent steps involve setting up the testing environment, defining testing stages (e.g., phase-based testing or continuous testing), specifying testing artifacts (e.g., bug reports, test reports), and choosing tools for test instrumentation. The final step assigns testers with the necessary expertise to ensure success. Figure 6.7 exemplifies how testers can tailor their testing plan, highlighting mandatory aspects such as objectives, artifacts, testers, stages, approaches, objects under test, and environments. Optional considerations like scripting levels, automation, and tools depend on the testing automation-level. For example, if testing is to be done manually, these may be unnecessary. We believe that this taxonomy can serve as a valuable resource for creating a complete IoT and traditional systems testing framework. To ensure accessibility, we will make this guide publicly available, helping practitioners not only understand the taxonomy but also effectively apply it.

6.5.5 Continuous Relevance and Adaptability

To ensure the taxonomy remains comprehensive and up-to-date, we began with a systematic review of studies published up to 2022, complemented by insights gathered through industry collaboration via surveys to validate and enrich the taxonomy. Recognizing the importance of incorporating the latest advancements, we extended our review at the time of submission by conducting an updated search for studies published between 2022 and November 2024, applying the same inclusion and exclusion criteria. This process identified a few new studies (i.e., [179, 152, 84]) not included in our initial review, from which additional testing aspects such as user attitude testing or resilience testing were extracted to refine the taxonomy where necessary. Furthermore, we will continue to review the literature and gather feedback from industry practitioners to regularly update the taxonomy and incorporate emerging aspects of IoT testing.

6.5.6 Fragmented IoT System Testing Aspects

Existing studies, including [176, 117] and [93, 55], provided a taxonomy focusing solely on testing types. Notably, study [115] focused on bug taxonomy within IoT systems. To the best of our knowledge, no IoT system testing taxonomy existed before our work. Our proposed taxonomy is the first of its kind and is also adaptable for traditional software testing. This contribution not only fills a significant gap in the literature but also lays a foundation for future research in IoT system testing. Furthermore, our taxonomy's adaptability for traditional software testing extends its utility beyond the IoT domain, making it a valuable resource for researchers and practitioners across different domains of software engineering.

6.5.7 Implication for Practitioners

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

Guidance for testing IoT systems

We observed that many studies often use some terms interchangeably to denote distinct concepts, potentially leading to confusion among testers. We proposed some definitions to clarify those concepts for better understanding. Yet, the taxonomy may not be exhaustive enough to guide the testers; it offers valuable guidance for novice practitioners embarking on their IoT system testing endeavors. By providing clear definitions and categorizations, this taxonomy serves as a foundational tool for establishing a common language within the testing community, fostering clearer communication and reducing ambiguity in IoT testing practices. Additionally, it lays the groundwork for further research and refinement, enabling ongoing improvements in testing methodologies and techniques tailored to the unique challenges of IoT environments. We made this taxonomy available online [14] for practitioners and researchers who may want to refer to it.

Revolutionizing Testing with LLM

Recent advancements in large language models (LLMs) have opened new avenues for automating various tasks in testing traditional systems. While it is unclear whether any studies have specifically focused on automating testing aspects within IoT systems, we believe that IoT system practitioners can equally benefit from using LLMs to automate many testing tasks. Recent studies [192, 21, 203, 71, 165, 193] studied LLMs for their potential to enhance test automation.

6.5.8 Implication for Researchers

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

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

6.6 Threats to Validity

There are threats to the validity of this chapter, the taxonomy, and its evaluation.

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

Internal Validity. Several factors could affect the internal validity of this chapter. Bias in the selection and grouping of testers is a potential threat, as differences in commitment,

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

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

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

6.7 Conclusion

This chapter introduced a novel IoT-specific testing taxonomy, which was developed from an analysis of 83 primary studies and refined through feedback from 220 industry practitioners. The proposed taxonomy categorizes seven crucial aspects of IoT systems testing: objectives, tools, testers, stages, environments, objects under test, and approaches. The empirical evaluation of this taxonomy involved 24 testers across two IoT systems, demonstrating that those equipped with the taxonomy could more effectively identify diverse test cases and scenarios. To facilitate its practical application, we provided guidelines on how to use the taxonomy and made it available online for unrestricted access.

The findings of this chapter have been published in the Journal of Systems and Software [120]. Building on this, the next chapter introduces a framework for testable technical aspects of IoT systems, focusing on software engineering-driven testability while excluding dependencies on external partners.

Chapter 7

Framework for Testing Technical SE Aspects of IoT Systems

7.1 Introduction

This chapter builds on our previously established taxonomy for testing IoT systems. While the taxonomy provides a comprehensive classification of testing aspects, we identified that many of these aspects cannot be effectively tested by software engineers alone. Testing IoT systems involves multiple dimensions, including connectivity, security, infrastructure, and software engineering, making it challenging to delineate the responsibilities of software engineers from those of other stakeholders, such as security, connectivity, and infrastructure teams. Despite the availability of a taxonomy to guide testers, a dedicated framework for testing the technical aspects of software engineering is necessary. Such a framework ensures that software engineers focus on testing the software-related components of IoT systems, while other aspects are either tested independently by relevant stakeholders or in collaboration with them. Despite the critical role of IoT systems in various domains, such frameworks remain underexplored [11, 25, 163]. Existing testing frameworks in software

engineering primarily focus on traditional software systems and may not adequately address the complexities of IoT systems [128]. To bridge this gap, this chapter analyzes the taxonomy for testing IoT systems and formulates a structured framework specifically for testing the technical software engineering aspects of IoT systems. The proposed framework identifies and extracts all technical aspects that software engineers can systematically test. This includes aspects such as functionality, performance, and interoperability. Furthermore, it defines the granularity of testing, maps identified aspects to specific testing scopes (e.g., single-layer or end-to-end testing), determines relevant test input artifacts (e.g., source code, behavioral models, specifications), and establishes test orchestration strategies (e.g., logging, observation, interception) and test execution strategies (e.g., black-box, white-box, or gray-box testing). We developed this framework based on the previous chapter, leveraging insights from industry expertise and our experience in IoT and software engineering. To validate the framework, we conducted a survey with software engineering professionals, assessing its practicality and identifying areas for refinement. Additionally, we performed experimental studies that included event log handling testing and data integrity testing to validate the framework's applicability in real-world scenarios.

This chapter answers the following research questions (RQs):

- **RQ9:** What are testable technical SE aspects of IoT systems?
- **RQ10:** How complete are the testable technical SE aspects of IoT systems?
- **RQ11:** How practical is the framework for real IoT systems?

The main contributions of this chapter are as follows:

- Propose **TISSEA**, a framework for testing the technical software engineering aspects of IoT systems. This framework identifies technical aspects, defines testing granularity for each layer of IoT systems, and specifies orchestration strategies, input artifacts, and execution methodologies.
- Evaluate TISSEA through a survey of 22 professionals, analyzing their feedback to refine

and enhance the framework.

- Conduct experimental case studies to validate the feasibility of testing the identified aspects using real-world IoT systems.

The evaluation results demonstrate the completeness of the proposed framework for testing the technical software engineering aspects of IoT systems. The empirical study on selected case studies demonstrated its practicability and showed promising results for improving IoT system quality. The rest of this chapter is structured as follows: Section 7.2 presents our research method. Section 7.3 focuses on framework derivation. Section 7.4 describes the framework. Section 7.5 presents the evaluation of the framework and the results. Section 7.7 discusses the findings. Section 7.8 highlights potential threats to validity. Section 7.9 concludes this chapter.

7.2 Research Method

Figure 7.1 describes the research method we used in this chapter. We followed 6 steps: 1) study the taxonomy for testing IoT system, 2) identification of technical software engineering (SE) aspects, 3) determination of test target for each technical SE aspect, 4) identify test methodology and instrumentation strategies for each aspect, 5) construct the framework, and 6) validate the framework through survey.

7.2.1 Study the taxonomy for testing IoT System

We studied the taxonomy for testing IoT systems [120] to identify all potential testable aspects of IoT systems, recognizing that the taxonomy encompasses numerous aspects relevant to various stakeholders.

7.2.2 Identify Technical SE Aspects and Test Granularities

We defined technical software engineering aspects to be the ones that can be conducted by software engineers alone without involving other stakeholders or end-users from technical implementation. We studied the taxonomy for testing IoT systems [120] to identify the

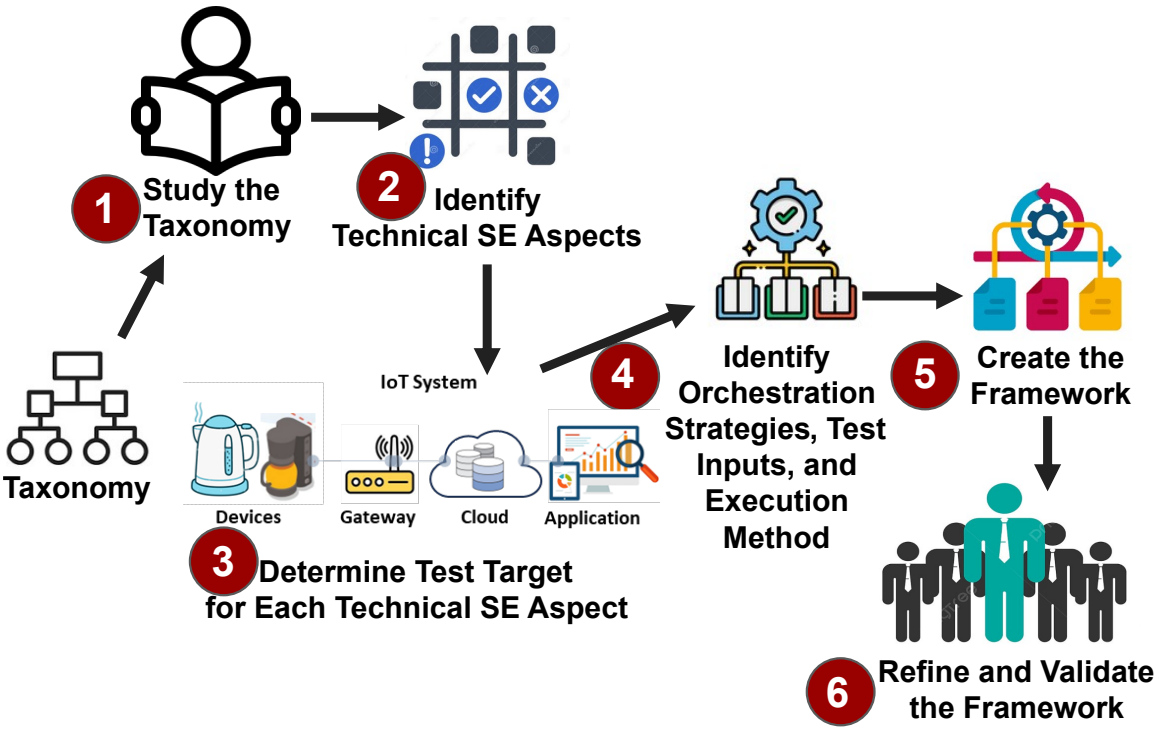


Figure 7.1: Research Method

technical SE aspects relevant to testing IoT systems. From all the potential testable aspects, we selected those that software engineers are capable of testing exclusively, based on their expertise and responsibilities. We also identified test granularities ranging from unit testing to system-level testing.

7.2.3 Determine Test Target for Each Technical SE Aspect

For each identified technical SE aspect in Section 7.2.2, we used the taxonomy in [120], to determine the appropriate targets (i.e., the objects or items under test, such as a specific device, application, or cloud) to focus the testing efforts.

7.2.4 Input Artifacts, Orchestration Strategies, and Execution Methods

Through our collaboration with industry partner, we proposed the most suitable test input artifacts, orchestration strategies, and execution methods for each technical SE aspect of

the IoT system identified in Section 7.2.2.

7.2.5 Build the Framework

We mapped the results of Section 7.2.2, Section 7.2.3, and Section 7.2.4 to construct the TISSEA framework, which guides software engineers in testing the technical SE aspects of IoT systems.

7.2.6 Validate the Framework

We validated the framework through the survey with software engineers experienced in IoT system development and testing to ensure its completeness. Additionally, we performed experiments on two case studies to assess the practicality and applicability of the proposed framework.

7.3 Framework Derivation

In this section, we discuss the details of each step involved in deriving TISSEA.

7.3.1 Study the Taxonomy

In the recent study [120], we suggested seven aspects to consider when testing IoT systems: (1) testing objectives, (2) testing tools and artifacts produced, (3) responsibilities of testers, (4) testing stages, (5) testing environment, (6) object or item under test, and (7) testing approaches. These aspects are generic, and not all of them fall within the responsibilities of software engineers. We define technical SE aspects as those elements of IoT testing that can be performed solely by software engineers, without requiring the involvement of other stakeholders such as hardware engineers, network engineers, or end users. This study focuses on identifying and analyzing such aspects.

7.3.2 Identify Technical SE Aspects

We studied each of these aspects to identify specific elements that could be tested in part or in full by software engineers. The goal was to identify testable aspects of an IoT system

based on the source code, assuming that the responsibility for testing these aspects belongs solely to software engineers, without the involvement of other stakeholders. Initially, we involved two researchers. Each researcher independently created a matrix that included each aspect. Each aspect in the matrix was marked with "Yes" if it was assessed as suitable for inclusion in TISSEA or "No" if it was not, based on the researcher's understanding. After both researchers completed this exercise, they compared their results and included the aspects marked "Yes" by both. Aspects marked "No" by both researchers were eliminated. In case of disagreement, where one researcher marked an aspect as "Yes" and the other marked it as "No", a third researcher was involved in the discussion until an agreement was reached, either by including or excluding such aspect. The third researcher, from our industry partner NTT, has over 10 years of industrial experience in testing IoT systems. We greatly benefited from his expertise in discussing each of these aspects. At the end of this process, all aspects marked as "Yes" were considered for inclusion in TISSEA.

7.3.3 Determine Test Target for Each Technical SE Aspect

The researchers analyzed the taxonomy in our recent study [120] to identify the possible test targets. The taxonomy proposed six layers of IoT systems. After reviewing these layers, the researchers observed that only four layers are common across any IoT system [26, 6, 159, 180]. These four layers include the device layer, network layer, platform layer (such as the cloud), and application layer (such as mobile or web applications). The researchers treated each of these layers as one distinct category of test target. They agreed to focus on these four layers because they form the core of most IoT system architectures. In addition to these layer-specific targets, and through collaboration with industry partners, the researchers also agreed to consider end to end (E2E) or system-level testing as another test target.

7.3.4 Identify Orchestration Strategies, Test Inputs, and Execution Method

To identify test orchestration strategies, test input artifacts, and execution methods, we created two matrices. The first matrix maps test orchestration strategies to the corresponding software engineering aspects and test granularities. The second matrix maps input artifacts and execution methods, such as a black box, a white box, or a gray box, to software engineering aspects and test granularities. Two researchers independently proposed values for each matrix cell, using the taxonomy and their own understanding. When they disagreed, they discussed the differences to reach a common decision. If they could not agree, they marked the item for review with our industry partner.

We collaborated with our industry partner NTT, a team of six professionals who test real-world IoT systems for clients. Their feedback helped us refine our initial proposals, and their suggestions shaped the final version of the matrices.

7.3.5 Create the Framework

The proposed framework provides guidance on seven components that software engineers should consider. In this study, we cover the following six components. First, we define the scope of testing, which refers to the test target, such as a specific device, gateway, cloud service, or end-user application, including mobile apps or traditional web apps. Second, we identify the technical aspects to test, such as functional testing, API testing, and others. Third, we specify the test granularity, including unit testing, integration testing, component testing, component integration testing, system testing, or end to end testing. Fourth, we consider the input artifacts available for testing, such as source code, specifications, and documentation. Fifth, we determine the test orchestration strategy, which may involve mocking, observations, interceptions, logging, or other techniques. Sixth, we choose the test execution method, such as black box, white box, or gray box, based on the available

input artifacts and knowledge of the system.

For these six components, we combine the results from earlier steps to create a unified view. The seventh component, which involves test generation, falls outside the scope of this study. Test generation depends on selections made in the other steps and varies across different approaches. Other studies explore this topic in the context of IoT systems and offer guidance based on specific technical aspects. We still include this step in the framework to ensure its completeness and to support software engineers in applying it effectively. A detailed description of this framework appears in Section 7.4.

7.4 Framework Description

This section presents TISSEA, a framework that guides software engineers in testing technical software engineering aspects of IoT systems. Figure 7.2 shows the high-level overview of the proposed framework.

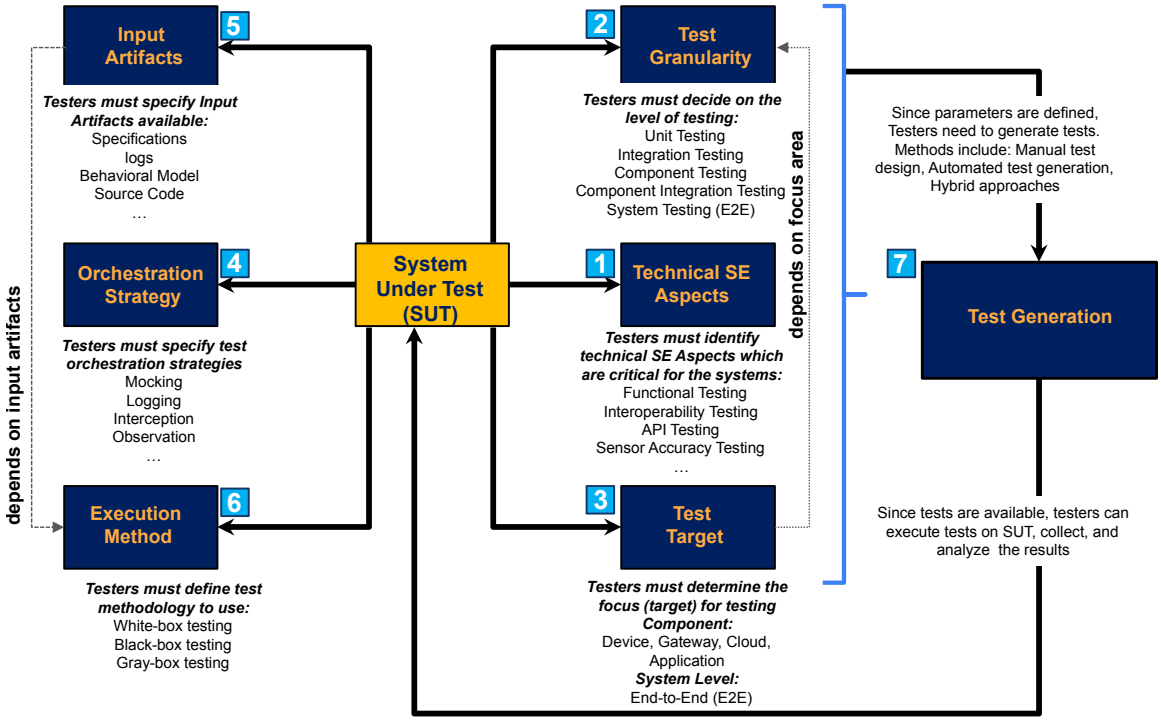


Figure 7.2: TISSEA: A Framework for Testing SE Aspects of IoT Systems

7.4.1 Technical SE Aspects and Test Granularity

We included 19 aspects that were marked as "Yes" in the previous step. Some aspects that could be tested by software engineers, such as usability, were excluded because they concern user experience (UX) rather than technical elements and involve end users. Although some testing aspects, such as connectivity and security typically fall under the responsibilities of other stakeholders, such as network and security engineers respectively, we acknowledged that software engineers can still test certain parts of these aspects. Therefore, we considered them for inclusion in TISSEA. The final list of selected aspects defines what we refer to as the technical SE aspects, which form the core components of TISSEA. These aspects clarify what can be systematically tested by software engineers in isolation. We now define these aspects in the context of IoT systems from SE perspective to improve clarity regarding the scope of testing responsibilities for software engineers. The ordering of those aspects begins with those fully within the testing responsibilities of software engineers, and progresses toward aspects where their role is more limited.

- Functional testing involves verifying that the software of the individual component or the entire IoT system performs as intended by validating their behavior against specified requirements.
- API testing involves verifying the behavior of Application Programming Interfaces (APIs) by sending requests and monitoring responses to ensure they function as expected.
- Data integrity testing ensures that data is accurately collected, transmitted, stored, and retrieved without unexpected modification or corruption.
- Data serialization testing verifies that complex data structures or objects are accurately converted into a linear sequence of bytes, such as JSON, XML, or binary formats that can be stored or transmitted, and that the serialized data can be deserialized to its original form.

- Data synchronization testing evaluates that multiple copies of data are consistent and up-to-date. This testing involves updating data in real-time or near-real-time across different devices.
- Sensor accuracy testing ensures that the data retrieved from sensors matches the expected real-world values within an acceptable range.
- Event handling testing ensures that triggered events (e.g., alarms, notifications) are processed correctly.
- Event logging testing validates that all significant events are logged for auditing and debugging.
- Interoperability testing ensures that devices from different manufacturers can seamlessly communicate, exchange data, and work together as intended to achieve the system functionality.
- Performance testing evaluates the performance of the IoT system under various loads and volumes of requests, such as the ability of algorithms to handle multiple requests/responses across diverse devices.
- Resilience testing involves evaluating the source code of IoT devices and applications to assess their ability to detect, resist, and recover from faults, errors, and unexpected events.
- Heterogeneity testing tests the system's ability to operate with various devices, protocols, and platforms.
- Connectivity testing involves analyzing and testing the source code of IoT devices and applications to validate data transmission/reception between IoT devices, the cloud, and the application.
- Compatibility testing involves evaluating the source code of IoT devices and applications to ensure correct interaction and interoperability with other devices, applications, and

communication protocols, enabling reliable operation across heterogeneous IoT ecosystems.

- Security testing for software engineers involves evaluating the source code of IoT devices and applications to identify vulnerabilities and weaknesses that could compromise confidentiality, integrity, and availability, such as buffer overflows, SQL injection, and insecure data storage.
- Scalability testing, from a software engineering perspective, involves analyzing the source code of IoT devices and applications to evaluate their ability to handle increased data volumes, loads, and number of connected devices.
- Energy efficiency testing involves evaluating the source code of IoT devices and applications to ensure optimal energy usage during operations, minimizing power consumption while maintaining performance.
- Power consumption testing verifies that the system minimizes power consumption during idle and active states.
- Device discovery testing evaluates the system's ability to discover and pair with nearby devices, as well as to be discovered by them.
- Latency Testing from a software engineering perspective involves analyzing and testing the source code of IoT devices and applications to measure, identify bottlenecks, and optimize the delay during data transmission.

After identifying the technical aspects, we identified three possible levels of granularity from the taxonomy that are relevant to software engineers during the development and testing of an IoT system. Two researchers suggested adding "component testing" and "component integration testing" as new granularity levels. Through discussions, all researchers agreed to include them in addition to the levels extracted from the taxonomy. We provide definitions for the identified test granularity levels below in the context of IoT systems.

- Unit testing involves verifying the functionality of individual classes or functions in the

source code deployed on a single IoT component (i.e, a node such as a specific device or cloud app). It is similar to unit testing in traditional software systems.

- Integration testing evaluates the interactions between multiple classes or functions in the source code of a single IoT component. It is similar to the integration testing in traditional software systems.
- Component testing assesses the overall functionality of a single IoT component, such as one device or application. It is similar to system testing in traditional software systems.
- Component integration testing verifies the seamless interaction between multiple IoT components, such as devices, gateways, and cloud services.
- System testing in IoT involves end-to-end validation of the entire IoT system, including devices, gateways, networks, cloud/services, and applications, to verify interaction and functionality.

7.4.2 Test Target

Test target refers to the item or object under test, such as a device, gateway, cloud, or application; for devices, each specific instance can be considered a distinct target. For example, testing a Fitbit smartwatch represents testing a specific device belonging to the IoT layer. Below, we categorized the following possible targets:

- Thing (Device) Layer comprises physical devices, such as sensors, actuators, and smart devices, that perceive and interact with the physical world.
- Gateway/Network Layer enables communication between devices and the cloud/service layer, providing data processing, protocol conversion, and network connectivity.
- **Service/Cloud Layer** provides infrastructure for storing, processing, and managing data, and supports services such as data analytics and device control.
- **Application Layer** delivers user-facing applications that use data from other layers to offer meaningful IoT services and functionalities.

Figure 7.3 illustrates how we mapped each layer to technical aspects and test granularity.

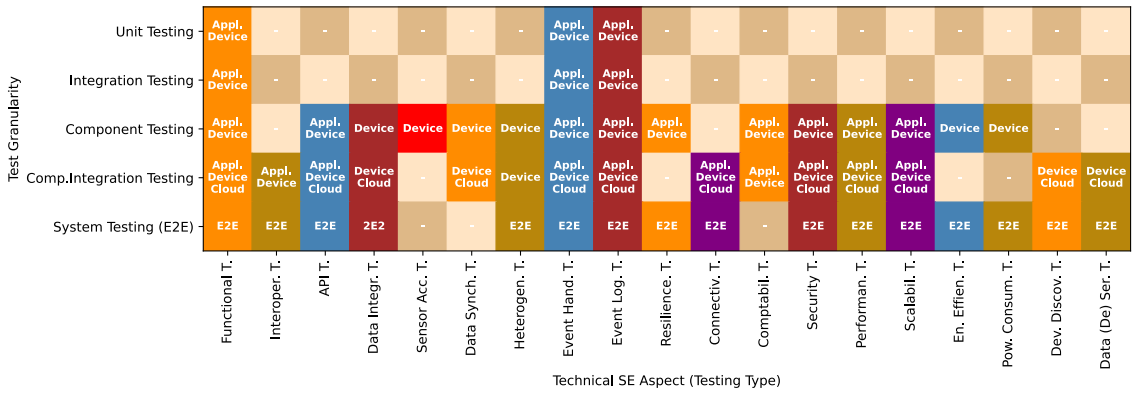


Figure 7.3: Technical SE Aspects and Test Granularity with their Test Target

Technical SE Aspects: Functional Testing, Interoperability Testing, API Testing, Data Integrity Testing, Sensor Accuracy Testing, Data Synchronization Testing, Heterogeneity Testing, Event Handling Testing, Event Logging Testing, Resilience Testing, Connectivity Testing, Compatibility Testing, Security Testing, Performance Testing, Scalability Testing, Energy Efficiency Testing, Power Consumption Testing, Device Discovery Testing, Data (De)Serialization Testing.**Test Granularity:** Unit Testing, Integration Testing, Component Testing, Component Integration Testing, E2E (End-to-End) Testing **IoT Layers**: Device, Gateway, Application, Cloud)

7.4.3 Orchestration Strategies

We mapped each technical SE aspect and its corresponding granularity to orchestration strategies. Figure 7.4 shows the resulting matrices, which offer a structured view of how these aspects and granularities align with orchestration strategies in IoT system testing.

7.4.4 Input Artifacts

As a result of the discussion between the researchers and industry partners, we proposed eight input artifacts of the SUT. These include Specifications, Source Code, Behavioral Models, End Points, Event Scenarios, Configuration Files, Point Cuts, and Reference Instruments. We mapped each technical SE aspect and its corresponding granularity to the input artifacts. Figure 7.5 shows the resulting matrices, which offer a structured view of

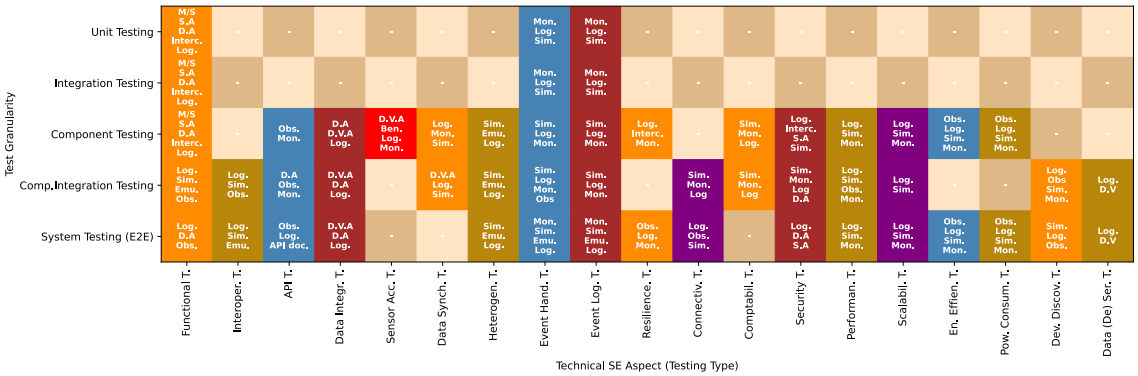


Figure 7.4: Orchestration Strategies

(**M/S:** Mocking/Stubbing, **S.A:** Static Analysis,**D.A:** Dynamic Analysis, **Interc.:** Interception (Fault Injection),**Log.:** Logging, **Sim.:** Simulation, **Emu.:** Emulation, **Obs.:** Observation, **D.V.A.:** Data Variation Analysis, **Mon.:** Monitoring, **D.V:** Data Validation)

how the aspects and granularities align with the input artifacts in IoT system testing.

7.4.5 Execution Method

As a result of discussions between the researchers and industry partners, we also proposed three types of test execution methods: black box, white box, and gray box. The idea is that the execution method depends on the type of input artifacts available for the SUT. For instance, if the source code is available, the test can be executed as a white box. If only specifications are available, the test must be executed as a black box. In cases where partial source code is available (i.e., when some components expose their internal logic while others do not) we consider gray box execution. We mapped each technical SE aspect and its corresponding granularity to the appropriate execution method. Figure 7.5 also illustrates this mapping, offering a structured view of how the aspects and granularities align with the selected execution methods in IoT system testing.

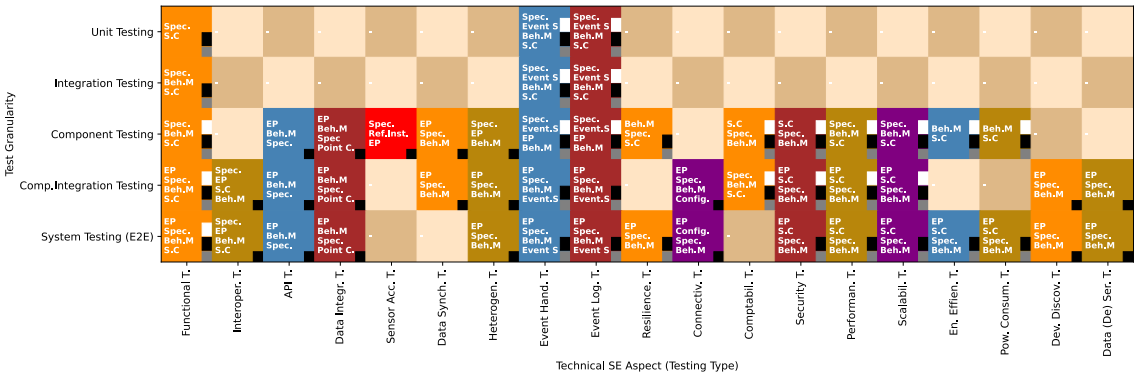


Figure 7.5: Input Artifacts and Execution Method

Spec.: Specification, **S.C:** Source Code,**Beh.M:** Behavioral Model, **EP:** End Point,**Point C.:** Point Cut,**Ref. Instr.:** Reference Instrument(i.e., Reference Data for Benchmarking),**Event S.:** Event Scenario,**Config.:** Configuration)

7.5 Framework Evaluation

To assess the completeness of our framework, we conducted a survey with software engineers experienced in IoT system development and testing. Additionally, we performed experiments on two case studies.

7.5.1 Survey

Conducting Survey

We obtained an ethics certificate for research involving human subjects. This survey aimed to gather insights from industry professionals to validate the practicality and relevance of our framework in real-world scenarios. The feedback collected offers a deeper understanding of practitioners' perspectives and the framework's potential for adoption in IoT system testing. We followed the steps below to conduct the survey.

- *Define Survey Objective*. The objective of this survey is to assess the completeness and

categorization of technical SE aspects in IoT systems.

- *Identify Target Participants*. We identified the target audience based on their relevance to the framework, including developers (i.e., software engineers), testers, and QA engineers involved in the development or testing of IoT systems.
- *Prepare Survey Questions*. We used a mix of quantitative and qualitative questions. Closed-ended questions with Likert scales (1–5) measured completeness and agreement, while open-ended questions gathered feedback in cases of disagreement. To ensure clarity, the questions were grouped into logical sections: Section 1 focused on demographics, and Section 2 addressed framework-specific questions.
- *Create the Survey*. We created the survey using Google Forms, and the link to the survey is online[1].
- *Conduct Pilot Survey*. We piloted the survey with four participants to ensure clarity and relevance. We refined the survey based on feedback from this small group before distributing it to a larger audience.
- *Distribute the Survey*. To identify participants, we searched for developers involved in IoT systems through GitHub commits and LinkedIn profiles. The survey was distributed to over 100 developers via various social media channels, primarily LinkedIn and Facebook IoT groups.
- *Analyze the Survey Data*. We manually analyzed the survey data and extracted participants' responses, reviewing any feedback in case of disagreements.
- *Integrate The Feedback*. We refined the framework using insights from the survey data and feedback.
- *Follow-up with Participants*. We conducted a follow-up survey after refining the framework, contacting participants who had reported disagreements to assess changes in their perceptions and gain their agreement.

[1] `https://shorturl.at/JxqGY`

Participant Demographics

Conducted over a 60-day period, we received feedback from 22 practitioners who reviewed the framework. Figure 7.6 presents the demographic details of the participants. The participants represented diverse roles, experience levels, IoT application domains, and geographical locations.

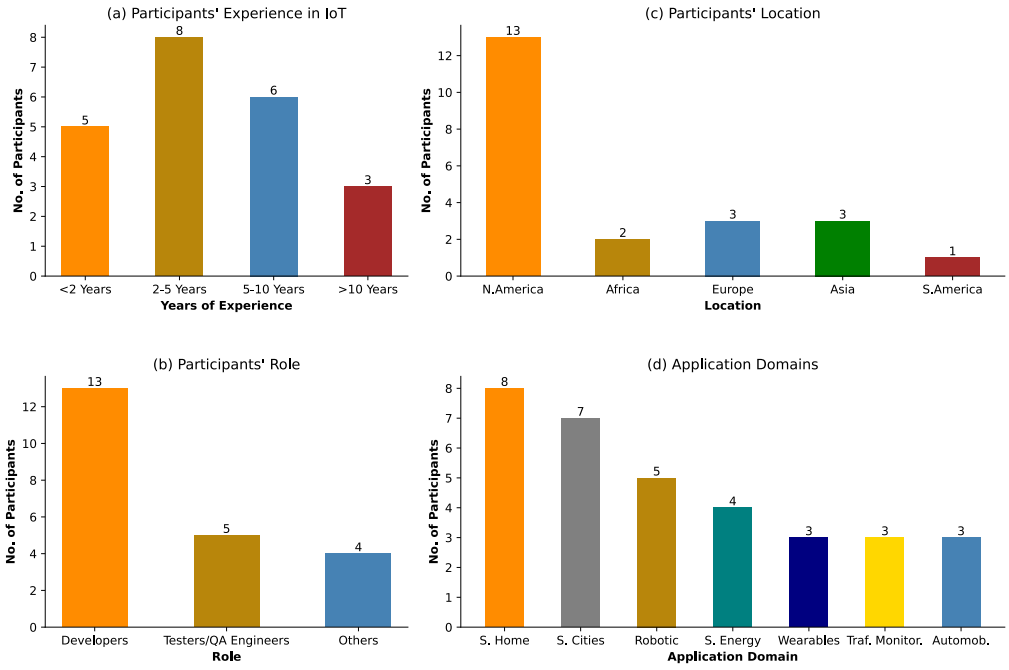


Figure 7.6: Participant demographics.

Participants' Roles. Figure 7.6 summarizes the role of the survey participants. The majority of the participants were Developers (59.09%, n=13), followed by Testers/QA Engineers(22.73%, n=5), and Other roles (18.18%, n=4).

Participants' Experience in IoT Systems. Figure 7.6 shows the experience level of our participants. Regarding experience, 36.36% (n=8) of the participants had 2-5 years of experience, 27.27% (n=6) had 5-10 years of experience, 22.73% (n=5) had less than 2 years of experience, and 13.64% (n=3) had more than 10 years of experience.

Participants' IoT Application Domains. Participants reported working in multiple

IoT application domains, as shown in Figure 7.6. The most common domains were Smart Homes (36.36%, n=8) and Smart Cities (31.82%, n=7). Other domains included Robotic (22.73%, n=5), Smart Energy (18.18%, n=4), and three equally represented domains: Wearables, Traffic Monitoring, and Automobile (13.64%, n=3 each).

Participants' Locations. Figure 7.6 shows the geographical location of our participants. In terms of geographical distribution, the majority of participants were from North America (59.09%, n=13). Other regions included Europe (13.64%, n=3), Asia (13.64%, n=3), Africa (9.09%, n=2), and South America (4.55%, n=1).

7.5.2 Case Studies

We conducted an experiment on WIMP as our system under test (SUT) across two case studies: (1) event logging and handling testing, and (2) data integrity testing. WIMP (Where Is My Professor) is an IoT-based system designed to help students track their professors' availability by collecting and analyzing location data from various IoT devices. WIMP integrates three core components: Fitbit Watch, Buddy Robot, and Mobile Phone, which work together to provide services to end users based on collected data and executed commands.

- Fitbit Watch: Collects real-time data such as location, activity, and physiological metrics (e.g., heart rate) from professors. The data is transmitted to the cloud via a paired Mobile Phone using the peerSocket messaging API.
- Mobile Phone: Serves as a communication bridge, enabling data transfer between the Fitbit Watch and the WIMP Server in the cloud.
- WIMP Server: Acts as the central application, processing data received from the Fitbit Watch and issuing commands to the Buddy Robot. It analyzes incoming information to determine appropriate actions for the Buddy Robot or responses for end users.
- Buddy Robot: Executes commands received from the WIMP Server, such as moving, rotating, speaking, or enabling its wheels, to deliver information or perform tasks.

Figure 7.7 illustrates our experimental setup.

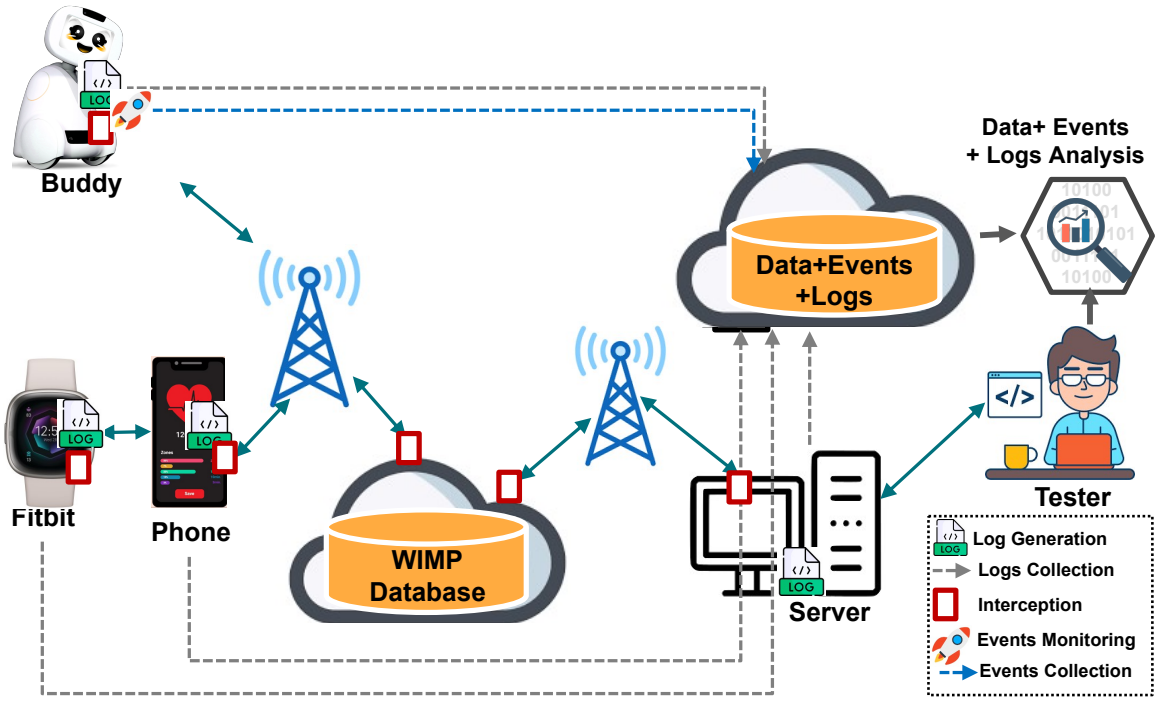


Figure 7.7: Experimental Setup

7.6 Results

In this section, we provide the answers to the following RQs.

- **RQ9:** What are testable technical SE aspects of IoT systems?
- **RQ10:** How complete are the testable technical SE aspects of IoT systems?
- **RQ11:** How practical is the framework for real IoT systems?

7.6.1 RQ9: What are testable technical SE aspects of IoT systems?

Figure 7.2 shows the high-level overview of TISSEA. TISSEA defines 19 technical SE aspects that are testable by software engineers in IoT systems. These aspects include functionality-related aspects such as functional testing, API testing, and event handling;

data-related aspects like data integrity, data synchronization, and data serialization; performance aspects, scalability aspects, energy efficiency, and power consumption aspects. TISSEA also includes aspects such as resilience, interoperability, heterogeneity, and latency. It also includes logging, basic security, connectivity, and device interaction aspects like compatibility, sensor accuracy, and device discovery. Each aspect in TISSEA is mapped to relevant test granularities, including unit testing, integration testing, component testing, component integration testing, and system testing across different IoT layers such as device, gateway, cloud, and application. TISSEA further guides test targets, input artifacts such as source code, specifications, and configuration files, execution methods including black box, white box, and gray box, and orchestration strategies such as simulation, monitoring, interception, and logging. These elements help software engineers understand which aspects fall within their testing responsibilities and how to systematically test them in IoT systems.

Takeaway for RQ9

TISSEA identifies 19 testable technical aspects in IoT systems and guides software engineers on testing IoT systems. It maps these aspects to testing levels, focus, scope, input artifacts, orchestration strategies, and execution methods.

7.6.2 RQ10:How complete are the testable technical SE aspects of IoT systems?

We now discuss the results of the survey conducted with 22 IoT practitioners, highlighting their feedback on the relevance and completeness of the technical aspects identified in TISSEA. The participants evaluated various technical SE aspects, test granularities, test targets, input artifacts, orchestration strategies, and execution methods using a 5-point Likert scale to indicate the extent to which each aspect was considered correctly identified and sufficiently complete. Table 7.1 summarizes the results, including mean ratings, standard deviations, t-statistics, p-values, and variances for each category.

Table 7.1: Framework Evaluation

Category	Mean	Std Dev	Variance	t-Statistic	p-Value
Tech. SE Aspects (Overall)	4.727	0.456	0.198	-2.806	0.011
Test Granularities	4.727	0.456	0.198	-2.806	0.011
Tech. SE Aspects for DL	4.864	0.351	0.118	-1.821	0.083
Tech. SE Aspects for GL	4.455	0.739	0.521	-3.464	0.002
Tech. SE Aspects for CL	4.636	0.581	0.322	-2.935	0.008
Tech. SE Aspects for AL	4.864	0.351	0.118	-1.821	0.083
Orchestration Strategies	4.818	0.395	0.149	-2.160	0.042
Execution Methods	4.955	0.213	0.043	-1.000	0.329
Inputs Artifacts	4.955	0.213	0.043	-1.000	0.329

✵ **DL**: Device Layer; **GL**: Gateway Layer; **CL**: Cloud Layer; **AL**: Application Layer.

The descriptive statistics show consistently high mean ratings across all technical aspects assessed, ranging from 4.455 to 4.955, indicating strong participant agreement overall. Statistical significance was determined based on a threshold of $p < 0.01$, where results with lower p-values indicate strong evidence of participant agreement beyond random chance.

- **Tech. SW Aspects and Test Granularities.** Both categories achieved mean ratings of 4.727 with low standard deviations (0.456), reflecting strong and consistent participant agreement.
- **Tech. Aspects for the Device Layer (DL) and Application Layer (AL).** These aspects recorded high mean ratings of 4.864 with minimal variability (0.351), indicating strong agreement across participants.
- **Tech. Aspects for the Gateway Layer (GL) and Cloud Layer (CL).** These aspects achieved mean ratings of 4.455 and 4.636, respectively, with higher variability (0.739 for GL and 0.581 for CL). Both categories recorded significant p-values ($p = 0.002$ for GL and $p = 0.008$ for CL), confirming statistically significant participant agreement despite more varied perceptions compared to other categories.
- **Orchestration Strategies.** This category recorded a high mean rating of 4.818 with low

standard deviation (0.395), supporting strong participant consensus.

- **Execution Methods and Inputs Artifacts.** Both categories achieved the highest mean ratings of 4.955 with very low variability (0.213), reflecting very strong and consistent participant agreement.

Overall, the survey results highlight strong participant consensus across all technical aspects, primarily reflected through consistently high mean ratings and low standard deviations. While the Gateway and Cloud layers demonstrated statistically significant agreement, they also exhibited higher variability compared to other categories.

Takeaways for RQ10

Strong and Consistent Agreement: Technical SE aspects, test granularities, device and application layer aspects, orchestration strategies, execution methods, and input artifacts achieved high mean ratings (above 4.7) with low standard deviations (below 0.5), reflecting strong and consistent participant agreement.

Agreement with Higher Variability: Gateway and Cloud layer aspects also received high mean ratings but exhibited higher standard deviations (0.739 and 0.581, respectively), suggesting strong agreement with slightly more varied participant perceptions, highlighting areas for further refinement.

7.6.3 RQ 11: How practical is the framework for real IoT systems?

We decompose this RQ into the following two sub-questions: (1) RQ11.1:Event Logging and Handling Testing and (2) RQ11.2: How can altered data be detected across different layers of an IoT system?

RQ 11.1: Event Logging and Handling Testing

We conducted event logging and handling tests on WIMP as our system under test (SUT) and answered two questions *RQ11.1.1:How can testers detect missing or incomplete logs in an IoT system?* and *RQ11.1.2:To what extent does system-generated event execution*

status align with the actual execution of the event? To answer RQ11.1.1, we collected the execution data from our SUT, and Table 7.2 summarizes our event logging findings.

Table 7.2: Expected vs. Captured Log Entries

	Expected	**Captured**	**Missing**
Device Layer	127	109	18
Application Layer	315	299	16

In the device layer, 18 log entries were missing. Among these, 14 were due to unexecuted actions (i.e., operations to be performed by SUT) caused by resource constraints when the device received multiple commands simultaneously, and 4 were due to the device hanging during execution.

In the application layer, 16 log entries were missing. Of these, 10 were caused by the device failing to execute actions and provide feedback, while 6 were due to connection failures between the application layer and the device.

Takeaway for RQ11.1.1

The findings reveal that some missing log entries in the device and application layers were potentially caused by factors such as resource constraints, execution failures, and connection issues, although the full range of causes remains uncertain. These results suggest that TISSEA can be useful in guiding testers to detect and investigate missing logs, helping to improve logging in IoT systems.

To answer RQ11.1.2, we collected the data from our SUT, and Table 7.3 shows our findings.

We considered tests involving 56 events on the Buddy robot and 59 events at the application layer. The application layer had more events because two were related to specific actions involving the Fitbit watch, while one was lost due to connectivity issues (i.e., network loss). Our focus was on events related to the Buddy robot, such as rotating, moving,

Table 7.3: Reported vs. Observed Events

		Execution Success	
	Executed	**Reported**	**Observed**
Device Layer	56	53	44
Application Layer	59	52	-

and speaking.

Based on the *execution status* reported by the system, 53 events were successfully handled (e.g., `WHEEL_ENABLE_FINISHED`, `WHEEL_MOVE_FINISHED`, `SPEAK_COMPLETED`, `MOVE_FINISHED`, `ROTATE_FINISHED`, etc.), while three events were not executed due to the device's inability to process new events before completing ongoing tasks. These unexecuted events were not included in the reported results.

For observed execution, we monitored the status of each event. Among the 53 events reported as successful by the system, only 44 were successful based on observation, while 9 were not. Of these 9, three were related to `ENABLE_WHEEL`, which could not be observed and were considered successful, leaving 6 events as incorrect results. For example, the system reported `MOVE_FINISHED` when the device moved only 1.7 meters instead of the required 2 meters, or `ROTATE_FINISHED` when it rotated 160 degrees instead of 180 degrees. These discrepancies indicate potential bugs in the device that cannot be detected without observation or other orchestration mechanisms.

For the application layer, 52 events were reported as successfully executed. However, one event reported as a success on the device did not have its execution status delivered to the application layer. Observation was not applicable for the application layer.

Takeaway for RQ11.1.2

The findings highlight the difference between system-reported and observed execution outcomes, with six incorrect event results on the Buddy robot, underscoring the need for observation-based testing and improved orchestration mechanisms to detect execution inconsistencies. These findings show that following TISSEA, testers can detect bugs by identifying mismatches between system-reported outcomes and the actual observed execution behavior.

RQ 11.2: Data Integrity Testing in IoT Systems

We experimented to evaluate how software engineers can test data integrity in IoT systems (i.e., *RQ11.2: How can altered data be detected across different layers of an IoT system?*). Data integrity ensures that data transmitted between IoT nodes remains unaltered and accurate. We acknowledge that some intermediate nodes may perform some computations or processing tasks. However, in our experiment, we assume that intermediate nodes do not modify the original data. We explored multiple techniques for detecting data modifications, including cryptographic hash functions (SHA-256), HMAC (Hash-based Message Authentication Code), digital signatures (RSA, ECDSA), TLS/SSL encryption, and redundant transmission. Each technique has its advantages and limitations for security and computational overhead. Given the resource constraints of IoT devices, we opted for a lightweight approach that eliminates the need for cryptographic computation, ensuring minimal resource consumption.

To test data integrity, we write a **cloud-assisted verification algorithm**, where each IoT node transmits a copy of its original data to the central cloud server along with a timestamp. The cloud server stores this original data and later verifies it against the data received at the next node. In our algorithm (Algorithm 5), if the node is the first in the network, it collects data and immediately sends a copy to the cloud before forwarding it to the next node. If the node is an intermediate node, it receives data from the previous

Algorithm 5 IoT Data Integrity Testing

```
1: function DataIntegrityTesting
2:     Let N ← Set of all IoT nodes
3:     Let D_r ← Data received by each node
4:     Let M_D ← Set of altered data entries
5:     Let D_c ← Cloud-stored data
6:     for each node n ∈ N do
7:         if n is the first node then
8:             Collect data d_n from node n
9:         else
10:            Receive data d_n from previous node n − 1
11:        end if
12:        Send d_n to the cloud server with timestamp t_n
13:        Perform processing task if any
14:        Send d_n to the next node n + 1
15:    end for
16:    for each node n ∈ N do
17:        for each received entry d_r ∈ D_r[n] do
18:            d_c ← D_c[t_n]
19:            if d_c ≠ d_r then
20:                M_D[n] ← d_r
21:                Report data integrity issue for node n
22:            end if
23:        end for
24:    end for
25: end function
```

node and forwards it further. The integrity verification occurs when the received data at each node is compared with the corresponding original data stored in the cloud using the timestamp. Any mismatch between the received and cloud-stored data indicates a data integrity issue. With this algorithm, we can ensure real-time integrity verification while minimizing computational overhead, making it an optimal solution for IoT environments. Guided by TISSEA, we defined the required input artifacts, test targets, test granularity, and orchestration strategy, which enabled us to detect data integrity violations.

We experimented and made nine interceptions on the phone, modifying some of the data received from the Fitbit as shown in Table 7.4. By comparing the received data with

Table 7.4: Data Integrity Check

		Phone		Rec. by Firebase		Rec. by App. From		
ID	**Fitbit**	**Received**	**After Interc.**	**Fitbit**	**Phone**	**Phone**	**Firebase**	**Status**
T1	"hr":84	"hr":84	"hr":84	"hr":84	"hr":84	"hr":84	"hr":84	Pass
T2	"hr":87	"hr":87	"hr":88	"hr":87	"hr":88	"hr":88	"hr":87	Fail
T3	"hr":83	"hr":83	"hr":85	"hr":83	"hr":85	"hr":85	"hr":83	Fail
T4	"hr":89	"hr":89	"hr":89	"hr":89	"hr":89	"hr":89	"hr":89	Pass
T5	"hr":86	"hr":86	"hr":87	"hr":86	"hr":87	"hr":87	"hr":86	Fail
T6	"hr":92	"hr":92	"hr":92	"hr":92	"hr":92	"hr":92	"hr":92	Pass
T7	"hr":85	"hr":85	"hr":86	"hr":85	"hr":86	"hr":86	"hr":85	Fail
T8	"hr":93	"hr":93	"hr":95	"hr":93	"hr":95	"hr":95	"hr":93	Fail
T9	"hr":88	"hr":88	"hr":88	"hr":88	"hr":88	"hr":88	"hr":88	Pass
T10	"hr":93	"hr":93	"hr":95	"hr":93	"hr":95	"hr":95	"hr":93	Fail
T11	"hr":91	"hr":91	"hr":91	"hr":91	"hr":91	"hr":91	"hr":91	Pass
T12	"hr":83	"hr":83	"hr":84	"hr":83	"hr":84	"hr":84	"hr":83	Fail
T13	"hr":80	"hr":80	"hr":83	"hr":80	"hr":83	"hr":83	"hr":80	Fail
T14	"hr":78	"hr":78	"hr":78	"hr":78	"hr":78	"hr":78	"hr":78	Pass
T15	"hr":79	"hr":79	"hr":80	"hr":79	"hr":80	"hr":80	"hr":79	Fail

✺ **hr** : Heart Rate Data; **After Interc.** : After Interception or Processing; **Rec. by Firebase** : Data Received at the Cloud Firebase; **Rec. by App. From**: Data Received by Application From;

the original data stored in our central cloud server, we detected data integrity violations. Table 7.5 presents the findings.

Table 7.5: Data Integrity Violation Results

		Violations	
Scenario	**Tests**	**Introduced**	**Detected**
W/O Violations	15	0	0
W. Violations	15	9	9

Takeaway for RQ11.2

The findings show that by storing the original data, we could detect any modifications made at intermediate nodes. Out of 15 sample tests executed, including 9 with forced interceptions, we successfully identified data integrity issues in our SUT by following TISSEA.

7.7 Discussion

In this section, we discuss key observations from the survey, case studies, and the implications of TISSEA for both researchers and practitioners.

7.7.1 Survey Evaluation

While deriving this framework, we extracted technical SE aspects for IoT systems not only from the taxonomy but also through input from industrial experts. Our survey results indicate that some of these aspects still require further exploration. Notably, connectivity and security testing are typically handled by specialized teams, such as network and security teams. However, we designed our framework to focus on aspects that software engineers can test directly. It is reasonable for software engineers to verify IoT device connectivity and data exchange with application or cloud layers. Similarly, they must ensure basic security measures like authentication and authorization, leaving core security tasks such as penetration testing and denial of service protection to security teams. Survey results show that while a few participants questioned the completeness of TISSEA specifically for testing at the cloud and gateway layers, all agreed on testing at the application and device levels within our framework. One possible reason for this feedback could be that software engineers may have limited knowledge of the cloud and network layers. Involving additional stakeholders such as network and cloud engineers could help clarify which responsibilities could reasonably fall under software engineering and how those responsibilities can be addressed in practice. These insights could be used to further refine and expand TISSEA to ensure broader applicability across IoT layers, which remains a focus of our future work.

7.7.2 Case Study Analysis

Our case studies revealed that combining multiple orchestration strategies helps uncover bugs that a single strategy might miss. For example, we observed a scenario where the system under test (SUT) received heart rate data and, based on the value, sent a command

to the Buddy robot to rotate 90 degrees at a specific speed. However, in practice, the Buddy robot rotated only 60–70 degrees, then stopped and reported successful execution. Relying solely on returned values could incorrectly mark this test as a pass when it is not.

Another critical aspect is data integrity, as IoT systems rely on data collected from multiple devices. Any compromise in this data can lead to incorrect system behavior, potentially endangering users. We experimented with using computed hash values at different stages of data exchange to detect unauthorized modifications. While this adds computational overhead, its importance in ensuring data integrity cannot be overlooked.

7.7.3 Implications of TISSEA for researchers and practitioners

This study has implications for both researchers and practitioners. For practitioners, the proposed framework clarifies the SE team's responsibilities in IoT system testing. It helps software engineers involved in testing IoT systems identify the specific aspects that fall within their responsibilities and understand what inputs, targets, and methods they should use to carry out those tests effectively.

For researchers, our study focuses on technical SE aspects, but many other domains require further investigation, including security, network, and infrastructure. Additional research is especially needed for the cloud and gateway layers, where the responsibilities of software engineers are less clearly defined. Involving other stakeholders such as network and cloud engineers, could help identify which aspects can be delegated to software engineers and how they can be addressed in practice. These efforts would be useful in updating TISSEA to make it more complete than its current state, ultimately serving as a complete guide for software engineers on the testing of IoT systems.

7.8 Threats to Validity

this chapter acknowledges several threats to validity, categorized into construct, internal, and external validity in our proposed framework and its validation.

Construct Validity. One potential threat to construct validity is the use of our taxonomy as a foundation for the framework. While we collaborated with industry partners to refine and expand the taxonomy, ensuring its relevance and completeness, there is still a risk that certain aspects may not fully capture all dimensions of IoT system testing. However, we continuously updated the taxonomy based on expert feedback and conducted an industry survey to validate its comprehensiveness.

Internal Validity. Bias in data analysis and conclusions presents a key internal validity threat. We mitigated this risk by using structured evaluation criteria and cross-validating our findings with multiple researchers. Additionally, the limited number of survey participants could impact the reliability of our findings, as recruiting experienced software engineers in IoT system development and testing is challenging. However, we carefully selected participants with relevant expertise to enhance the credibility of the results.

External Validity. The generalizability of our experimental results is a potential concern, as our evaluation was conducted on a single small-scale IoT system. Due to the difficulty of accessing open-source IoT systems with sufficient complexity, our findings may not fully represent large-scale industrial IoT environments. To address this limitation, we supplemented our experimental results with insights from industry professionals through surveys, increasing the practical relevance of our study. While these threats exist, we have taken appropriate measures to minimize their impact and enhance the reliability of our findings.

7.9 Conclusion

This chapter presented TISSEA, a framework designed to guide software engineers in testing technical software engineering aspects of IoT systems. Building on an IoT system testing taxonomy, we identified and mapped key testable aspects to test granularities, test targets, test input artifacts, test orchestration strategies, and test execution methods.

We conducted two types of evaluations: a survey with 22 professionals and two case

studies. These evaluations demonstrated that TISSEA can effectively guide software engineers in conducting testing at the device and application layers. However, further investigation is needed to support testing at the gateway and cloud layers. Experimental findings showed that by following TISSEA, software engineers were able to detect issues related to logging and event handling. Additionally, they could identify data integrity issues during transmission between nodes. These findings indicate that TISSEA helps software engineers understand which technical aspects of IoT systems fall within their responsibilities and provides guidance on how to test them.

The future work should focus on refining the framework and conducting additional case studies to evaluate other technical aspects of IoT systems using a broader set of real-world systems.

In the next chapter, we follow this framework by focusing on end-to-end functional testing of IoT systems, selecting one of the aspects identified.

Chapter 8

Approach for Functional E2E Testing of IoT Systems

8.1 Introduction

Building on the findings from the previous chapter, where we identified key testable aspects of IoT systems in software engineering (SE), this chapter investigates automated test generation for functional end-to-end (E2E) testing of IoT systems. Functional testing verifies that a system behaves according to its specified functional requirements. E2E functional testing ensures that data flows correctly from devices to gateways, cloud, and applications and that the expected outcome is achieved. To systematically assess different test generation options, we explored four approaches based on use case specifications (UCSs):

- **Custom Approach (CA)**: A structured keyword-based technique for generating tests.
- **Single-Stage LLM Approach (SSLA)**: A direct test generation using a single LLM prompt.
- **Multi-Stage LLM Approach (MSLA)**: An LLM process with multiple sequential stages.
- **Hybrid Approach (HA)**: A combination of structured keyword-based technique and LLM-based generation.

This comparison aims to answer **RQ12: How do test generation approaches compare for E2E testing of IoT Systems?** Our preliminary findings indicate that while LLM-based approaches demonstrate potential to automate test generation, they can introduce some unpredictability and inconsistencies in test correctness and completeness which requires some manual intervention. These findings motivate further investigation of LLM-based test generation, leading to **RQ13: How reliable are LLMs in E2E test generation for IoT systems?**

We proposed TGenAI, an LLM-based approach to generate functional E2E tests for IoT systems, and conducted an empirical evaluation, analyzing its strengths and weaknesses in IoT system testing. While LLMs demonstrated the ability to generate tests, they also suffered from inconsistent scenario coverage. Given these limitations, we sought a more systematic approach to ensure consistency in test generation.

We introduced FUNEETIS, a systematic, custom approach for functional end-to-end testing of IoT systems. FUNEETIS systematically derives test cases directly from UCSs by leveraging structured, pattern-based statements within the given specifications. Additionally, we define IoT-specific test metrics to assess test effectiveness and the bug detection capability of the generated tests.

FUNEETIS builds on prior research in test generation for IoT and embedded systems (i.e., [104, 106, 74, 191]). FUNEETIS supports functional end-to-end test generation for IoT systems from UCSs written in natural language and executes the generated tests to detect bugs. It first converts UCSs into a structured, machine-readable format such as JSON. Based on this structured representation, we construct a Use Case Test Model (UCTM) to systematically derive test scenarios, extracting key elements such as actions, constraints, and execution targets (test runners). The extracted information, including test inputs, expected outputs, constraints, and scenarios, is then used to generate test data. This test data,

along with the system under test (SUT) description and a manually defined test oracle, enables the automated generation of end-to-end tests that cover all layers of the IoT system under study. Furthermore, FUNEETIS generates instrumentation code for each IoT node based on the SUT description. This code is injected at predefined execution points to capture runtime execution data, which is then stored in a central database for further analysis. The generated tests are executed against this real-time execution data to detect potential bugs. To evaluate the effectiveness of FUNEETIS, we conducted an empirical evaluation on WIMP to answer two more research questions (i.e., **RQ14: How can functional end-to-end test cases be generated for IoT systems?** and **RQ15: How can generated tests be executed to detect bugs in IoT systems?**) The results demonstrate the ability of FUNEETIS to systematically generate and execute functional end-to-end tests, addressing the limitations observed in LLM-based test generation. Generally, this chapter answers the following research questions (RQs):

- RQ12: How do test generation approaches compare for IoT E2E testing?
- RQ13: How reliable are LLMs in E2E test generation for IoT systems?
- RQ14: How can functional end-to-end test cases be generated for IoT systems?
- RQ15: How can generated tests be executed to detect bugs in IoT systems?

This chapter presents the following key contributions:

- We propose various approaches to generate IoT specific tests from given UCSs, evaluate their accuracy, and identify directions for potential further investigation.
- We present TGenAI, an LLM-based approach that generates diverse testing artifacts, including constraints, test scenarios, and test cases, and validate it through an empirical study in a real world IoT scenario, and highlight its limitations.
- We present a novel testing approach, FUNEETIS, which systematically generates end-to-end tests of IoT systems from use case specifications (UCSs).
- We adopt the Restricted Use Case Modeling (RUCM) approach from [204, 205] and

extended it by introducing new keywords tailored for IoT system specifications. These additions facilitate the automation of test generation by ensuring that key IoT-specific elements, such as device interactions, protocols, and actions/operations, are explicitly captured in the specifications.

- We propose a template for describing the IoT system, capturing interactions between nodes and execution paths. This format enables code generation for each node involved, thereby improving code instrumentation.
- We propose five metrics for end-to-end functional testing to quantitatively evaluate the effectiveness of end-to-end tests, focusing on their accuracy in covering key aspects of IoT systems.
- We provide four algorithms implemented in FUNEETIS: (1) an algorithm that processes use case specifications to generate test scenarios, (2) an algorithm that derives test data from these test scenarios, (3) an algorithm that generates test cases from the test data, and (4) an algorithm that generates instrumentation code that captures runtime data from each IoT node. These algorithms automate and enhance the test generation process for IoT systems.
- We empirically evaluate FUNEETIS on a real-world IoT system, "WIMP", demonstrating its effectiveness in identifying potential bugs and improving test coverage for IoT systems.

8.1.1 Challenges Under Investigation

Despite the existence of a few studies on testing IoT systems, practitioners continue to face several challenges. Recent surveys (e.g., [128, 64, 127]) have identified several challenges in IoT systems testing. This chapter focuses on the following key challenges associated with IoT system testing:

- **Challenge 1 (C1) - End-to-End Testing Complexity:** Conducting end-to-end testing in IoT systems is difficult because the source code for individual components is often

unavailable, preventing traditional unit or integration testing. Furthermore, some IoT devices lack the capability to execute conventional tests such as unit and integration tests, as typically done in traditional systems, due to hardware constraints. Therefore, test approaches that do not require source code access are essential for comprehensive end-to-end testing.

- **Challenge 2 (C2) - Handling Dynamic Device Interactions and Unpredictable Environment:** In IoT systems, devices interact with each other and their environment in complex ways, making it challenging to predict their behavior. These interactions are often dynamic, influenced by real-time factors such as network variability, internet usage, and differences between real and simulated environments. Traditional approaches rely mainly on simulation-based testing and assume a static execution flow, overlooking the heterogeneous technologies and runtime environments typical of IoT systems, making them unsuitable for end-to-end IoT testing. Figure 8.1 illustrates an example of the complexity of IoT systems.
- **Challenge 3 (C3) - Heterogeneity in IoT Systems:** IoT environments consist of diverse technologies, devices, protocols, and communication mechanisms, leading to significant challenges in test generation and execution while identifying the bugs. The variability of components makes it difficult to create a unified approach that effectively addresses all possible interactions and scenarios.
- **Challenge 4 (C4) - Complexity of Test Instrumentation:** IoT systems consist of diverse devices, each using different technologies, protocols, and execution environments. This heterogeneity makes test instrumentation challenging, as there is no standardized way to execute tests uniformly across all components.
- **Challenge 5 (C5) - Limitations of RUCM for IoT Systems Test Automation:** The Restricted Use Case Modeling (RUCM) format is widely used for specifying system behaviors in structured natural language. However, RUCM was not originally designed

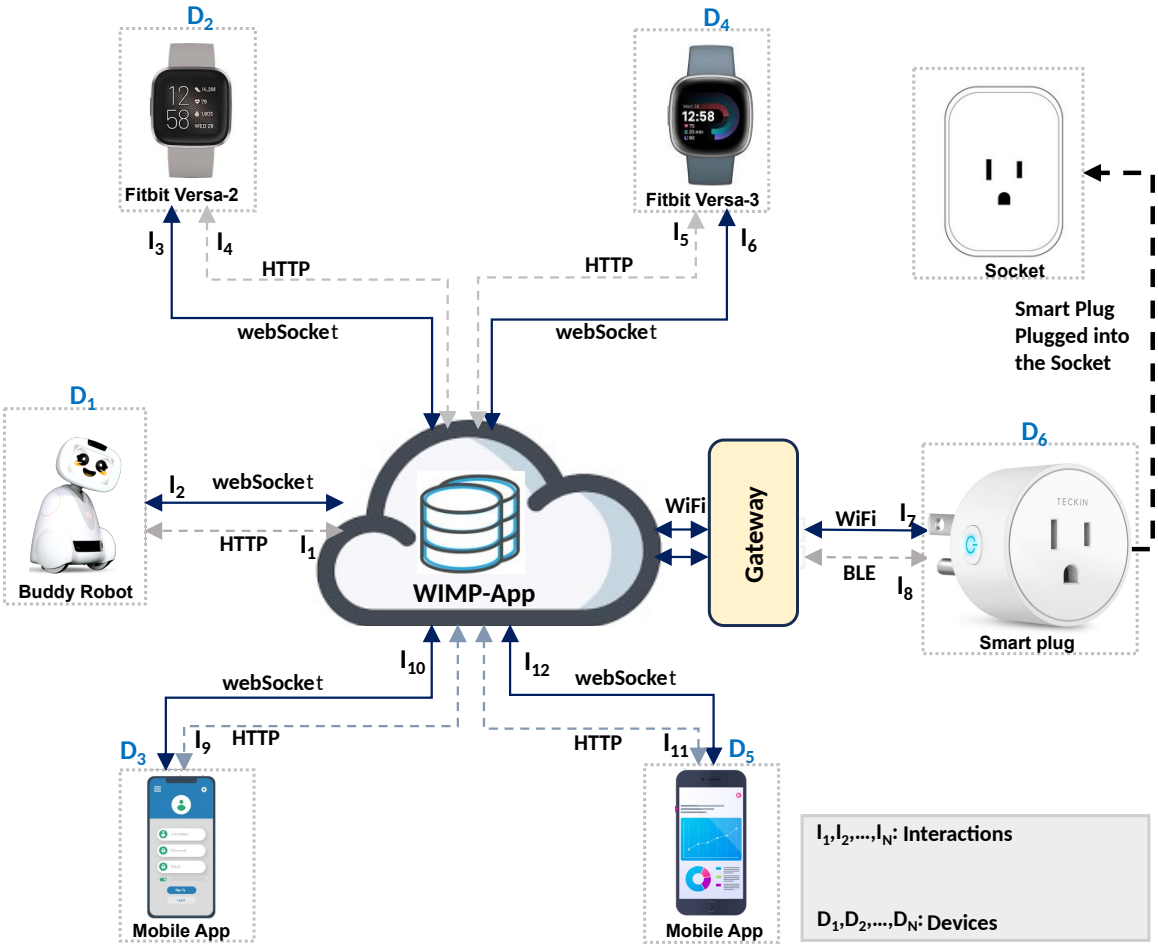


Figure 8.1: Interactions in IoT System

for IoT systems, making it insufficient for capturing critical IoT-specific details needed for automated test generation.

- **Challenge 6 (C6) - Lack of Standardized Metrics for IoT Testing:** Evaluating the effectiveness and efficiency of IoT testing remains a challenge due to the absence of well-established, universally accepted metrics. Existing proposals, such as the four metrics introduced by Hu et al. [74], do not fully capture the unique requirements of end-to-end functional testing for IoT. This gap highlights the need for specialized metrics tailored to the evaluation of end-to-end tests of IoT systems.

8.1.2 Assumptions

In this chapter, we propose an approach for end-to-end test generation and execution for IoT systems together with some metrics for test evaluation based on the following assumptions:

- **Assumption 1 (A1) - Compliance with Use Case Specification (UCS) Templates:** The UCS strictly follows the proposed templates, ensuring consistency in structure and format to facilitate automated test generation.
- **Assumption 2 (A2) - Instrumentation of Execution Nodes:** Developers are willing to inject instrumentation code into each execution node if it is not feasible to use other techniques such as Aspect-Oriented Programming (AOP) for automated instrumentation.
- **Assumption 3 (A3) - Test Execution Data Collection:** For each test execution, relevant data must be collected and stored in a dedicated test database. Each execution record must include an assigned test identifier for traceability and analysis.
- **Assumption 4 (A4) - Human-In-the-Loop (HITL) with Technical Knowledge:** Any Human-In-the-Loop (HITL) involved in the testing process must possess technical knowledge of the System Under Test (SUT) to ensure meaningful and accurate feedback.
- **Assumption 5 (A5) - Reserved Keywords in UCS:** The introduced keywords in the UCS are treated as reserved keywords to prevent ambiguity during test generation.
- **Assumption 6 (A6) - Deferred Specification of Execution Targets:** When defining UCS, precise details regarding execution targets, such as communication protocols, host addresses, and ports may not be fully determined at an early stage. These details are assumed to be provided later by the development team.

The rest of this chapter is as follows: Section **??** describes our approach. Section 8.2.4 discusses the proposed metrics for test evaluation. Section 8.3 presents our empirical evaluation results. Section 8.4 discusses our findings. Section 8.5 presents possible threats that could affect the validity of our chapter. Finally, Section 8.6 concludes with future work.

8.2 Research Method

Our research method is depicted in Figure 8.2 and consists of three main steps. First, the

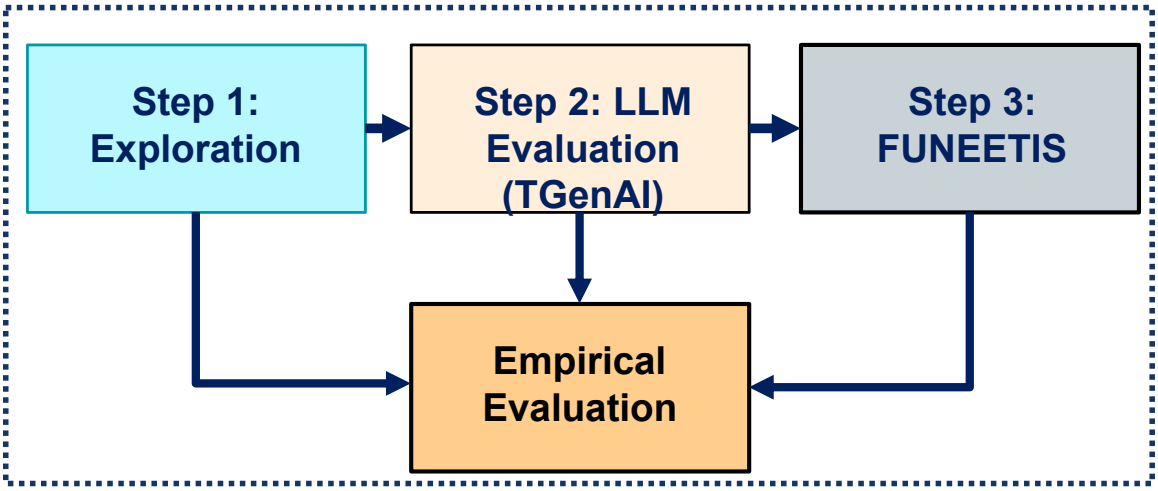


Figure 8.2: Research Method

Exploratory Phase compares four different test generation approaches based on use case specifications (UCSs) to evaluate their effectiveness. Second, the **LLM Evaluation** phase conducts an in-depth empirical assessment of LLM-based test generation to determine its reliability. Third, the **FUNEETIS Approach** introduces and evaluates a systematic custom approach that does not rely on AI or NLP techniques. For each step, we conduct an empirical evaluation to assess its effectiveness. In the subsequent sections, we describe each step in detail.

8.2.1 Exploratory Study

We explore 4 approaches to generate tests from UCS, each offering distinct characteristics and benefits: Custom Approach (CA), Single-Stage LLM Approach (SSLA), Multi-Stage LLM Approach (MSLA), and Hybrid Approach (HA).

Custom Approach (CA)

The Custom Approach (CA) shown in Figure 8.3 uses logic trees and parsing algorithms to extract data directly from use case structures. Although partially inspired by recent studies on test generation from UCS for embedded systems [191], this approach does not use any advanced AI or NLP techniques. The test data generation approach follows a structured process that begins with **Use Case Specification (UCS) Conversion**, where system interactions, conditions, and constraints are documented using a standardized template and transformed into a structured format for automated processing. Next, **Scenario Extraction** identifies all possible execution paths by analyzing basic, alternative, and exceptional flows, ensuring comprehensive coverage of system behaviors. Finally, **Test Data Generation** derives structured inputs from these scenarios, extracting key parameters such as operations, input values, target components, and expected outcomes. The generated test data are formatted into structured payloads, making it suitable for automated validation. This approach has the potential to generate test input while ensuring alignment with documented execution logic, with further details provided in one of the next chapters. If the

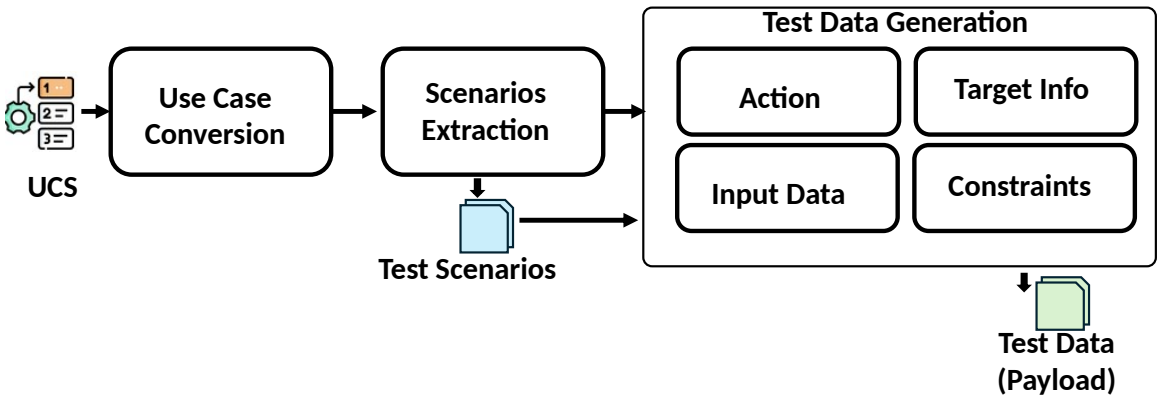


Figure 8.3: Custom Approach (CA)

input use case is complete, correctly formatted, and defect-free, CA should ensure accurate and exhaustive scenario coverage. By reorganizing formal data and generating values

from precise constraints, perfect inputs yield perfect outputs, offering reliability, consistency, and comprehensive coverage. However, the approach is highly sensitive to input quality (i.e., UCSs), with any errors or omissions affecting the results. This approach relies on strictly well-written and well-structured use case specifications (UCSs) to enable accurate extraction of execution details. If the UCS is complete and correctly formatted, the approach ensures accurate and exhaustive scenario coverage by systematically extracting execution flows. However, any errors, inconsistencies, or missing information in the UCS can lead to incorrect or incomplete test data generation.

Single-Stage LLM Approach (SSLA)

The SSLA shown in Figure 8.4 uses an LLM to generate tests in a single, detailed prompt that includes the use case, test format, and detailed instructions. The use case specification is inserted into a prompt and sent to ChatGPT-4o via its user interface. While flexible and effective for simple use cases, it may struggle with complex scenarios, yielding inconsistent results and limited scenario coverage.

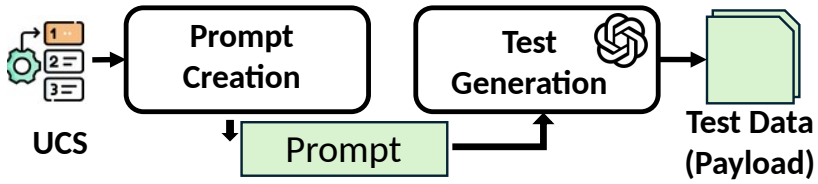


Figure 8.4: Single-Stage LLM Approach (SSLA)

Multi-Stage LLM Approach (MSLA)

MSLA shown in Figure 8.5 also uses LLMs, but divides the process into multiple prompts for greater precision and granularity. Reducing the workload per prompt minimizes errors, misunderstandings, and omissions. It first takes the use case from its JSON file and uses it in the prompt, which is sent to ChatGPT-4o via the OpenAI API to identify all scenarios. The API response is then analyzed to separate the scenarios. Each scenario is sent back in a prompt to ChatGPT-4o, and each response contains test data (i.e., payload). This approach

retains SSLA's flexibility while improving accuracy.

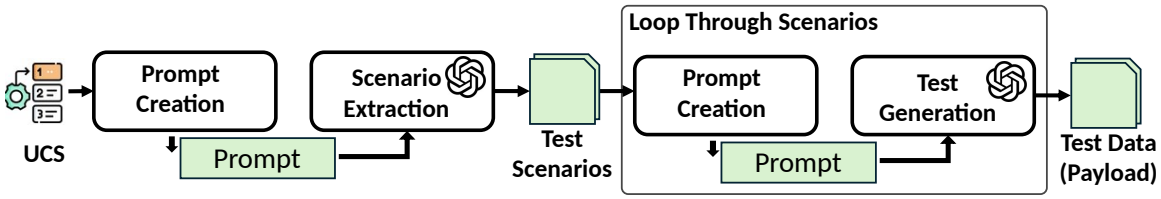


Figure 8.5: Multi-Stage LLM Approach (MSLA)

Hybrid Approach (HA)

The HA shown in Figure 8.6 combines elements of CA and LLM-based methods, leveraging the strengths of both. The first two steps mirror CA exactly, while the final two mirror MSLA instead, using ChatGPT-4o via the OpenAI API. This hybrid model aims to balance reliability, consistency, and flexibility, offering an improved user experience. Though still in its exploratory phase, the implementation highlights the potential of HA. Exploratory

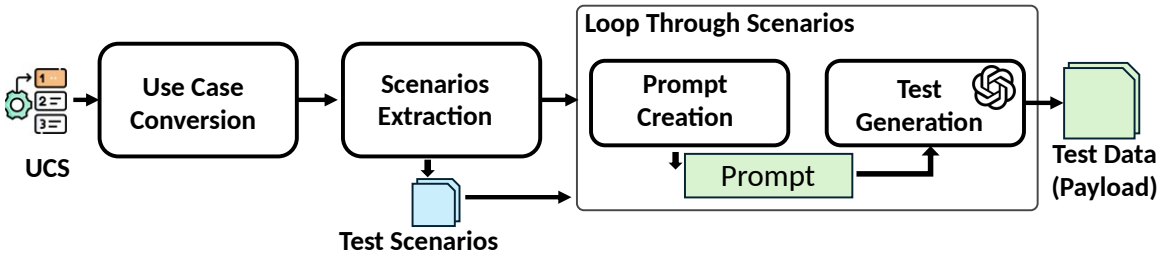


Figure 8.6: Hybrid Approach (HA)

step answers **RQ12: How do test generation approaches compare for E2E testing of IoT Systems?**

8.2.2 TGenAI Approach

In this section, we focus more on the proposed LLM-based approach, TGenAI. TGenAI generates end-to-end tests based on a provided prompt, which includes both the use case specification and an example test case. TGenAI uses SAMMO to optimize the initial prompt and enhance its clarity for better LLM comprehension. Listing 1 shows an example of the expected tests to be generated. We refer to this test as an end-to-end test because

it covers the entire flow from the sensing device, such as a Fitbit watch, to the gateway (a smartphone in this case), then to the cloud, where the cloud application determines the action to be performed by the Buddy robot. Due to the comprehensive nature of this test, we classify it as an end-to-end or system-level test.

```
{
  "1": {
    "TC_ID": "TBD",
    "name": "TBD",
    "steps": [
      {
        "entity": "fitbit",
        "operation": "sendHeartData",
        "inputs": {"heartrate": "80 bpm"},
        "target": {"protocol": "http","name": "smartphone"},
        "expectations": {"Execution_status": "OK"}
      },
      {
        "entity": "smartphone",
        "operation": "sendHeartData",
        "inputs": {"heartrate": "80 bpm"},
        "target": { "protocol": "websocket","name": "cloudapp"},
        "expectations": {"Execution_status": "OK" }
      },
      {
        "entity": "cloudApp",
        "operation": "sendMoveCommand",
        "inputs": {"distance": "1","speed": "1"},
        "target": {"protocol": "websocket","name": "buddy"},
        "expectations": {"Execution_status": "OK" }
      },
      {
        "entity": "buddy",
        "operation": "sendMoveStatus",
        "inputs": { "execution_status": "move_finished" },
        "target": {"protocol": "websocket","name": "cloudapp"},
        "expectations": {"Execution_status": "MOVE_FINISHED"}
      }
    ]
  }
}
```

Listing 1: Part of Sample End-to-End Test

TGenAI involves the following steps: (1) prompt formulation, (2) prompt optimization, (3) LLM invocation using API, (4) evaluating the generated test, (5) refining the prompt if the generated test does not meet expectations, and (6) refining the test for execution. These steps are outlined in Figure 8.7 and explained in subsequent sections.

- **Prompt Formulation**. Prompt formulation is a critical aspect of our approach, structured

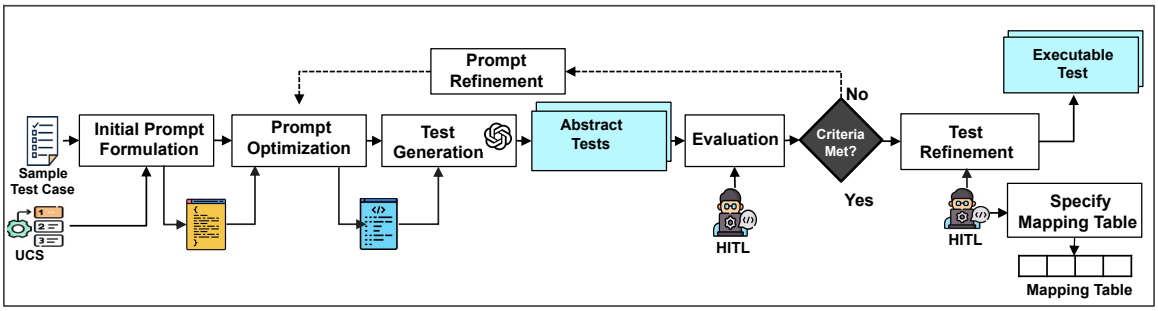


Figure 8.7: TGenAI Overview

to guide the LLM in generating accurate and relevant test cases. A prompt typically comprises three key components: (1) the question or task we want the LLM to address, (2) the input, such as a use case specification or source code, and (3) a few examples illustrating the expected output from the LLM. While our focus is on generating test cases from use case specifications, we have also tested using source code as input. In this case, the model often performs well without needing some examples (zero-shot) [192].

It is essential to emphasize that for complex systems, such as IoT systems, which involve heterogeneous devices, diverse protocols, and multiple technologies, providing sample test cases as part of the prompt is often necessary. Without such examples, the generated tests may not be executable or relevant. In this study, we adapted the test case format from [191], originally designed for embedded systems, to better accommodate the complexity of IoT systems. This modified format, which supports the specific needs of IoT testing, is illustrated in Listing 1.

- **Prompt Optimization**. Large Language Models (LLMs) can now process longer and more complex inputs, enabling the use of more elaborate prompts. However, even with these advancements, prompts often require tuning to enhance performance in real-world applications. Tobias et al. [164] proposed Structure-Aware Multi-objective Metaprompt Optimization (SAMMO). SAMMO is an innovative open-source tool designed to enhance prompt optimization by allowing practitioners and researchers to refine prompts dynamically with minimal manual intervention. Unlike traditional methods that treat

prompts as static inputs, SAMMO introduces the concept of metaprompts, allowing prompts to function as dynamic, programmable entities that can be seamlessly modified by removing or replacing components. Figure 8.8 shows three main aspects that can be optimized for every prompt by SAMMO.

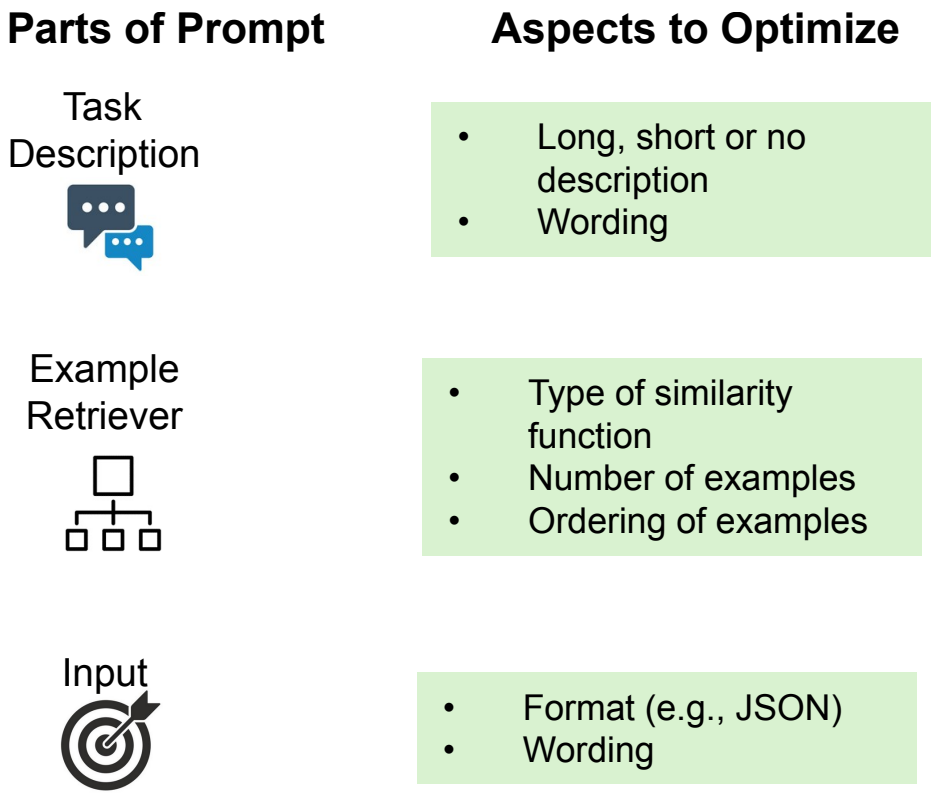


Figure 8.8: Parts of the Prompt that Can Be Optimized

– **Task Description:** The task description provides a static, high-level explanation of the problem to be solved. It remains independent of the specific input but can be optimized through paraphrasing, reordering, or refining wording to improve clarity and effectiveness.
– **Example Retriever:** This component retrieves relevant past examples to enhance the context provided to the language model. Optimizing this step involves selecting the most relevant examples using similarity functions (e.g., cosine similarity, semantic embeddings) and adjusting the number or formatting of retrieved examples.

– **Input:** The input consists of the user's query or prompt that requires processing. Optimization strategies include formatting (e.g., structured JSON vs. plain text), restructuring (e.g., breaking down complex inputs into simpler parts), and refining for clarity.

It is worth noting that if the initial prompt is already well-structured, the benefits of SAMMO may be less pronounced; however, when prompts lack structure, omit relevant examples, or contain poorly formatted inputs, SAMMO plays a crucial role in optimizing and refining them.

- **LLM Invocation**. We developed a tool that uses LLM APIs to generate tests from a given prompt. The specific segment of code used to call the API is shown in Listing 2.

```
public static String callLLM(String prompt) {
      String url = "https://api.openai.com/v1/chat/completions";
      String apiKey = "<VALID-KEY>";
      String model = "gpt-4o";
      try {
          URL obj = new URL(url);
          HttpURLConnection con = (HttpURLConnection) obj.openConnection();
          con.setRequestMethod("POST");
          con.setRequestProperty("Authorization", "Bearer " + apiKey);
          con.setRequestProperty("Content-Type", "application/json");
          String body = "{\"model\":\"" + model + "\", \"messages\":[{\"role\":\"user\",
          ↪  \"content\":\"" + prompt + "\"}]}";
          con.setDoOutput(true);
          OutputStreamWriter writer = new OutputStreamWriter(con.getOutputStream());
          writer.write(body);
          writer.flush();
          writer.close();
          // Check the response code
          int responseCode = con.getResponseCode();
          if (responseCode != HttpURLConnection.HTTP_OK) {
              <<do-something>>
          }
          // Get the response
          BufferedReader in = new BufferedReader(new
          ↪  InputStreamReader(con.getInputStream()));
          String inputLine;
          StringBuffer response = new StringBuffer();
          while ((inputLine = in.readLine()) != null) {
              response.append(inputLine);
          }
          in.close();
}
```

Listing 2: Segment of Code to Invoke LLM using API

- **Evaluation**. The engineer (i.e., Human-In-The-Loop (HITL)) manually evaluates the

generated tests based on coverage, completeness, and correctness using predefined criteria. For coverage, the engineer ensures all scenarios, input ranges, expected results, actions, protocols, devices, and interactions. For completeness, HITL checks that no information is missing in the generated test. For correctness, HITL confirms that the generated tests are syntactically valid.

- **Prompt Refinement** if the generated tests do not meet expectations, iteratively refine the prompt and regenerate tests.
- **Specify Mapping Table**. HITL creates a mapping table to match the patterns identified in the generated test cases with the corresponding operations and inputs in the implementation of the system under test (SUT). Table 8.1 provides an example of this mapping table.

Pattern in Generated Test		**Result in Executable Test**
Input	"angle":"90", "speed":"120"	angle=90.0f, speed=120.0f
Operation	"Buddy rotates"	rotate(120.0f,90.0f)
Input	"speed":"140", "distance":"2"	distance=2.0f,speed=140.0f
Operation	"Buddy moves"	move(140.0f,2.0f)

Table 8.1: Part of the Mapping Table for WIMP Test

- **Test Refinement for Execution**. Convert the validated test cases into executable test scripts by structuring them for automation frameworks or test execution tools.

It is worth noting that the test case generation process is semi-automated through an API. However, due to the complexity of IoT systems, human-in-the-loop (HITL) involvement remains essential. HITL refines prompts to enhance scenario coverage and ensures that the generated tests are meaningful, executable, and aligned with the actual system under test.

TGenAI step answers **RQ13:How reliable are LLMs in E2E test generation for IoT systems?**

8.2.3 FUNEETIS Approach

This section presents FUNEETIS, our custom test generation approach designed to address the limitations observed in TGenAI, particularly the inconsistencies in generated results and the extensive Human-in-the-Loop (HITL) effort required to ensure correctness and completeness. FUNEETIS adopts a structured and deterministic approach that integrates runtime execution data capture with real-world scenario validation. Specifically, it captures execution behavior during system operation, generates corresponding test cases, and validates them against the recorded data to check the behavior of the system under test (SUT). To support structured test derivation, we adopt the Restricted Use Case Modeling (RUCM) template. RUCM has been shown effective in previous studies for specifying use case scenarios with well-defined constraints. We further enhance RUCM with new domain-specific keywords that facilitate the automated extraction of relevant information needed for test generation. This enhancement improves the precision and consistency of the generated tests, enabling systematic coverage of possible scenarios. Figure 8.9 shows the overview of FUNEETIS. The implementation of FUNEETIS is publicly available in our GitHub repository[1].

Restricted Use Case Specification Template

We based our chapter on the RUCM template proposed by Yue et al. [204] for restricted use case specifications (UCS) and added new keywords for documenting the requirements of IoT systems. Given the heterogeneity of devices, protocols, and communication flows, it is essential to apply standardized methods for documenting use case specifications to ensure effective automated testing and reproducibility of results. We introduced several keywords to facilitate the extraction of actions, target information, and constraints. Table 8.3 provides an example of the specifications of one use case in this chapter.

[1] `https://github.com/ptidejteam/funeetis-FUNEETIS`

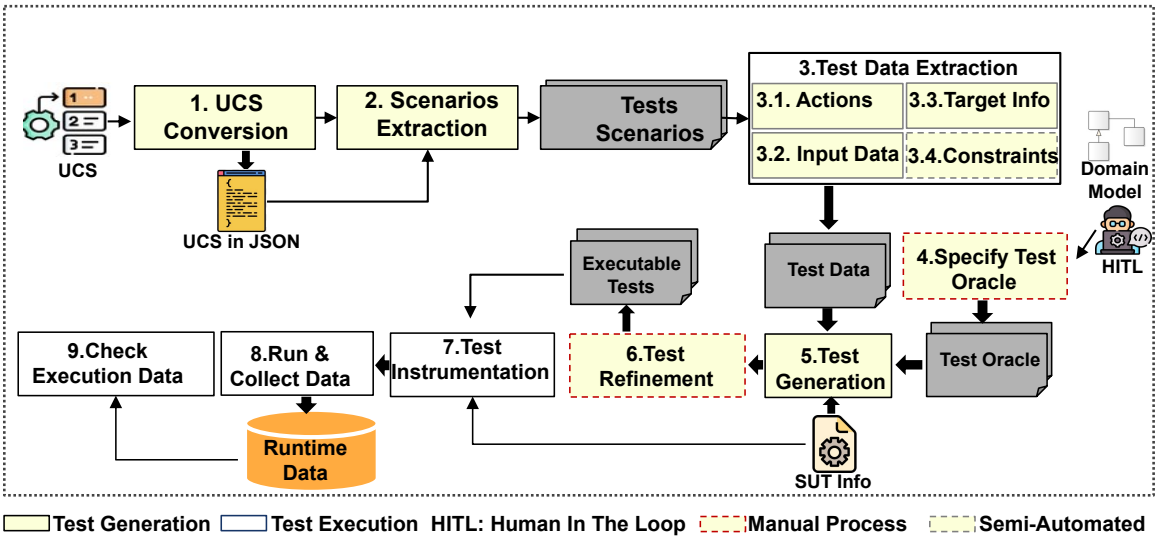


Figure 8.9: Overview of FUNEETIS

Patterns for IoT Use Case Specifications

We categorized each statement (i.e., step) in a use case specification into four main patterns. Table 8.2 shows the sentence structure for each pattern.

Table 8.2: Patterns for Statement in UCS Flow

#	Pattern
S1	{NODE1} SENDS {COMMAND/FEEDBACK/DATA} [WITH {INPUTS}] TO {NODE2} OVER {PROTOCOL}
S2	{NODE1} RECEIVES {COMMAND/FEEDBACK/DATA} [WITH {INPUTS}] FROM {NODE2} OVER {PROTOCOL}
S3	{NODE} {ACTION}{ATTRIBUTE} [WITH {INPUTS}][IS {CONSTRAINT}]
S4	{ATTRIBUTE1} IS {VALUE1},[{ATTRIBUTE2} IS {VALUE2}, . . .]

✺ **S1**: SENDING step; **S2**: RECEIVING step; **S3**:PROCESSING step; **S4**:POSTCONDITION.

- **Interaction Statement (i.e., S1 and S2)** represents two nodes exchanging data/command, identified as the sender and receiver.
  - **Explanation:** In statements where two nodes interact, the structure starts with the *(Node1)*, followed by the *action verb* such as SENDS or RECEIVES, then the *qualifier* to represent any additional details including data, command, feedback or input parameters needed if any to fully define the action. The use of keywords [FROM/TO] are followed by *Node2*. To include the protocol that will be used, we

introduced the keyword *OVER*.

  - **Example for S1:** WIMP SENDS Move_command WITH distance=25, speed=30 TO Buddy OVER MQTT
  - **Example for S2:** Buddy RECEIVES Move_command WITH distance=25, speed=30 FROM WIMP OVER MQTT

- **Processing/Computation Statement (S3) (No interaction)** represents internal computation or command execution within a single node, without involving interaction with other nodes.
  - **Explanation:** For processing or computation statement, the structure starts with a *Node* performing the required action, followed by *action*, and any *qualifier*. The qualifier represents any additional details including data, command, feedback or input parameters needed to fully define the action.
  - **Example1:** WIMP VALIDATES if heart rate IS between 60 and 100bpm
  - **Example2:** BUDDY PERFORMS move with distance=25, speed=30
- **Postcondition Statement (S4)**
  - **Explanation:** Postcondition statements start with left operand to represent a system property, attribute, or variable whose state is being described in the postcondition. This is the subject of the statement, such as MOVE_STATUS, which reflects the system's state after execution. This is followed by the **operator** to represent the relational or assignment action indicating the relationship between the left operand and the right operand. For example: the relational operator: "is," "is not," "equals," "not equals," etc., to evaluate or assert a condition. This is followed by **Right Operand** to represent the value that defines the expected result after the execution is completed.
  - **Example:** move_execution_status IS OK

Table 8.3: Part of WIMP Use Case Specifications

Use Case: Fitbit Sends Data and Buddy Moves
1.1 Precondition
EnableWheels is 0
Mood is 1
1.2 Basic Flow
1. Fitbit **SENDS** heart data to SmartPhone with heartrate=80 bpm Over HTTP
2. SmartPhone **RECEIVES** the heartrate data From Fitbit
3. SmartPhone **SENDS** heart data with heartrate=80 bpm to CloudApp over WebSocket
4. CloudApp **RECEIVES** heartrate data FROM fitbit Over HTTP
5. CloudApp **CHECKS THAT** heartrate is between 60 and 100 bpm
6. CloudApp **SENDS** MOVE command to Buddy with distance=1 and speed=1 Over WebSocket
7. Buddy **RECEIVES** MOVE command from CloudApp
8. Buddy **SETS** EnableWheels with leftWheel=1 and rightWheel=1
9. Buddy **MOVES** at specified distance and speed.
10. Buddy **SENDS** MOVE status to CloudApp with execution_status=WHEEL_MOVE_FINISHED Over WebSocket
11. CloudApp **RECEIVES** MOVE execution status from Buddy Over WebSocket
Postcondition
Execution_status is WHEEL_MOVE_FINISHED
1.3 Bounded Alternative Flow
RFS 1-11
1. **IF** CloudApp **RECEIVES** no heartrate data FROM SmartPhone **THEN**
2. CloudApp **SENDS** ROTATE command To Buddy with angle=50 and speed=30 Over WebSocket
3. Buddy **SETS** EnableWheels with leftWheel=1 and rightWheel=1
4. Buddy **ROTATES** at specified angle and speed.
5. Buddy **SENDS** ROTATE status to CloudApp with execution_status=WHEEL_MOVE_FINISHED Over WebSocket
6. CloudApp **RECEIVES** ROTATE execution status from Buddy Over WebSocket
7. **ENDIF**
Postcondition
Execution_status is WHEEL_MOVE_FINISHED
1.4 Specific Alternative Flow
RFS 8
1. **IF** EnableWheels is not set to TRUE **THEN**
2. Buddy **SETS** execution_status to NOK
3. Buddy **SENDS** EnableWheel status to cloudApp with execution_status=NOK Over WebSocket
4. cloudApp **RECEIVES** execution status from Buddy Over WebSocket
5. **ENDIF**
Postcondition
Execution_status is NOK

Use Case Test Model

We defined Use Case Test Model (UCTM) to guide the extraction of the information/data from UCS. The UCTM serves as a structured approach to derive test scenarios from UCS, ensuring systematic and comprehensive test generation for IoT systems. In IoT environments, where heterogeneous devices, communication protocols, and cloud services interact dynamically, the UCTM provides a systematic way to capture functional flows, alternative paths, and exception handling. By leveraging the UCTM, test cases are generated based on predefined UCS templates, incorporating essential interactions between IoT components

such as sensors, actuators, gateways, and cloud services. Applying the UCTM in IoT system testing ensures a clear mapping between functional requirements (i.e., UCSs) and test scenarios, facilitating automated test scenario generation. Figure 8.10 presents the metamodel for the UCTM.

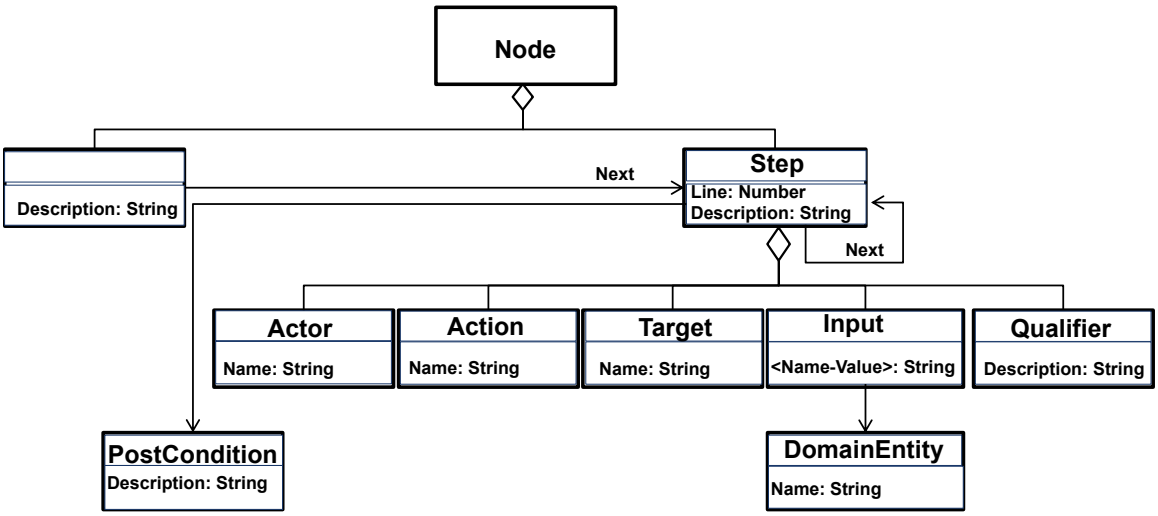


Figure 8.10: Metamodel for use case test model

Step 1: Use Case Conversion

The conversion of use case specifications from plain text into JSON format involves several key steps, leveraging the structured nature of use cases to create a well-defined JSON document. This process is critical for enhancing readability, facilitating analysis, and supporting automated processing tools. Algorithm 6 explains the process of converting UCS from plain text into JSON format. In Line 2, we create an empty JSON object. From Line 3 to Line 18, we read each line of the given UCS and append it to the corresponding section in JSON file, such as the basic flow, bounded flow, or specific flow.

Conversion begins with setting up a data structure that categorizes the various parts of the use case, such as the name, preconditions, basic flow, bounded alternatives, and specific alternatives. The main task involves reading the text document sequentially to identify and classify sections based on specific keywords or patterns indicative of different

Algorithm 6 Conversion of UCSs to JSON

Require: A text file containing use case specifications
Ensure: A JSON file representing the structured use case specifications

$\mathcal{F} \leftarrow$ Use case specification file
$\mathcal{J} \leftarrow$ Empty JSON object ▷ Initialize an empty JSON object
while not end of file $\mathcal{F}$ **do** ▷ Read until the end of file
    $line \leftarrow$ readLine($\mathcal{F}$) ▷ Read a line from the file
    if line contains "use case" **then**
        Update JSON object with use case name
    else if line contains "precondition" **then**
        Switch to preconditions section in JSON
    else if line contains "basic flow" **then**
        Switch to basic flow section in JSON
    else if line contains "bounded alternative" **then**
        Switch to bounded flow section in JSON
    else if line contains "specific alternative" **then**
        Switch to specific flow section in JSON
    else
        Update the content under current section
    end if
end while
Finalize sections and add to main JSON object
write($\mathcal{J}$) ▷ Write $\mathcal{J}$ to a file

components of a use case. This consists of recognizing headers like “Use Case Name” or “Preconditions”, which signal the start of new sections. Once a section is identified, all following text is gathered into the appropriate category until a new header is detected. This method ensures that all relevant details are grouped correctly according to their roles within the use case. The process is flexible enough to handle different types of content within each section. For example, a basic flow section might contain several steps described one after the other, whereas a preconditions section could simply list conditions without a particular order. Error handling is also an integral part of the process, ensuring the text meets expected formats and contains all necessary information. This step prevents errors and ensures the reliability of the output. After all text has been processed and assigned to the corresponding sections, the completed structured data is then prepared for output, typically serialized into a format like JSON, making it ready for use in applications such as automated testing tools or documentation systems. Listing 3 shows part of converted UCS into JSON format.

```
{
  "preconditions": [
    { "EnableWheels": "0" },
    { "Mood": "1" }
  ],
  "name": "fitbit sends data and buddy moves",
  "basic_Flow": {
    "steps": [
      {"1": "Fitbit SENDS heart data to SmartPhone with heartrate=80 bpm Over HTTP"},
      {"2": "SmartPhone RECEIVES the heartrate data From Fitbit"},
      {"3": "SmartPhone SENDS heart data with heartrate=80 bpm to Cloud over WebSocket"},
      {"4": "Cloud RECEIVES heartrate data FROM fitbit Over HTTP"},
      {"5": "Cloud CHECKS THAT heartrate is between 60 and 100 bpm"},
      {"6": "Cloud SENDS MOVE cmd to Buddy with distance=1 and speed=1 Over WebSocket"},
      {"7": "Buddy RECEIVES MOVE cmd from Cloud" },
      {"8": "Buddy SETS EnableWheels with leftWheel=1 and rightWheel=1" },
      {"9": "Buddy MOVES at specified distance and speed" },
      {"10": "Buddy SENDS MOVE status to Cloud with execution_status=... Over WebSocket"},
      {"11": "Cloud RECEIVES MOVE execution status from Buddy Over WebSocket" }
    ],
    "postconditions": [ { "Execution_status": "WHEEL_MOVE_FINISHED" } ]
  }
}
```

Listing 3: Part of Converted UCS in JSON Format

Step 2: Scenarios Extraction

The extraction of test scenarios from use case specifications plays a pivotal role in the testing process, as these scenarios are crucial for automating tests and ensuring that software behaviors align with specified requirements. This process involves several steps, mainly identifying different execution paths (i.e., basic flow, bounded flow, and specific flow) and defining post-conditions for each execution path. In the context of this chapter, each execution path indicates a scenario to be tested. Algorithm 7 details the process for converting use case specifications into test scenarios.

Algorithm 7 Extraction of Test Scenarios

Require: A JSON-formatted UCS
Ensure: A JSON-formatted test scenarios
1: $\mathcal{J} \leftarrow$ A JSON formatted UCS
2: $\mathcal{S} \leftarrow \{\}$ ▷ Initialize an empty set for scenarios
3: **for** $k \in$ {"basic_Flow", "bounded_flows", "specific_flows"} **do**
4: **if** $\mathcal{J}$.has(k) **then**
5: $\mathcal{N} \leftarrow$ New JSON object
6: content $\leftarrow$ empty list
7: postconditions $\leftarrow$ empty string
8: **for** entry in $\mathcal{J}[k]$ **do**
9: **if** entry contains "steps" **then**
10: content.append(entry["steps"])
11: **else if** entry contains "postconditions" **then**
12: postconditions $\leftarrow$ entry["postconditions"]
13: **end if**
14: **end for**
15: $\mathcal{N}$.put("steps", content)
16: $\mathcal{N}$.put("postconditions", postconditions)
17: $\mathcal{S}$.put($k, \mathcal{N}$) ▷ Add the scenario in S
18: **end if**
19: **end for**
20: return $\mathcal{S}$ ▷ Return scenarios

The process begins by initializing an empty set to store each scenario, as shown in Line 2. This ensures that all data extracted from the JSON-formatted UCS is captured in an organized manner. From Lines 3 to 14, the algorithm proceeds to iteratively examine

each predefined key (i.e., `basic_flow`, `bound_flow`, `specific_flow`) in the UCS, checking for the presence of scenario data under these keys. For each valid key found (i.e., Line 4), a new JSON object is created to represent an individual scenario (i.e., Line 5). Within this new object, the algorithm gathers a list of steps and the associated postconditions (i.e., from Lines 8 to 13). Steps are crucial as they detail the sequence of actions or events in the scenario, while postconditions define the expected state at the scenario's conclusion. Notably, preconditions are omitted unless explicitly stated, acknowledging that they are not mandatory for defining the context of every scenario. However, each scenario can rely on the global preconditions specified in the UCS. Once all pertinent data are collected for a scenario, specifically steps and postconditions, the data are stored under the respective flow type in the overarching set. This structuring is intentional to maintain clarity and ease of access to scenario information. The algorithm concludes by returning the complete set of organized scenarios.

Listing 4 shows some of the scenarios extracted from the UCS in Step 1.

```
{
 "1": {
    "postconditions": [{"Execution_status": "MOVE_FINISHED"}],
    "steps": [
      {"1": "Fitbit SENDS heart data to SmartPhone with heartrate=80 bpm Over HTTP"},
      {"2": "SmartPhone RECEIVES the heartrate data From Fitbit"},
      {"3": "SmartPhone SENDS heart data with heartrate=80 bpm to Cloud over WebSocket"},
      {"4": "Cloud RECEIVES heartrate data FROM fitbit Over HTTP"},
      {"5": "Cloud CHECKS THAT heartrate is between 60 and 100 bpm"},
      {"6": "Cloud SENDS MOVE cmd to Buddy with distance=1 and speed=1 Over WebSocket"},
      {"7": "Buddy RECEIVES MOVE cmd from Cloud" },
      {"8": "Buddy SETS EnableWheels with leftWheel=1 and rightWheel=1" },
      {"9": "Buddy MOVES at specified distance and speed" },
      {"10": "Buddy SENDS MOVE status to Cloud with execution_status=... Over WebSocket"},
      {"11": "Cloud RECEIVES MOVE execution status from Buddy Over WebSocket" }
    ]
    },
 "2": {...},
 "3": {...}
}
```

Listing 4: Scenarios from UCS

Step 3: Test Data Extraction

The next step is the extraction of test data (payload) from the test scenarios. This process involves extracting the operation to be executed, inputs, execution target, and expectations (i.e., postconditions if provided). Algorithm 7 summarizes the test data extraction process. This algorithm transforms structured scenarios into test data in the form of a payload, suit-

Algorithm 8 Test Data Generation from Scenarios

Require: $\mathcal{J} \leftarrow$ test scenarios
Ensure: $\mathcal{T} \leftarrow$ test data ▷ i.e., payloads
1: $\mathcal{T} \leftarrow \{\}$ ▷ Initialize an empty JSON object for test data
2: **for** each scenario in $\mathcal{J}$ **do**
3: $\mathcal{TD} \leftarrow \{\}$ ▷ Initialize an empty object for this test data
4: **for** each step in scenario **do**
5: operation $\leftarrow$ extractVerb$(step)$
6: inputs $\leftarrow$ extractInputs$(step)$
7: target $\leftarrow$ extractTargetInfo$(step)$
8: expectations $\leftarrow$ extractExpectations$(step)$
9: Construct test step with the operation, inputs, target, and expectations
10: Add constructed step to $\mathcal{TD}$
11: **end for**
12: Append $\mathcal{TD}$ to $\mathcal{T}$
13: **end for**
14: **return** $\mathcal{T}$

able for automated testing processes. The process is designed to ensure that each test case is both comprehensive and precisely reflective of the scenario steps.

- **Initialization of Test Data**: The process begins by creating an empty JSON object (i.e., line 1), $\mathcal{T}$, which serves as the container for all the test data generated from the scenarios. This object ensures that each piece of test data is stored and organized effectively.
- **Scenario Iteration**: For every scenario in the JSON object $\mathcal{J}$, the algorithm initializes a new, empty object $\mathcal{TD}$ (i.e., line 3), designated for storing the test data of that particular scenario. This systematic approach helps segregate the data per scenario, maintaining clarity and structure.
- **Step Processing**: Within each scenario, individual steps are processed sequentially. For

each step, the algorithm extracts key components such as the operation (verb), inputs, target information, and expectations. This extraction is explained in lines 5 to 8.

- **Operation Extraction**: The verb, indicative of the action to be performed, is extracted to define the nature of the operation in the test data. Keywords such as "SENDS" and "RECEIVES" are used to pinpoint the type of action or operation performed by entities in the scenario.
- **Inputs Extraction**: Any inputs required for the operation are gathered, ensuring that the test data reflects the prerequisites for each operation. The "WITH" keyword helps identify and extract the necessary inputs.
- **Target Information Extraction**: Details about the target of the operation, such as the protocol and target entity, are compiled to specify where and how the operation impacts the system. The "TO" keyword identifies the target entity, and "OVER" identifies the communication protocol.
- **Expectations Extraction**: Expected outcomes are identified to establish the criteria against which the operation's success or failure is evaluated. Expected outcomes are derived from scenario preconditions, and using the keyword "IS", the algorithm extracts the expected value for a successfully executed scenario.
- **Test Step Construction**: After extracting the necessary details from each step, a structured test step is constructed and added to the test data object $\mathcal{TD}$ as shown in lines 9 and 10. This aggregation ensures that all steps within a scenario are sequentially organized and encapsulated within the same test data object.
- **Compiling and Returning Test Data**: Once all steps in a given scenario are processed, the complete test data object $\mathcal{TD}$ is appended to the overarching container $\mathcal{T}$, as shown in line 12. This procedure is repeated for each scenario, culminating in a comprehensive JSON object containing structured test data for all scenarios.

The complete set of structured test data, $\mathcal{T}$, is returned, and ready for use in test generation.

Listing 5 shows part of the generated test data from our UCS. Note that in the generated test data, the expectation for each step assumes the value specified in the postcondition for the entire scenario. However, this value is only valid after executing all steps. Therefore, Human-in-the-Loop (HITL) intervention is required to provide the expectation for each step, since such detailed values are not captured by the UCSs.

```
{
  "1": {
    "TC_ID": "TBD",
    "name": "TBD",
    "steps": [
      {
        "entity": "fitbit",
        "operation": "sendHeartData",
        "inputs": {"heartrate": "80 bpm"},
        "target": {"protocol": "http","name": "smartphone"},
        "expectations": {"Execution_status": "OK"}
      },
      {
        "entity": "smartphone",
        "operation": "sendHeartData",
        "inputs": {"heartrate": "80 bpm"},
        "target": { "protocol": "websocket","name": "cloudapp"},
        "expectations": {"Execution_status": "OK" }
      },
      {
        "entity": "cloudApp",
        "operation": "sendMoveCommand",
        "inputs": {"distance": "1","speed": "1"},
        "target": {"protocol": "websocket","name": "buddy"},
        "expectations": {"Execution_status": "OK" }
      },
      {
        "entity": "buddy",
        "operation": "sendMoveStatus",
        "inputs": { "execution_status": "wheel_move_finished" },
        "target": {"protocol": "websocket","name": "cloudapp"},
        "expectations": {"Execution_status": "WHEEL_MOVE_FINISHED"}
      }
    ]
  }
}
```

Listing 5: Part of Generated Payload (i.e., Test Data)

Step 4: Test Oracle

In our approach, test oracles are manually defined by testers or developers based on their system knowledge. The process involves specifying criteria to determine whether a test case passes or fails by comparing actual outputs against the expected outcomes outlined in

the test scenario. Testers define multiple input sets along with their corresponding expected outputs, ensuring a structured validation process. These oracles serve as a reference to systematically verify system behavior and assess correctness. Since the real-time execution data is stored in the cloud, testers or engineers can access this data by writing code to fetch it from the cloud. Listing 6 shows an example of real-time execution data retrieval for the Buddy device. Similar examples for other components, such as the fitbit, smartphone, and cloudApp, exist in our repository. Listing 7 shows some code that can be used to access the real-time execution data for a specific layer of the IoT system.

Step 5: Test Case Generation

We leverage the use of generated test data (payload) and the description of the SUT to generate test cases. This process involves analyzing each payload to identify the entity or node that will execute the operation (action), operation name, outcome (i.e., expectation), and inputs. We use the description of the SUT to extract source code-related information. Algorithm 9 summarizes the test case generation process.

This algorithm generates executable test suites from a given payload (i.e., test data P) and description file of the SUT (i.e., D). It consists of the following key steps:

- **Initialization and Setup**: The algorithm begins by reading SUT description (i.e., line 1), which details the entities or components involved in the system. These entities are stored in the set M. Another set, E, is initialized to keep track of entities that are actively used in the test data, ensuring that only necessary components are set up in the test environment, which helps avoid redundant setup operations.
- **Test Class Content Setup**: Before diving into the specifics of the test cases, the algorithm prepares a foundational structure for the test class, including necessary programming imports and the basic class definition as shown in line 3. This structure is essential as it forms the scaffold on which the individual test methods will be built.
- **Entity Initialization**: Within the setup method of the test class, the algorithm iterates

```
public class BuddyDataFetcher {
 String firebaseUrl;
 public BuddyDataFetcher(String firebaseUrl) {
  this.firebaseUrl=firebaseUrl;
 }
 public Double getForwardedDistance(int myTaskID) {
  double distanceForward=0;
     FirebaseConnector fbDataFetcher=new FirebaseConnector();
        try {
            String content = fbDataFetcher.fetchData(firebaseUrl);
            JSONObject jsonObject = new JSONObject(content);
            for (String key : jsonObject.keySet()) {
                JSONObject entry = jsonObject.getJSONObject(key);
                int taskID = entry.getInt("taskID");
                int heartRate = entry.getInt("heartRate");
                JSONObject inputs = entry.getJSONObject("inputs");
                JSONObject actualResults = entry.getJSONObject("actualResults");
                String message = actualResults.getString("msg");
                double distance = inputs.getDouble("distance");
                if (taskID == myTaskID) {
                    distanceForward=distance;
                    System.out.println("Heart Rate for taskID 2: " + heartRate);

                    break;
            }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return distanceForward;
    }
 public Boolean hasMoveCompleted(int myTaskID) {
  FirebaseConnector fbDataFetcher=new FirebaseConnector();
        try {
            String content = fbDataFetcher.fetchData(firebaseUrl);
            JSONObject jsonObject = new JSONObject(content);
            for (String key : jsonObject.keySet()) {
                JSONObject entry = jsonObject.getJSONObject(key);
                int taskID = entry.getInt("taskID");
                JSONObject actualResults = entry.getJSONObject("actualResults");
                JSONObject expectedResults = entry.getJSONObject("expectations");
                String message = actualResults.getString("msg");
                String expectedMessage=expectedResults.getString("msg");
                if (taskID == myTaskID) {
                    return message.equals(expectedMessage);
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

Listing 6: Part of Buddy Real-time Execution Data Retrieval

```
public class RealTimeExecutionDataFetcher{
 public static void getExecutionData(int taskID) throws IOException {
  FitbitDataFetcher fdf=new FitbitDataFetcher("TBP");
  GatewayDataFetcher gdf=new GatewayDataFetcher("TBP");
  CloudDataFetcher cdf=new CloudDataFetcher("TBP");
  BuddyDataFetcher bdf=new BuddyDataFetcher("TBP");
        Integer fitbitHeartRate = fdf.getHeartRate(taskID);
        Integer gatewayHeartRate=gdf.getHeartRate(taskID);
        Integer cloudHeartRate=cdf.getHeartRate(taskID);
        Integer buddyForwardDistance =bdf.getForwardedDistance(taskID);
        Boolean buddyMoveCompleted=bdf.hasMoveCompleted(taskID);
}
```

Listing 7: Real-time Execution Data Instantiation for WIMP

Algorithm 9 Test Case Generation

Require: Valid set of Test Data P and SUT description D
Ensure: $T \leftarrow$ File containing executable tests

$M \leftarrow$ Extract method names from D
$E \leftarrow \emptyset$ ▷ Instantiated Classes
$T \leftarrow$ Initialize test class with imports
Begin `setUp()` method
for each $entry \in P$ **do**
    for each $step \in entry.steps$ **do**
        $e \leftarrow$ step.entity
        if $e \notin E$ **then**
            Instantiate class C_e for e
            $E \leftarrow E \cup C_e$
        end if
    end for
end for
End `setUp()` method
for each $entry \in P$ **do**
    Begin new test method
    for each $step \in entry.steps$ **do**
        Extract $operation$, $inputs$, and $expectations$ from $step$
        if $operation$ not in any method name in M **then**
            continue to next step
        end if
        Generate **assertion code** for $operation$ using $inputs$ and $expectations$
        Append **assertion code** to T
    end for
    End test method
end for
Output T

over the test data. For each step in test data, it checks if the involved entity is recognized in the SUT. If the entity is valid and not already initialized, it is added to E, and the initialization code for that entity is appended to the test class content. This initialialization is covered by lines 4 to 14 of the algorithm. This ensures that every entity needed in the test cases is properly instantiated before any test operations are performed.

- **Test Method Construction**: For each entry in test data, a new test method is created within the class (i.e., lines 15 to 26). The algorithm extracts detailed information from each step of the test data, such as the operation to be performed, the inputs required, and the expected outcomes. Using this information, it generates a specific assertion code that will effectively test whether the system's behavior aligns with the expectations outlined in the test steps. Listing 8 shows an example of an end-to-end test generated by FUNEETIS.

```
@Before
public void setUp() {
    fitbitFetcher = new fitbitDataFetcher("TBP");
    gatewayFetcher = new gatewayDataFetcher("TBP");
    cloudFetcher = new cloudDataFetcher("TBP");
    buddyFetcher = new buddyDataFetcher("TBP");
}

@Test
public void testCase1() {
    assertEquals(80, fitbitFetcher.getdHeartData(2));
    assertEquals(80, gatewayFetcher.getHeartData(2));
    assertEquals(80, cloudFetcher.getHeartData(2));
    assertEquals(1.0, buddyFetcher.getMoveDistance(2));
    assertEquals("MOVE_FINISHED", buddyFetcher.getExecutionStatus(2));
}
```

Listing 8: Part of the Generated test

Step 6: Refinement of Generated Test

Generated tests may require refinement to ensure their readiness for execution. A human-in-the-loop (HITL), such as a tester or developer with technical knowledge of SUT, plays a crucial role in this process. HITL can provide essential configuration details, such as the URL of a cloud database for runtime data collection. They can also verify that test cases include the correct parameters and expected outcomes. This refinement step ensures the

generated tests are ready for execution.

Step 7: Instrumentation of Test

Instrumentation is essential for collecting runtime execution data from the SUT. In our approach, we use the SUT description to generate code that can be injected into devices or other running components. Given the diverse technologies and programming languages in IoT systems, we leverage recent advancements in large language models (LLMs). We formulate prompts using extracted SUT information and test data to identify the appropriate methods for instrumentation. In our implementation, we integrate the OpenAI API to generate the required code for each node. To interject this code into the running system, we can use aspect-oriented programming (AOP) or request developers to manually insert the code. This instrumentation enables precise runtime data collection, enhancing the effectiveness of test execution and analysis. Algorithm 10 explains this approach. Lines 1 to 2 read the SUT description file, extract all nodes, and store them in $\mathcal{N}$. Lines 4 to 10 iterate through each node, create a prompt, and invoke the OpenAI API with that prompt. Line 9 displays the returned results, which include the instrumentation code. The generated code

Algorithm 10 Instrumentation Code Generation

Require: A valid system configuration file
Ensure: Instrumentation code for each node in the SUT
1: $\mathcal{F} \leftarrow$ SUT description file
2: $\mathcal{N} \leftarrow \emptyset$ ▷ Initialize an empty set for nodes
3: $\mathcal{N} \leftarrow$ getNodes($\mathcal{F}$) ▷ Extract all nodes (components) N from the SUT
4: **for** each $n \in \mathcal{N}$ **do** ▷ Iterate through each node in the set of nodes
5: $\mathcal{P} \leftarrow \emptyset$ ▷ Initialize an empty set for prompt elements
6: $\mathcal{P} \leftarrow$ extractSUTDescriptionElements(n) ▷ Get SUT description elements
7: $prompt \leftarrow$ ConstructPrompt($\mathcal{P}$) ▷ Construct the prompt
8: $Q \leftarrow$ InvokeOpenAIAPI($prompt$) ▷ Invoke the OpenAI API with the prompt
9: **print** Q ▷ Return the instrumentation code
10: **end for**

is subsequently refined and injected into the running node or component to collect runtime execution data. This approach effectively automates the generation of node-specific

instrumentation code, regardless of the programming language or operating system in use. Listing 9 shows an example of refined test instrumentation code that can be inserted into the Buddy to collect runtime data and send it to a centralized real-time execution database.

```
   public class FirebaseDataSender {
    private DatabaseReference mDatabase;
    public FirebaseDataSender() {
        // Set up the database reference with the specific URL
        mDatabase = FirebaseDatabase.getInstance("URL").getReference();
    }
    public void sendDataToFirebase(int test_ID, int hr, double d,double s,String action,
    String ts, String eR, String aR) {
        Map<String, Object> dataMap = new HashMap<>();
        dataMap.put("timestamp", ts);
        dataMap.put("operation", action);
        dataMap.put("heartRate", hr);
        dataMap.put("actualResults", aR);
        dataMap.put("expectations", eR);
        dataMap.put("taskID", test_ID);

        Map<String, Object> inputs = new HashMap<>();
        inputs.put("distance", d);
        inputs.put("speed", s);
        dataMap.put("inputs", inputs);
        // Send the data to Firebase under the node 'buddy.json'
        mDatabase.child("buddy.json").push().setValue(dataMap);
    }
}
```

Listing 9: Example of test instrumentation code for Buddy

Step 8: Run and Collect Data

To collect runtime execution data, our approach injects or intercepts the running application within the targeted node (i.e., component) and executes the generated instrumentation code. Once the SUT is executed, the intercepting mechanism captures the runtime execution data and transfers it to a centralized database for further analysis. Collecting runtime execution data enables the assessment of the SUT's behavior at various execution stages across the entire system, thereby achieving end-to-end testing. Figure 8.11 illustrates the workflow for capturing runtime execution data. In this scenario, *Fitbit-V2* transmits a heart rate reading of 82 bpm. The *Galaxy-10* device, acting as a gateway, forwards this data to the *CloudApp*,

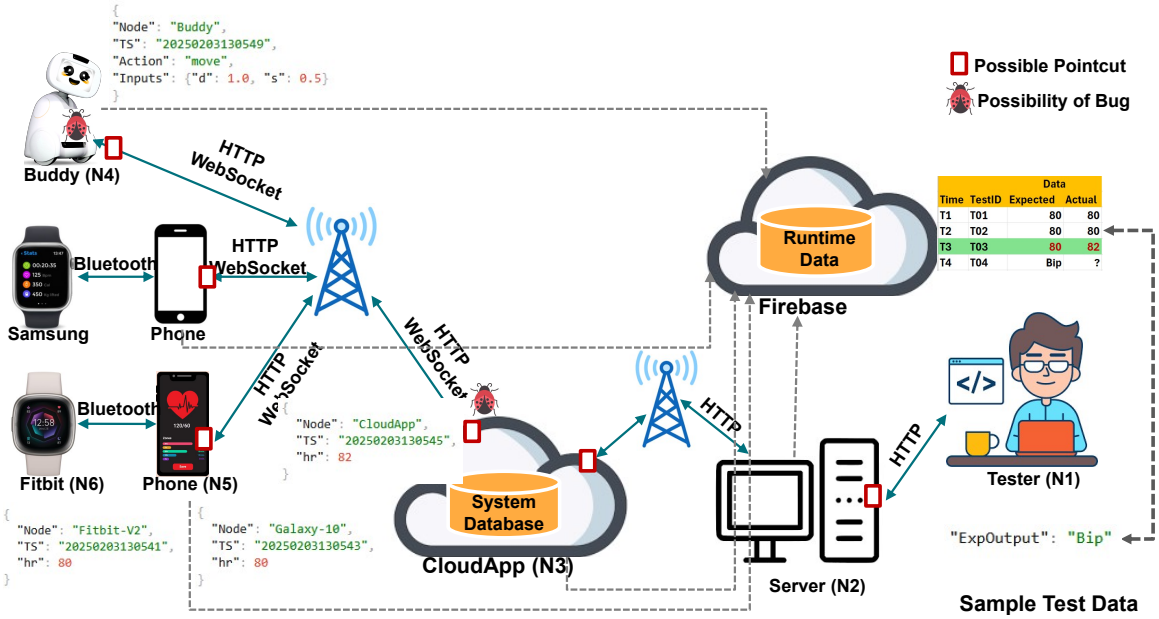


Figure 8.11: WIMP Execution Scenario

which initially receives the value unaltered. However, the *CloudApp* modifies the heart rate value from 80 to 82 bpm before sending a request to the *Buddy* robot, instructing it to move a distance of 1.0 meters at a speed of 0.5 m/s. The expected outcome of this execution is a beep sound from the *Buddy*; however, due to the modification, the *Buddy* now initiates movement instead.

Step 9: Check Execution Data

FUNEETIS collects runtime execution data by injecting instrumentation code into the SUT and intercepting its execution. The captured data is then transmitted to a cloud-based database, where it is stored in real-time. Using this live execution data, we execute our test cases that reflect the actual behavior of the system under various conditions. Assertions are dynamically defined based on this data, ensuring that the test cases validate real-world execution scenarios rather than relying solely on predefined expectations. Figure 8.12 illustrates the process.

Listing 10 shows the sample data collected from Buddy.

FUNEETIS step answers the following RQs:

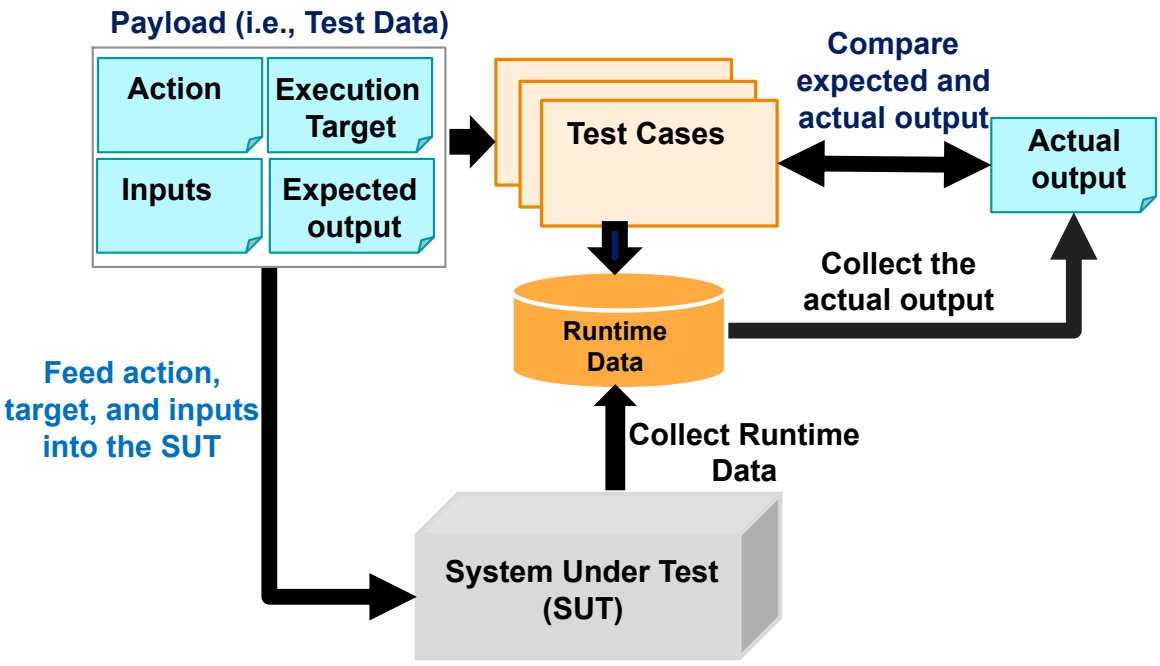


Figure 8.12: Test Execution Overview

- RQ14: How can functional end-to-end test cases be generated for IoT systems?
  - RQ14.1: How accurate is FUNEETIS in generating test scenarios and test data for SUT?
  - RQ14.2: How does FUNEETIS compare to manual testing?
  - RQ14.3: How well do the generated test achieve coverage of various test metrics?
- RQ15: How well does FUNEETIS detect bugs in IoT systems?

8.2.4 Evaluation Metrics

Hu et al. [74] proposed four coverage criteria for evaluating testing thoroughness in IoT systems: eUtility, device, compatibility, and connection coverage. These criteria were based on combinatorial testing path selection. Building on their work, we propose new coverage criteria for testing IoT systems based on FUNEETIS.

- Action Coverage $AC_{cov}(A, T)$. An action or operation refers to a specific task or function that the system performs. Action coverage is the degree to which tests exercise the actions

```
{
 "buddyExecutionData":{
    "actualResults":{
       "msg":"WHEEL_MOVE_FINISHED"
    },
    "expectations":{
       "msg":"WHEEL_MOVE_FINISHED"
    },
    "heartRate":94,
    "inputs":{
       "distance":2,
       "speed":1.5
    },
    "operation":"move",
    "target":{
       "method":"send",
       "name":"Buddy",
       "protocol":"Websocket"
    },
    "taskID":1,
    "timestamp":"2025-02-05"
 }
}
```


Listing 10: Part of sample data collected from Buddy

or behaviors triggered by commands.

$$AC_{cov}(A,T) = \frac{\text{Actions Covered by Tests (A)}}{\text{Total Actions in the UCSs (T)}} \times 100\% \tag{2}$$

- Device Coverage $DC_{cov}(D,T)$. IoT devices are physical objects embedded with sensors, software, and other technologies that enable them to connect and exchange data over the Internet. Device coverage refers to the extent to which distinct devices within IoT system are covered by tests.

$$DC_{cov}(D,T) = \frac{\text{Distinct Devices Covered by Tests (D)}}{\text{Total Distinct Devices in the SUT (T)}} \times 100\% \tag{3}$$

- Interaction Coverage $IC_{cov}(I,T)$. Interaction refers to the bidirectional or unidirectional exchange of data, commands, or signals between devices, application layers, or other system components. Interaction coverage is the degree to which the data exchange and communication flows between interconnected IoT components (devices, cloud services,

mobile apps, robots, etc.) are covered by tests.

$$IC_{cov}(I,T) = \frac{\text{Interactions Covered by Tests (I)}}{\text{Total Interactions in the SUT (T)}} \times 100\% \tag{4}$$

- Protocol Coverage $Pr_{cov}(Pr,T)$. A protocol in IoT systems is a set of rules that governs how devices and components communicate and exchange data. Examples include MQTT for lightweight communication, HTTP for web-based interactions, WebSocket for real-time communication, and Bluetooth for short-range connectivity. Protocol Coverage in IoT systems refers to the extent to which all communication protocols used within the IoT system (such as WebSocket, HTTP, COAP, MQTT, etc.) have been included in tests.

$$Pr_{cov}(Pr,T) = \frac{\text{Protocols Covered by Tests (Pr)}}{\text{Total Protocols in the SUT (T)}} \times 100\% \tag{5}$$

- Scenario Coverage $S_{cov}(S,T)$. Test scenarios are sets of possible paths derived from use case specifications that describe different ways a system behaves. They define specific conditions, inputs, actions, and expected results that need to be tested to verify that the system behaves correctly according to requirements. Scenario coverage in the context of IoT systems refers to the extent to which all scenarios explicitly defined in the use case specification (e.g., basic flow, alternative flow, specific flow) are covered by tests.

$$SC_{cov}(S,T) = \frac{\text{Scenarios covered by Tests (S)}}{\text{Explicit Scenarios in the SUT (T)}} \times 100\% \tag{6}$$

8.2.5 Tests Prioritization

Based on the above coverage criteria, we construct the following prioritization function that calculates the prioritization score P_S of each test following Equation 7

$$P_S = \sum_{i=1}^{N} W_i \times C_i \tag{7}$$

Where:

$W_i =$ Weight of Each Coverage Criterion.

$C_i \ =$ Coverage for each criterion.

The main purpose of test prioritization is to ensure that the high-priority tests are executed first. We use Algorithm 11 to find the top-priority tests.

Algorithm 11 Prioritization of Tests

function TestsSelection(Tests T)
    Input
        T_{All} All Tests
        W Weight of Coverage Metrics
    Output
        T_s Set of Selected Tests
    Let $T_{All} \leftarrow T$, $T_s \leftarrow \emptyset$
    For Each t in T_{All} **do**
        $P_S = \sum_{i=1}^{N} W_i \times C_i$ ▷ Prioritization Score
        if $P_S > TargetScore$ **then**
            $T_s \leftarrow t$ ▷ Add test (t) to Selected tests T_s
        end if
    end for
    return T_s
end function

Weightage Assignment. Weightage for each criterion (W_i) is determined based on the specific priorities of the project or system under evaluation. For simplicity, in this chapter, we assume equal weights:

$$W_i = \frac{1}{n} \tag{8}$$

Where:

$W_i =$ Weight of Metric i.

$n \quad =$ Total number of Metrics.

8.3 Results

To evaluate the performance of LLM-based and custom approaches, we conducted experiments on WIMP. We establish ground truth (GT) based on a manual experiment conducted by experienced testers and test cases generated using our TGenAI approach.

In summary this chapter answer the following research questions (RQs):

- RQ12: How do test generation approaches compare for E2E testing of IoT Systems?
- RQ13: How reliable are LLMs in E2E test generation for IoT systems?
- RQ14: How can functional end-to-end test cases be generated for IoT systems?
  - RQ14.1: How accurate is FUNEETIS in generating test scenarios and test data for SUT?
  - RQ14.2: How does FUNEETIS compare to manual testing?
  - RQ14.3: How well do the generated test achieve coverage of various test metrics?
- RQ15: How well does FUNEETIS detect bugs in IoT systems?

To answer these questions, we conducted experiments and evaluated FUNEETIS on an IoT system.

8.3.1 RQ12: How do test generation approaches compare for E2E testing of IoT Systems?

To evaluate the different approaches, we used one simple use case from the WIMP system consisting of five scenarios. Using this specification, we tested all four approaches and compared their outputs with the expected results.

Scenario Coverage

Scenario coverage measures how well each approach identifies test scenarios from use cases by comparing generated tests to all valid scenarios. The results in Table 8.4 indicate that CA, SSLA, and HA achieved full scenario coverage, with 100% precision and recall. MSLA, however, failed to correctly identify one scenario, leading to a 95% coverage rate, with corresponding precision and recall reductions.

Preliminary results suggest the potential of CA and LLM-based approaches in generating test scenarios, motivating deeper exploration.

Table 8.4: Scenarios Generation

	Scenarios Coverage		Accuracy	
Approach	**Valid**	**Invalid**	**Precision**	**Recall**
CA	100%	0%	100%	100%
SSLA	100%	0%	100%	100%
MSLA	95%	5%	95%	95%
HA	100%	0%	100%	100%

Correctness of Generated Tests

The correctness of the generated tests assesses whether the execution steps are identified correctly for each scenario. Table 8.5 shows that CA and HA achieved perfect accuracy with 100% precision and recall. SSLA performed well with 94% coverage and high validity (99%), while MSLA had minor inaccuracies (96% precision, 95% recall).

Table 8.5: Correctness of Generated Tests

	Test Steps			Accuracy	
Approach	**Coverage**	**Valid**	**Invalid**	**Precision**	**Recall**
CA	100%	100%	0%	100%	100%
SSLA	94%	99%	1%	98%	97%
MSLA	100%	96%	4%	96%	95%
HA	100%	100%	0%	100%	100%

Correctness of Each Execution Step

We further examined the accuracy of individual execution steps by analyzing operation correctness, protocol adherence, method validity, and field-level accuracy. The summarized results of this analysis are presented in Table 8.6. While all approaches performed well in terms of general operation correctness, node identification, and expectations, notable variations were observed in input generation accuracy. Specifically, CA exhibited reduced accuracy in input generation (87%), whereas MSLA showed a lower expectation of correctness (91%). Despite performing well overall, HA had minor inaccuracies in input

correctness (96%). In contrast, SSLA demonstrated perfect accuracy across all evaluated aspects.

Table 8.6: Correctness of Each Execution Step in Generated Test

Approach	Operation	Protocol	Method	Node	Input	Expectation
CA	100%	100%	100%	100%	87%	100%
SSLA	100%	100%	100%	100%	100%	100%
MSLA	100%	100%	100%	100%	100%	91%
HA	100%	100%	100%	100%	96%	100%

Takeaway for RQ12

While both CA and SSLA show promising results in scenario identification and execution step accuracy, a more comprehensive investigation is required to explore their full capabilities.

8.3.2 RQ13: How reliable are LLMs in E2E test generation for IoT systems?

To evaluate TGenAI (shown in Figure 8.7), we constructed a ground truth (GT) using five postgraduate students in software engineering (three PhDs and two Masters) with experience in software testing. The participants received training on the WIMP system to ensure familiarity with its functionalities. Each participant manually generated test cases (TCs) for three predefined use cases while we recorded the time taken for completion. The generated test cases were assessed for correctness by executing them on the system under test (SUT) and evaluated for completeness to determine whether all possible scenarios from the use case specifications were covered. We decompose this RQ into the following subquestions:

- RQ13.1: To what extent can TGenAI generate the correct tests for the E2E testing of IoT systems?
- RQ13.2: How does the TGenAI approach compare to manual test case generation?

• RQ13.3: How do LLM APIs compare to GUIs in generating test cases?

To answer RQ13.1 and RQ13.2, we executed TGenAI using multiple models (i.e., based on [193]) and different iterations. To answer RQ13.3, we used the ChatGPT web interface to run our prompts instead of using the API in our tool to generate tests. For instance, in our first run with **GPT-4o-mini**, we prompted it to "Generate tests from the use case specification <we provided a UCS> following this template <we provided a template>", which generated six test cases, five of which were correct. In the second run, we prompted, "Some scenarios are missing. Please cover all scenarios", and it generated six test cases covering all scenarios. In the third interaction, we asked, "Can you find more tests?", resulting in eight test cases; however, two of them did not correspond to any scenario in the given UCS. This pattern was observed across other models and different UCSs. Our findings suggest that using the web interface can yield better results than API calls.

RQ13.1: To what extent can TGenAI generate the correct tests for the E2E testing of IoT systems?

We ran TGenAI using multiple models and different iterations. Table 8.7 shows the results generated by TGenAI. We selected the top models recommended by the previous study

Table 8.7: TGenAI Results Using Different Models

		UCS-1 (# of Test Cases)			UCS-2 (# of Test Cases)			UCS-3 (# of Test Cases)			Execution Time (ms)		
Model	**Run**	**Total**	**Correct**	**Incorrect**	**Total**	**Correct**	**Incorrect**	**Total**	**Correct**	**Incorrect**	**UCS 1**	**UCS 2**	**UCS 3**
GPT-4o-mini	Run 1	6	4	2	8	6	2	6	4	2	23802	22410	19246
	Run 2	5	4	1	7	5	2	7	4	3	17466	14325	30764
	Run 3	6	5	1	8	7	1	6	5	1	10370	15318	10767
GPT-4o	Run 1	6	5	1	7	6	1	6	4	2	77460	83358	54886
	Run 2	6	5	1	7	6	1	6	5	1	67780	51065	95935
	Run 3	7	6	1	7	7	0	6	6	0	20486	71705	52503
GPT-4 Turbo	Run 1	5	4	1	3	2	1	6	3	3	123716	121391	90852
	Run 2	5	3	2	6	4	2	5	3	2	109247	91102	132745
	Run 3	6	5	1	7	6	1	6	5	1	106397	114513	81547
GPT-3.5 Turbo	Run 1	6	5	1	6	3	3	1	1	0	146411	143125	99684
	Run 2	10	5	5	8	1	7	8	4	4	157864	115623	172702
	Run 3	7	4	3	9	2	7	9	5	4	121284	135850	141159

[192] for test automation. We ran TGenAI three times, evaluating the generated tests after each run. Table 8.7 summarizes the results, including the total number of generated tests.

We define correctness based on whether the generated tests contain valid steps, even if they require manual intervention via Human-in-the-Loop (HITL). An incorrect test either lacks steps or does not correspond to any specific scenario. The expected number of correct tests for UCS1, UCS2, and UCS3 are 6, 7, and 6, respectively. Any generated number exceeding these values indicates extraneous or incorrect tests. We analyzed our results and found the following:

- **GPT-4o** performed well, achieving the expected number of correct tests for UCS2 and UCS3 in the third run. However, slight inconsistencies appeared, such as generating an extra test in UCS1 during the third run.
- **GPT-4o-mini** demonstrated some inconsistencies. In Run 2, it did not build upon the previous run for UCS1, producing only 5 correct tests instead of 6. Similarly, for UCS2, it dropped from 8 correct tests in Run 1 to 7 in Run 2 and further back to 6 in Run 3. Additionally, for UCS2, in both Run 1 and Run 3, it generated more tests than expected, indicating the presence of incorrect tests.
- **GPT-4 Turbo** exhibited variability in correctness, generating incorrect tests and missing scenarios across runs. For example, in the second run of UCS3, it produced two incorrect tests, while in the first and third runs, it did not reach the expected number of correct tests.
- **GPT-3.5 Turbo** was the most inconsistent. It sometimes exceeded the expected number of correct tests, such as generating 10 tests for UCS1 in Run 2 by varying input values within the same scenario. However, it frequently generated incorrect tests, particularly for UCS2 and UCS3, with up to seven incorrect tests in some runs.

Takeaway for RQ13.1

While we observed many inconsistencies across different models, the results indicate that LLMs can still generate some valid tests for IoT systems based on use case specifications. However, Human-in-the-Loop (HITL) intervention remains essential to filter out incorrect tests, refine the generated tests, and ensure completeness before execution.

RQ13.2: How does the TGenAI approach compare to manual test case generation?

We compared the manual approach with TGenAI based on three factors: test generation time, scenario coverage, and input coverage. Table 8.8 presents the time spent by engineers versus the time taken by each model to generate tests.

Table 8.8: Test Generation Time (in seconds)

	TGenAI			Engineer		
Model	**UCS-1**	**UCS-2**	**UCS-3**	**UCS-1**	**UCS-2**	**UCS-3**
GPT-4o-mini	17	17	20	3921	2374	1580
GPT-4o	55	69	68	3921	2374	1580
GPT-4 Turbo	113	109	102	3921	2374	1580
GPT-3.5 Turbo	142	132	138	3921	2374	1580

TGenAI significantly reduced test generation time compared to manual efforts. GPT-4o-mini completed tasks in as little as 20 seconds, while GPT-3.5 Turbo took up to 142 seconds. Table 8.9 compares the maximum number of tests generated by TGenAI and engineers.

Table 8.9: Tests Generated

	TGenAI			Engineer		
Model	**UCS-1**	**UCS-2**	**UCS-3**	**UCS-1**	**UCS-2**	**UCS-3**
GPT-4o-mini	5	8	6	6	5	5
GPT-4o	6	7	6	6	5	5
GPT-4 Turbo	5	6	5	6	5	5
GPT-3.5 Turbo	10	9	19	6	5	5

In some cases, engineers created more tests than TGenAI, but TGenAI occasionally generated more tests, such as in UCS-2. GPT-3.5 Turbo showed some inconsistencies in UCS-1. For input coverage, both manual and AI-generated tests had limitations. Engineers did not explore different input ranges (e.g., lower and upper boundaries), whereas TGenAI occasionally considered multiple inputs for a single scenario. For instance, GPT-4o-mini generated 8 tests from 7 possible scenarios in UCS-2, while GPT-3.5 Turbo generated 10 tests from 6 scenarios in UCS-1 and up to 19 tests in UCS-3.

Takeaway for RQ13.2

TGenAI significantly reduced test generation time and often sometimes produced more tests than manual efforts, sometimes less. However, both approaches missed some scenarios, and LLMs occasionally made errors by generating incorrect tests.

RQ13.3: How do LLM APIs compare to GUIs in generating test cases?

As the API results are already presented in Table 8.7 , Table 8.10 shows the results obtained from each LLM using the web interface across multiple iterations.

Table 8.10: Model Performance Evaluation of Web Interface vs. API Call

		UCS-1 (# of Test Cases)			UCS-2 (# of Test Cases)			UCS-3 (# of Test Cases)		
Model	**Run**	**Total**	**Correct**	**Incorrect**	**Total**	**Correct**	**incorrect**	**Total**	**Correct**	**Incorrect**
GPT-4o-mini	Run 1	6	5	1	7	5	2	6	4	2
	Run 2	6	6	0	7	5	2	7	6	1
	Run 3	8	6	2	9	7	2	8	6	2
GPT-4o	Run 1	6	5	1	5	5	0	5	5	0
	Run 2	6	6	0	7	7	0	6	6	0
	Run 3	8	6	2	8	7	1	8	6	2
GPT-4 Turbo	Run 1	4	2	2	5	3	2	5	1	4
	Run 2	7	5	2	6	4	2	6	4	2
	Run 3	7	5	2	9	5	4	6	5	1

We compared the ChatGPT web interface and API call based on four key aspects:

- **Consistency in Improving Previous Results:** When using the web interface, the models

were able to refine their responses based on previous interactions. For example, GPT-4o-mini initially generated 5 tests for UCS1, then improved to 6 in the second run, and finally produced 8 by including variations in inputs. In contrast, API calls lacked memory across runs, leading to inconsistent improvements. For instance, GPT-4o-mini generated 6 tests in the first API run for UCS1 but regressed to 5 in the second run before returning to 6 in the third.

- **Improvement in Correctness from Previous Results:** The web interface demonstrated the ability to refine responses based on iterative feedback, leading to a higher number of correct test cases. For example, in UCS2, GPT-4o-mini through the web interface improved from 5 correct tests in Run 1 to 7 in Run 3. However, in the API results, the same model fluctuated between 5 and 7 correct tests across runs, showing less reliability in progressively improving correctness.
- **Extracting the Correct Tests:** The web interface facilitated iterative prompting, ensuring that missing scenarios were progressively covered. This was evident in UCS2 and UCS3, where test extraction improved with follow-up prompts. The API, however, often generated fluctuating results without maintaining correctness across runs. In particular, GPT-3.5 Turbo struggled significantly in extracting correct tests via API, producing large variations across runs.
- **The Number of Invalid Tests:** The web interface showed a reduced number of incorrect tests over iterations, demonstrating adaptation to refining output. For instance, GPT-4 Turbo reduced incorrect test generation in UCS2 from 4 in Run 1 to 1 in Run 3 when using the web interface. However, in the API, incorrect test generation remained unstable, as seen in GPT-3.5 Turbo, which produced up to 7 incorrect tests in UCS2 across different runs.

Our findings indicate that the web interface provides a more adaptive and refined approach to test generation due to its context retention, which allows for iterative improvements. In

contrast, the API struggles with consistency, often failing to build upon previous results, leading to incorrect and incorrect test case generation. While the API is essential for seamless integration into automation pipelines, addressing consistency and context retention issues could enhance its effectiveness in test generation.

Takeaway for RQ13.3

Our findings indicate that the web interface provides a more adaptive and refined approach to test generation due to its context retention, which allows for iterative improvements. In contrast, the API struggles with consistency, often failing to build upon previous results, leading to incorrect and unstable test case generation. This is likely due to additional system instructions embedded in the web interface, which are not publicly available for API usage. Optimizing the prompts in the API requests may help reduce this gap, but further research is needed to find the reasons behind such differences between the quality of the response generated by the web interface and the response of the API call.

8.3.3 RQ14: How can functional end-to-end test cases be generated for IoT systems?

RQ14.1: Test Scenario and Test Data Generation Accuracy

To evaluate the accuracy of FUNEETIS in generating test scenarios, and test data, and test cases, we conducted an empirical evaluation on WIMP involving three use case specifications (UCS-1, UCS-2, UCS-3). The evaluation considers three key aspects: scenario extraction accuracy, test data generation accuracy, and test case generation accuracy. Table 8.11 shows our experimental results. We assess accuracy using precision, defined as the ratio of correctly generated artifacts to the total generated artifacts, and recall, which measures the ratio of correctly generated artifacts to the total expected artifacts. The results indicate that the scenario extraction accuracy is 100% for UCS-1, UCS-2, and UCS-3. This

Table 8.11: Scenario and Test Data Generation Accuracy

	Scenarios Extraction		Test Data Extraction		Test Case Generation	
UCS	**Precision (%)**	**Recall (%)**	**Precision (%)**	**Recall (%)**	**Precision (%)**	**Recall (%)**
UCS-1	100	100	94	92	85	80
UCS-2	100	100	96	94	82	78
UCS-3	100	100	95	93	83	79

suggests that FUNEETIS effectively extracts relevant execution scenarios from the use case specifications. Similarly, test data generation accuracy is slightly lower but remains strong, ranging from 94% to 96%, ensuring that extracted scenarios are well-mapped to representative test data. However, test case generation accuracy is comparatively lower, with values not exceeding 85%. This is due to the necessity of human-in-the-loop (HITL) intervention, where manual adjustments are required to refine the generated test cases for correctness and completeness. The need for HITL highlights areas where automated generation can be improved but also reflects the complexity of translating high-level UCSs into executable test cases.

Takeaway for RQ14.1

FUNEETIS shows high accuracy in scenario extraction and test data generation, but test case generation requires some manual intervention, highlighting areas for improvement in test automation.

RQ14.2: Comparison of FUNEETIS and Manual Testing

We compared the manual approach and the proposed approach by evaluating two variables: the time required to generate tests and the test scenarios covered in generated tests. Table 8.12 presents a comparison of the time spent by engineers versus both manual testing effort and the TGenAI approach. We observed that FUNEETIS significantly reduced the time required to generate tests compared to both engineers and our previous LLM-Based Approach. We did not compared the efforts required with test case generation since in both

Table 8.12: Time Required to Create Test Data (i.e., Payload)

	Time (secs)		
Approach	**UCS-1**	**UCS-2**	**UCS-3**
Manual	3922	2375	1580
TGENAI	55	69	68
FUNEETIS	14	15	12

manual testing or our previous approach, we did not include the step to convert the test data (i.e., payload) into executable test cases.

We also evaluated FUNEETIS on its capability to generate scenarios and test data compared to the manual effort and our previous approach. Table 8.13 shows our findings.

Table 8.13: Test Scenario Generated

	Test Scenario Coverage		
Approach	**UCS-1**	**UCS-2**	**UCS-3**
Manual	6	7	5
TGENAI	6	7	6
FUNEETIS	6	7	6

The results show that both FUNEETIS and TGENAI (using ChatGPT-4o) achieved the maximum possible test scenario coverage, confirming their effectiveness in generating comprehensive test scenarios. Specifically, the highest number of scenarios possible was 6 for UCS-1, 7 for UCS-2, and 6 for UCS-3, all of which were fully covered by both approaches. In contrast, manual efforts fell slightly short for UCS-3.

Takeaway for RQ14.2

FUNEETIS significantly reduced test data generation time while maintaining maximum test scenario coverage, making it the most efficient approach.

RQ14.3: Metric Coverage in Generated Tests

We assessed the accuracy of FUNEETIS on generated tests. Table 8.14 shows our findings. The results highlight the effectiveness of FUNEETIS across various IoT-specific metrics,

Table 8.14: Metrics Coverage

	Nodes		Protocols		Scenarios		Interactions		Actions	
UCS	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
UCS-1	100%	100%	100%	100%	100%	100%	78%	78%	89%	73%
UCS-2	100%	100%	100%	100%	100%	100%	86%	75%	82%	69%
UCS-3	100%	100%	100%	100%	100%	100%	78%	70%	85%	69%

with precision and recall providing key insights into its accuracy.

- **Node and Protocol Coverage:** FUNEETIS achieved 100% precision and recall across all use case specifications (UCSs), accurately identifying all nodes and protocols without missing any elements.
- **Scenario Coverage:** FUNEETIS maintained strong performance, achieving 100% precision and recall for UCS-1, UCS-2, and UCS-3.
- **Interaction Coverage:** FUNEETIS demonstrated reliable precision across UCSs (78%-86%), but recall varied slightly, ranging from 70%-78%, showing some limitations in capturing all interactions.
- **Action Coverage:** Precision remained high (82%-89%), while recall showed some variations (69%-73%), indicating that some actions were partially captured.

Takeaway for RQ14.3

FUNEETIS achieved 100% accuracy in node, protocol, and scenario coverage. However, interaction and action coverage remain low without manual interventions.

8.3.4 RQ15: How can generated tests be executed to detect bugs in IoT systems?

To assess the effectiveness of FUNEETIS in detecting bugs across different IoT layers, we conducted a series of test runs, starting with 10 tests and progressively increasing in increments of 10 up to 60 cumulative test executions. Figure 8.13 shows the bugs detected based on our SUT. In each iteration, we recorded the number of bugs detected in four

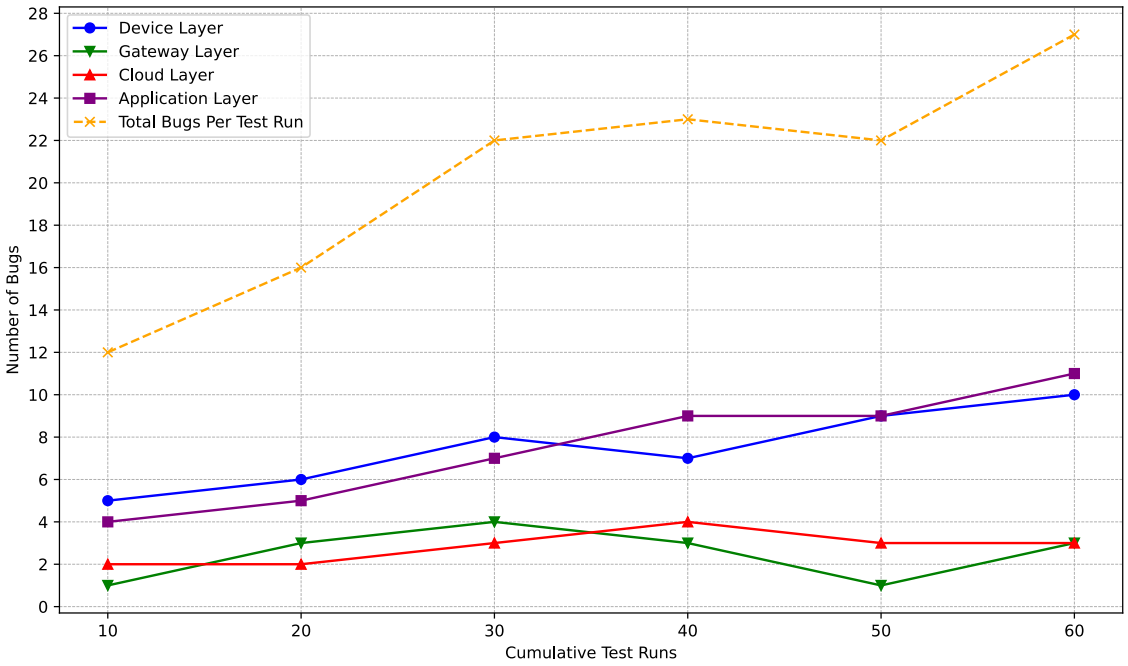


Figure 8.13: Bug Detection Results

layers: Device Layer, Gateway Layer, Cloud Layer, and Application Layer. Additionally, we computed the total number of bugs per test run to evaluate the cumulative detection capability of FUNEETIS. The results indicate that the number of detected bugs varied across layers. The Device Layer consistently exhibited the highest number of detected bugs, ranging from 5 to 10 per iteration. The Application Layer also showed a progressive increase in detected bugs, reaching a peak of 11 bugs in the final iteration. The Gateway and Cloud Layers, in contrast, had lower bug counts with minor variations. A key observation

from the data is that as the number of test runs increased, the total number of detected bugs also grew progressively, confirming that FUNEETIS is effective in uncovering bugs across multiple layers of the IoT system. Without end-to-end testing, testers focusing solely on the application layer might have detected fewer bugs than when following FUNEETIS.

The distribution of bugs across the IoT system nodes reveals that issues are not isolated to a single component but span the entire system architecture. At the **device level**, bugs are further categorized into Fitbit, phone, and buddy robot. Over six cumulative test runs, **Fitbit bugs remained low**, ranging from 1 to 2, with a total of 9 bugs detected. This suggests that Fitbit, having limited interaction points or simpler functionality, encounters fewer issues. **Phone bugs showed a steady increase**, starting from 1 in the first run and reaching 3 by the last two runs, with a total of 13 bugs, indicating increased functional complexity. **Buddy Robot consistently recorded the highest number of device-level bugs**, starting with 3 and rising to 5 by the final run, totaling 23 bugs, likely due to its integration and behavioral complexity. Figure 8.14 shows the bugs we identified for each device.

Takeaway for RQ15

Experimental results show that FUNEETIS effectively detects bugs across IoT layers, with the highest defects in Device and Application Layers. The lower bug count in Gateway and Cloud Layers may be due to less processing or computation in these layers. Without end-to-end testing, testers focusing solely on the application layer might have detected fewer bugs than with FUNEETIS.

8.4 Discussions

This section discusses our results and implications for both researchers and practitioners.

- **Limitations of LLMs in generating tests from UCS.** We identify limitations of LLMs in processing natural text inputs, including inconsistencies caused by lack of context and

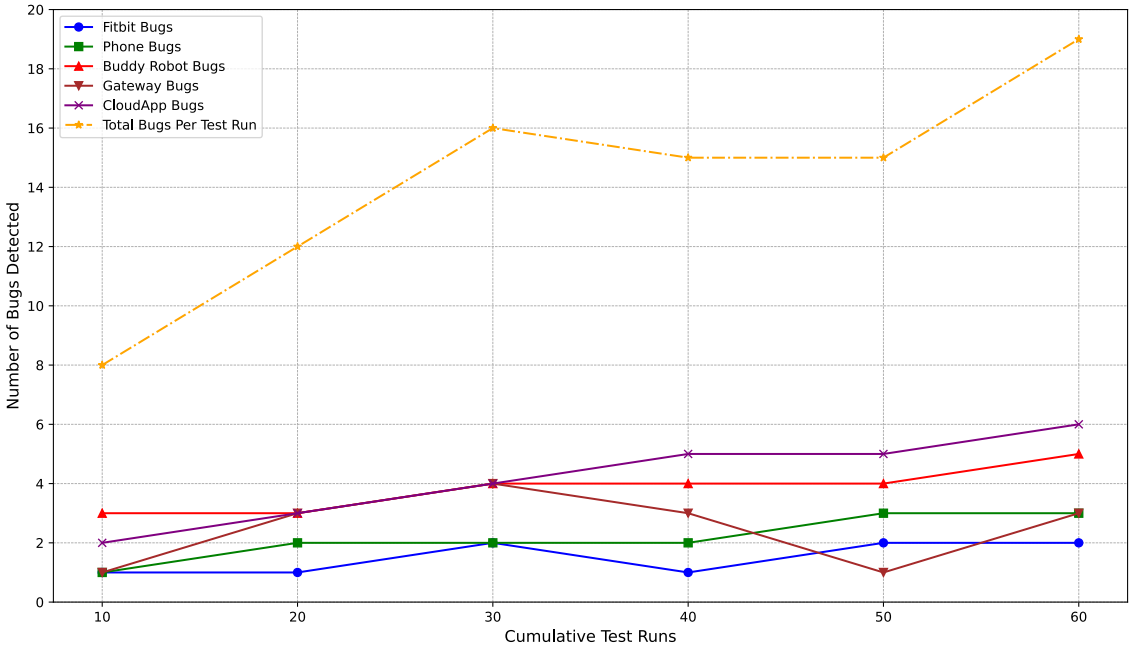


Figure 8.14: Bug Detection for each Component

difficulty handling complex use case descriptions, while observing strong performance when source code is provided. LLMs can perform well in test generation when the input is the source code. However, when input is use case specifications, they often require significant manual intervention and still may not guarantee reliable results.

- **Dependency on Well-Structured UCS.** For FUNEETIS to generate the required test data (i.e., payload) and test scenarios, requirements must be strictly written following the proposed patterns and recommended keywords for UCS. Failure to comply with these guidelines may result in incorrect and incomplete test data generation.
- **Using Test Data to Generate Tests:** FUNEETIS relies on test data (i.e., payload) and applications that access real-time runtime execution data to generate executable test cases. The final test cases are executed on an application that reads the actual runtime execution data of the System Under Test (SUT), as shown in Figure 8.12. Without this approach, executing tests across all nodes (i.e., components) of an IoT system would be complex and impractical.

- **Generating and Injecting Code in SUT for Test Instrumentation:** To collect real-world runtime execution data, FUNEETIS generates instrumentation code, which is injected into the SUT. This can be achieved through techniques such as Aspect-Oriented Programming (AOP) and code injection or by requiring developers to manually add the code at identified pointcuts based on the SUT description. In our experiment, we manually injected the instrumentation code, as AOP was not supported in all nodes due to varying coding styles, with some not following Object-Oriented Programming.
- **Implications for Researchers and Practitioners:** For researchers, our work provides a foundation for further exploration into improving test generation and execution for end-to-end testing of IoT systems. Our results serve as a proof-of-concept (PoC) and could be extended to enhance generalizability while reducing the involvement of human in the loop (HITL). For practitioners, FUNEETIS offers a structured approach to conducting end-to-end functional testing, ensuring that all IoT system layers are considered rather than focusing only on specific ones.
- **Using LLM for Code Instrumentation:** Leveraging Large Language Models (LLMs) for automated code instrumentation could enhance the efficiency of test execution by reducing manual intervention. We leveraged the use of LLMs to generate the instrumentation code for various IoT implementations.

8.5 Threats to Validity

This chapter has several threats to validity. The first threat is that our evaluation is based on a single IoT system with limited use case scenarios due to the lack of open-source IoT systems for testing. While this system includes key IoT aspects such as devices, protocols, interactions, and operations, it may not fully represent all IoT variations. Experimenting with multiple systems would enhance generalizability, but finding suitable open systems remains a challenge. Another threat is the dependency on well-structured UCSs rather than plain text. Although this requires additional effort in writing specifications, it

is a necessary trade-off given the complexity of IoT systems. Finally, dependencies on Human-in-the-Loop (HITL) techniques present another challenge. While some level of human intervention is necessary due to the complexity of IoT systems, minimizing this involvement remains a key goal for future work. Our approach aims to reduce manual effort while maintaining accuracy and completeness in test generation and execution. For LLM-based approach, the choice of the Large Language Models (LLMs) also introduces a potential threat to validity. Our results are based solely on the LLMs selected in this study (e.g., GPT-4o and GPT-3.5-turbo), and the use of alternative models may lead to different outcomes. Similarly, the choice of **SAMMO** [164] as the prompt optimization tool is another threat. Other optimization tools or strategies might yield different results. We also acknowledge that we only used the default values for temperature and top_p when interacting with the ChatGPT API. This may have constrained the diversity or determinism of the generated responses. Adjusting these parameters could potentially yield different results, affecting the reproducibility and generalizability of our findings.

8.6 Conclusion

This chapter explored different techniques for generating functional end-to-end (E2E) tests for IoT systems. Our initial investigation compared both a structured custom approach and LLM-based techniques, revealing promising results for both. This led to a deeper exploration of LLM-based test generation through TGenAI. While LLMs demonstrated the potential to automate test generation, they also introduced inconsistencies and required significant Human-in-the-Loop (HITL) intervention to ensure correctness and completeness. These limitations motivated a shift in focus toward a more structured and deterministic custom approach, culminating in the development of FUNEETIS. FUNEETIS systematically derives test scenarios, test data, and executable test cases from structured use case specifications (UCSs). By extending the Restricted Use Case Modeling (RUCM) approach with IoT-specific keywords, FUNEETIS ensures that critical system elements such as device

interactions, protocols, and actions are explicitly captured, enabling automated test generation. Additionally, this chapter proposed a structured representation of IoT systems to facilitate instrumentation code generation. To support automated test execution, FUNEETIS includes four key algorithms: (1) scenario extraction from UCSs, (2) test data generation, (3) test case generation, and (4) test instrumentation to collect runtime execution data. While some HITL involvement remains necessary, FUNEETIS marks a significant step toward full automation of functional end-to-end IoT system testing. Our evaluation demonstrated that FUNEETIS significantly reduces test generation time compared to manual approaches while ensuring full scenario coverage. Additionally, we introduced five IoT-specific test metrics to assess the thoroughness of generated tests, confirming that FUNEETIS provides comprehensive node, protocol, and scenario coverage. Empirical evaluation on the WIMP system further showed that FUNEETIS enhances bug detection across IoT layers, particularly in the Device and Application Layers. Future work will focus on further automation to minimize manual intervention, improving test execution efficiency and extending validation across additional IoT systems.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

This thesis proposed an approach for functional end-to-end (E2E) testing of IoT systems, addressing the challenges associated with automating test generation and execution for IoT systems. By systematically investigating IoT testing practices in academia and industry, we identified key gaps and proposed approaches that bridge these gaps. In particular, we focus on the following gaps: the lack of a testing guide tailored for testing IoT systems, the need for test generation and execution approaches for end-to-end testing, and the absence of metrics to assess the effectiveness of generated tests for functional end-to-end test automation in IoT systems.

Our contributions include the development of an IoT testing taxonomy, a software engineering framework for IoT system testing, and practical approaches for generating and executing IoT system tests. To realise these contributions, we followed three main steps: (1) problem identification, (2) taxonomy and framework development, and (3) functional E2E testing approaches.

Problem Identification. We began by analyzing the state of IoT system testing through a systematic literature review (SLR) and an industry study. The SLR examined 83 primary studies, identifying testing objectives, approaches, tools, and challenges. It compiled 47 quality attributes, including five specific to IoT, and analyzed 19 IoT testing tools and 15 testbeds, highlighting gaps in automation and standardization. To complement these findings, the industry study surveyed 49 IoT practitioners, conducted interviews with 9 experts, and analyzed four years of Eclipse IoT survey data. This provided critical insights into real-world IoT testing challenges, including the lack of standardized testing frameworks, heavy reliance on manual testing, and the need for automated test case generation.

Developing a Taxonomy and Framework for IoT Testing. To address the gap of the need for a testing guide, we introduced an **IoT-specific testing taxonomy** that categorizes seven key aspects of IoT testing: objectives, tools, testers, stages, environments, objects under test, and approaches. The taxonomy was validated through feedback from over 200 practitioners and tested in two case studies, demonstrating its practical utility in guiding testers. Building on this, we developed **TISSEA**, a framework for testing the technical software engineering aspects of IoT systems. This framework maps critical testing aspects with test granularities to test targets, inputs artifacts, test orchestration strategies, and execution method. Our evaluation, conducted with 22 professionals and two case studies, confirmed that TISSEA can comprehensively guide software engineers to test technical SE aspects of IoT systems at the device and application layers. However, additional research is needed to refine orchestration strategies for the gateway and cloud layers.

Developing Approaches for Functional End-to-End IoT System Testing. We explored two complementary approaches: (1) testing IoT devices in isolation and (2) end-to-end functional testing. For **isolated IoT device testing**, we applied both traditional source code analysis and Large Language Models (LLMs) for test generation. While these approaches

successfully produced executable tests, hardware constraints posed challenges for executing tests on real IoT devices. Although this part was not included in the main body of this report, the findings are presented in our published papers [124, 119].

For **end-to-end testing**, we initially explored four approaches for generating tests from specifications, including LLM-based, custom rule-based, and hybrid approaches. These approaches were selected because they address the gap in generating and executing end-to-end tests for IoT systems, particularly in scenarios where the source code of the entire system or detailed behavioral models may not be available. Based on a comparative analysis of preliminary results to assess their effectiveness, we selected two promising approaches for further investigation: (1) test generation using LLMs (TGenAI) and (2) custom rule-based test generation (FUNEETIS). These approaches demonstrate strong potential in bridging the gap in both the generation and execution of end-to-end tests for IoT systems.

Test Generation Using LLMs (TGenAI): This approach used LLMs to generate system-level tests from use case specifications, leveraging few-shot prompt engineering and optimization through SAMMO. While it reduced manual effort, Human-in-the-Loop (HITL) intervention was still required to refine the generated tests.

End-to-End Testing with FUNEETIS: FUNEETIS systematically generates test scenarios, test data, and executable test cases from structured UCSs. It extends the *Restricted Use Case Modeling (RUCM)* approach with IoT-specific keywords, ensuring comprehensive test coverage across IoT system layers. Additionally, we proposed five *IoT-specific test metrics* to assess the effectiveness of test generation and execution. To support automated execution, FUNEETIS incorporates four key algorithms for *scenario extraction, test data generation, test case generation, and runtime instrumentation*. Our evaluation on the WIMP system demonstrated that FUNEETIS significantly reduces test generation time, enhances bug detection, and ensures comprehensive test coverage across IoT layers.

On limitations and lessons learned, this research journey uncovered several insights

and areas where improvements could be made. One notable limitation is the use of Large Language Models (LLMs) for generating tests directly from use case specifications. While LLMs offer promising capabilities, our findings suggest that use case specifications can be challenging for LLMs to understand, which may lead them to miss the full context and overlook certain scenarios or misinterpret the intended behavior. In contrast, LLMs tend to perform better when source code is available, as it provides more detailed and concrete information. Nevertheless, hybrid approaches appear promising. For example, in FUNEETIS, after extracting structured test data such as scenarios, actions, input data, target information, and constraints using a rule-based method, LLMs could potentially be used more effectively to translate this data into executable tests. This integration was not explored exhaustively in this work and remains a direction for future research.

Another limitation stems from the early focus of the research. During the early phase of this research, a significant amount of time was dedicated to exploring test generation from source code for end-to-end testing. This path proved challenging for several reasons. Most notably, source code was often unavailable for many IoT devices, and orchestrating distributed testing across a heterogeneous ecosystem of devices, platforms, and technologies proved to be extremely complex. Each component required a dedicated test agent or executor tailored to its specific runtime environment, programming language, and communication protocol. Triggering tests, collecting results proved highly difficult and impractical in my research. These complexities led me to conclude that this approach was impractical at the time and focused on middleware-based testing, where real-time execution data from each component could be captured and leveraged to drive test execution.

In parallel, significant effort was devoted to developing a technical guide through a taxonomy and framework aimed at guiding the testers. While this was a valuable step, feedback from some practitioners suggested that what matters most in the industry is having practical, usable tools that support test execution effectively. In retrospect, greater effort

could have been invested in maturing FUNEETIS and making it more robust and industry-ready.

Despite these limitations, the outcomes of this research lay a solid foundation for advancing automated end-to-end testing in IoT systems. FUNEETIS, though not yet fully mature, is a meaningful step toward solving the broader challenge of end-to-end functional testing in IoT systems. It may be a small step, but it contributes to a broader and ongoing effort in tackling one of the field's most pressing challenges.

To conclude, this research demonstrates that functional end-to-end test automation for IoT systems is feasible. We, therefore, affirmatively answer our problem statement: It is indeed possible to automatically generate and execute tests for functional end-to-end testing of IoT systems in order to detect bugs.

Final Remarks

This thesis demonstrated that functional end-to-end test automation for IoT systems is feasible. Recalling our problem statement, "Given these complexities, is it possible to automate functional end-to-end test generation and systematically execute the generated tests to detect and localize bugs in IoT systems?", we can conclude that "it is possible to automatically generate tests for functional end-to-end testing of IoT systems and execute the generated tests to detect bugs".

9.2 Future Work

Building on the findings of this research, several avenues for future work can be explored to further enhance the automation and applicability of the proposed approach.

In the short term, my immediate focus will be on refining and expanding the automation capabilities of FUNEETIS to improve its efficiency and scalability. This includes reducing human intervention in test case refinement, optimizing test data generation, and enhancing instrumentation techniques. Furthermore, I want to validate this approach on a wider set

of IoT systems to assess its generalizability across different architectures, devices, and communication protocols. Additionally, I will assess the possibility of delegating some steps of FUNEETIS to Large Language Models (LLMs), which could further streamline and simplify the automation process.

In the mid-term, my focus will be on establishing a standardized framework for Use Case Specifications (UCS) tailored specifically for IoT systems. Another critical focus will be on enhancing the proof-of-concept tool developed in this research to make it industry-ready. This involves refining its usability, integrating it with commonly used IoT development environments, and ensuring that practitioners can seamlessly adopt it in real-world IoT system testing. By making the tool more accessible and robust, the approach can be effectively integrated into industrial IoT system testing workflows, benefiting both researchers and practitioners. Furthermore, I will also be looking for industry partners with systems ready to test, to further assess and validate the capability of FUNEETIS in practical settings.

In the long term, I will extend FUNEETIS beyond the functional aspect to explore the automation of other technical software engineering aspects of IoT systems, such as security, scalability, performance, and interoperability testing. I plan to collaborate with PhD students interested in IoT systems testing to fully turn our approach into an industry-ready product.

By advancing these research directions, this research aims to contribute towards a more robust and fully automated approach for end-to-end testing of IoT systems.

Chapter 10

Publications

- A Multi-Method Study of Internet of Things Systems Testing in Industry [125].
- A Systematic Review: Objectives, Approaches, Tools, and Challenges [126].
- Taxonomy For IoT Systems Testing: Practical Guidance for Practitioners [123].
- Towards an automated approach for testing IoT devices [124].
- A Systematic Literature Review of IoT System Architectural Styles and Their Quality Requirements [139]
- An Exploratory Study on Code Quality, Testing, Data Accuracy, and Practical Use Cases of IoT Wearables [119].
- Analysis of Microservices-Based IoT Systems: Deployment Challenges, Industry Practices, and Performance Insights [199].
- IoT Systems Testing: Taxonomy, Empirical Findings, and Recommendations [120].
- TGenAI: LLM-based Approach for Functional Test Cases Generation for IoT System [129].
- Test Generation from Use Case Specifications for IoT Systems: Custom, LLM-Based, and Hybrid Approaches [28].
- TISSEA: A Framework for Testing IoT Systems Based on Technical Software Engineering Aspects [122].
- FUNEETIS: Approach for Functional End-to-End Testing of IoT Systems [121].

- A Systematic Literature Review of Machine Learning Approaches for Migrating Monolithic Systems to Microservices [181].
- Exploring the Impacts of Antipatterns on Object-Oriented, Service-Oriented, and Mobile-Oriented Systems [113].

Bibliography

[1] What makes a good qualitative research question? https://www.bl.uk/business-and-ip-centre/articles/what-makes-a-good-qualitative-research-question. [Accessed 02-Jun-2023].

[2] ISO/IEC 25010. Software Product Quality. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Accessed 16-Feb-2023].

[3] ISO/IEC/IEEE 29119-1:2022. Software and Systems Engineering - Software Testing. https://www.iso.org/standard/81291.html, 2022. [Accessed 10-Feb-2023].

[4] Saqlain Abbas, Mehwish Naz, Zaeem Anwaar, Muhammad Usman Farooq, and Farasat Ullah Khan. Availability Testing of IoT-Based Health Care Devices: A Survey. In *2021 International Conference on Innovative Computing (ICIC)*, pages 1–6. IEEE, 2021.

[5] Saqlain Abbas, Mehwish Naz, Zaeem Anwaar, Muhammad Usman Farooq, and Farasat Ullah Khan. Availability Testing of IoT-Based Health Care Devices: A Survey. In *2021 International Conference on Innovative Computing (ICIC)*, pages 1–6, 2021.

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

[7] Bestoun S Ahmed, Miroslav Bures, Karel Frajtak, and Tomas Cerny. Aspects of Quality in Internet of Things (IoT) Solutions: A Systematic Mapping Study. *IEEE Access*, 7:13758–13780, 2019.

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

[9] Rohit Akhilesh, Oliver Bills, Naveen Chilamkurti, and Mohammad Jabed Morshed Chowdhury. Automated penetration testing framework for smart-home-based IoT devices. *Future Internet*, 14(10):276, 2022.

[10] Hesham Alaqail and Shakeel Ahmed. Overview of software testing standard ISO/IEC/IEEE 29119. *International Journal of Computer Science and Network Security (IJCSNS)*, 18(2):112–116, 2018.

[11] Dharun Anandayuvaraj and James C Davis. Reflecting on Recurring Failures in IoT Development. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.

[12] Nahid Anwar and Susmita Kar. Review paper on various software testing techniques & strategies. *Global Journal of Computer Science and Technology*, 19(2):43–49, 2019.

[13] Shadi Attarha and Anna Förster. AssureSense: A Framework for Enabling Sensor Fault Detection in Low-Power IoT Edge Devices. *IEEE Sensors Journal*, 2024.

[14] Abdullah Ayub Khan, Asif Ali Laghari, Zaffar Ahmed Shaikh, Zdzislawa Dacko-Pikiewicz, and Sebastian Kot. Internet of Things (IoT) Security With Blockchain Technology: A State-of-the-Art Review. *IEEE Access*, 10:122679–122695, 2022.

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

[16] Ilja Behnke, Lauritz Thamsen, and Odej Kao. Héctor: A framework for testing IoT applications across heterogeneous edge and cloud testbeds. In *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing companion*, pages 15–20. Association for Computing Machinery, 2019.

[17] Ilja Behnke, Lauritz Thamsen, and Odej Kao. Héctor: A framework for testing IoT applications across heterogeneous edge and cloud testbeds. In *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing companion*, pages 15–20. Association for Computing Machinery, 2019.

[18] Jossekin Beilharz, Philipp Wiesner, Arne Boockmeyer, Lukas Pirl, Dirk Friedenberger, Florian Brokhausen, Ilja Behnke, Andreas Polze, and Lauritz Thamsen. Continuously Testing Distributed IoT Systems: An Overview Of The State Of The Art. In *Service-Oriented Computing–ICSOC 2021 Workshops: AIOps, STRAPS, AI-PA and Satellite Events, Dubai, United Arab Emirates, November 22–25, 2021, Proceedings*, pages 336–350. Springer, 2022.

[19] Khalil Ben Kalboussi, Farah Barika Ktata, and Ikram Amous. A simulation framework for IoT networks intrusion and penetration testing. In *International Conference on Digital Technologies and Applications*, pages 252–263. Springer, 2023.

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

[21] Mohamed Boukhlif, Nassim Kharmoum, and Mohamed Hanine. LLMs for Intelligent Software Testing: A Comparative Study. In *Proceedings of the 7th International Conference on Networking, Intelligent Systems and Security*, pages 1–8, 2024.

[22] Miroslav Bures, Bestoun S. Ahmed, Vaclav Rechtberger, Matej Klima, Michal Trnka, Miroslav Jaros, Xavier Bellekens, Dani Almog, and Pavel Herout. PatrIoT: IoT Automated Interoperability and Integration Testing Framework. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 454–459, 2021.

[23] Miroslav Bures, Bestoun S Ahmed, Vaclav Rechtberger, Matej Klima, Michal Trnka, Miroslav Jaros, Xavier Bellekens, Dani Almog, and Pavel Herout. PatrIoT: IoT automated interoperability and integration testing framework. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 454–459. IEEE, 2021.

[24] Miroslav Bures, Bestoun S Ahmed, Vaclav Rechtberger, Matej Klima, Michal Trnka, Miroslav Jaros, Xavier Bellekens, Dani Almog, and Pavel Herout. Patriot: IoT automated interoperability and integration testing framework. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 454–459. IEEE, 2021.

[25] Miroslav Bures, Tomas Cerny, and Bestoun S Ahmed. Internet of things: Current challenges in the quality assurance and testing methods. In *International conference on information science and applications*, pages 625–634. Springer, 2018.

[26] Burhan, Muhammad and Rehman, Rana Asif and Khan, Bilal and Kim, Byung-Seo. IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey. *Sensors*, 18(9), 2018.

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

[28] Zacharie Chenail-Larcher, Jean Baptiste Minani, and Naouel Moha. Test Generation from Use Case Specifications for IoT Systems: Custom, LLM-Based, and Hybrid Approaches. In *Proceedings of the 18th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Naples, Italy, 2025. IEEE.

[29] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. Internet of Things (IoT): Research, Simulators, and Testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 2018.

[30] Arina Cheverda, Ahror Jabborov, Artem Kruglov, and Giancarlo Succi. State-of-the-Art Review of Taxonomies for Quality Assessment of Intelligent Software Systems. In *2022 3rd International Informatics and Software Engineering Conference (IISEC)*, pages 1–6, 2022.

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

[32] Jacob Cohen. Weighted Kappa: Nominal Scale Agreement Provision For Scaled Disagreement Or Partial Credit. *Psychological bulletin*, 70(4):213, 1968.

[33] Alison Cooke, Debbie Smith, and Andrew Booth. Beyond PICO: the SPIDER Tool For Qualitative Evidence Synthesis. *Qualitative health research*, 22(10):1435–1443, 2012.

[34] Alison Cooke, Debbie Smith, and Andrew Booth. Beyond PICO: the SPIDER tool for qualitative evidence synthesis. *Qualitative health research*, 22(10):1435–1443, 2012.

[35] Riccardo Coppola and Emil Alégroth. A Taxonomy of Metrics for GUI-based Testing Research: A Systematic Literature Review. *Information and Software Technology*, page 107, 2022.

[36] Mariela Cortés, Raphael Saraiva, Marcia Souza, Patricia Mello, and Pamella Soares. Adoption of Software Testing in Internet of Things: A Systematic Literature Mapping. In *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing*, pages 3–11, 2019.

[37] Victor Costa, Gustavo Girardon, Maicon Bernardino, Rodrigo Machado, Guilherme Legramante, Anibal Neto, Fábio Paulo Basso, and Elder de Macedo Rodrigues. Taxonomy of Performance Testing Tools: A Systematic Literature Review. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1997–2004, 2020.

[38] Rareş Cristea, Mihail Feraru, and Ciprian Paduraru. Building blocks for IoT testing: a benchmark of IoT apps and a functional testing framework. In *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*, pages 25–32, 2022.

[39] Hugo Diogo Queirós Cunha. Low-Code Solution for IoT Testing. *ThinkMind*, 2019.

[40] Jeanderson Cândido, Maurício Aniche, and Arie Deursen. Log-based software monitoring: a systematic mapping study. *PeerJ Computer Science*, 7:e489, 05 2021.

[41] João Pedro Dias, Flávio Couto, Ana CR Paiva, and Hugo Sereno Ferreira. A Brief Overview of Existing Tools for Testing the Internet-of-Things. In *2018 IEEE international conference on software testing, verification and validation workshops (ICSTW)*, pages 104–109. IEEE, 2018.

[42] João Dias, Flavio Couto, Ana Paiva, and Hugo Ferreira. A brief overview of existing tools for testing the internet-of-things. In *2018 IEEE international conference on software testing, verification and validation workshops (ICSTW)*, pages 104–109. IEEE, 2018.

[43] Riya Dutta, Diego Elias Costa, Emad Shihab, and Tanja Tajmel. Diversity Awareness in Software Engineering Participant Research. *arXiv preprint arXiv:2302.00042*, 2023.

[44] Tore Dyba, Torgeir Dingsoyr, and Geir K Hanssen. Applying Systematic Reviews To Diverse Study Types: An Experience Report. In *First international symposium on empirical software engineering and measurement (ESEM 2007)*, pages 225–234. IEEE, 2007.

[45] Maialen Eceiza, Jose Luis Flores, and Mikel Iturbe. Fuzzing the Internet of Things: A Review on the Techniques and Challenges for Efficient Vulnerability Discovery in Embedded Systems. *IEEE Internet of Things Journal*, 8(13):10390–10411, 2021.

[46] Nasir U Eisty, George K Thiruvathukal, and Jeffrey C Carver. A survey of software metric use in research software development. In *2018 IEEE 14th international conference on e-Science (e-Science)*, pages 212–222. IEEE, 2018.

[47] Jawaher Abdulwahab Fadhil and Qusay Idrees Sarhan. A Survey on Internet of Things (IoT) Testing. In *2022 International Conference on Computer Science and Software Engineering (CSASE)*, pages 77–83. IEEE, 2022.

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

[49] Mahdi Fahmideh, Aakash Ahmad, Ali Behnaz, John Grundy, and Willy Susilo. Software Engineering For Internet of Things: The Practitioners' Perspective. *IEEE Transactions on Software Engineering*, 48(8):2857–2878, 2021.

[50] Mahdi Fahmideh, Aakash Ahmad, Ali Behnaz, John Grundy, and Willy Susilo. Software engineering for internet of things: The practitioners' perspective. *IEEE Transactions on Software Engineering*, 48(8):2857–2878, 2022.

[51] Wen Fei, Hiroyuki Ohno, and Srinivas Sampalli. A Systematic Review of IoT Security: Research Potential, Challenges and Future Directions. *ACM Computing Surveys*, 2023.

[52] Michael Felderer, Philipp Zech, Ruth Breu, Matthias Büchler, and Alexander Pretschner. Model-based Security Testing: A Taxonomy and Systematic Classification. *Software Testing, Verification and Reliability*, 26(2):119–148, 2016.

[53] Rangga Firdaus, Nina Kurnia Hikmawati, Yusuf Durachman, Herlino Nanang, Dewi Khairani, and Muhammad Syauqi Hazimi. Usability Testing Analysis of a Company Website in Indonesia. In *2022 Seventh International Conference on Informatics and Computing (ICIC)*, pages 1–6. IEEE, 2022.

[54] Donald Firesmith. A Taxonomy of Testing. Carnegie Mellon University, Software Engineering Institute's Insights (blog), Aug 2015. Accessed: 2024-Jan-23.

[55] Donald Firesmith. A Taxonomy of Testing. Carnegie Mellon University, Software Engineering Institute's Insights (blog), Aug 2015. Accessed: 2023-May-2.

[56] Donald G Firesmith. *Common system and software testing pitfalls: how to prevent and mitigate them: descriptions, symptoms, consequences, causes, and recommendations*. Addison-Wesley Professional, 2014.

[57] Fischbach, Jannik and Frattini, Julian and Vogelsang, Andreas and Mendez, Daniel and Unterkalmsteiner, Michael and Wehrle, Andreas and Henao, Pablo Restrepo and Yousefi,

Parisa and Juricic, Tedi and Radduenz, Jeannette and others. Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. *Journal of Systems and Software*, 197:111549, 2023.

[58] Giancarlo Fortino, Claudio Savaglio, Giandomenico Spezzano, and MengChu Zhou. Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):223–236, 2020.

[59] Lavínia Freitas and Valéria Lelli. Using Machine Learning on Testing IoT Applications: A Systematic Mapping. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*, pages 348–358, 2022.

[60] Bernhard Garn, Dominik-Philip Schreiber, Dimitris E Simos, Rick Kuhn, Jeff Voas, and Raghu Kacker. Combinatorial methods for testing Internet of Things smart home systems. *Software Testing, Verification and Reliability*, 32(2):e1805, 2022.

[61] Vahid Garousi, Michael Felderer, Çağrı Murat Karapıçak, and Uğur Yılmaz. Testing embedded software: A survey of the literature. *Information and Software Technology*, 104:14–45, 2018.

[62] Vahid Garousi, Michael Felderer, Marco Kuhrmann, and Kadir Herkiloğlu. What industry wants from academia in software testing? Hearing practitioners' opinions. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 65–69, 2017.

[63] Ahmad Nauman Ghazi, Kai Petersen, Sri Sai Vijay Raj Reddy, and Harini Nekkanti. Survey research in software engineering: Problems and mitigation strategies. *IEEE Access*, 7:24703–24718, 2018.

[64] Anna Katrina Gomez and SimiKamini Bajaj. Challenges of testing complex Internet of Things (IoT) devices and systems. In *11th international conference on knowledge and systems engineering (KSE)*, pages 1–4. IEEE, Institute of Electrical and Electronics Engineers Inc., Oct 2019.

[65] Claudia Greco, Giancarlo Fortino, Bruno Crispo, and Kim-Kwang Raymond Choo. AI-enabled IoT penetration testing: state-of-the-art and research challenges. *Enterprise Information Systems*, 17(9), 2023.

[66] Ligia Georgeta Guşeilă, Dragoş-Vasile Bratu, and Sorin-Aurel Moraru. Continuous testing in the development of IoT applications. In H Hacid, M Aldwairi, MR Bouadjenek, M Petrocchi, N Faci, F Outay, A Beheshti, L Thamsen, and H Dong, editors, *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, volume 13236 of *Lecture Notes in Computer Science*, pages 1–6. IEEE, 2019. 19th International Conference on Service-Oriented Computing (ICSOC), ELECTR NETWORK, NOV 22-25, 2021.

[67] Ligia Georgeta Guşeilă, Dragoş-Vasile Bratu, and Sorin-Aurel Moraru. Continuous testing in the development of iot applications. In *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, pages 1–6. IEEE, 2019.

[68] Jon D Hagar. Software test architectures and advanced support environments for IoT. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 252–256. IEEE, 2018. testing approaches for embedded systems can be extended to IoT systems.

[69] Jon Duncan Hagar. *IoT System Testing*. Springer, Hot Sulphur Springs, USA, 2022.

[70] Jon Duncan Hagar. *IoT System Testing*. Apress, 2022.

[71] Muhammad Hassan, Sallar Ahmadi-Pour, Khushboo Qayyum, Chandan Kumar Jha, and Rolf Drechsler. LLM-Guided Formal Verification Coupled with Mutation Testing. In *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–2, 2024.

[72] Zozo Hassan, Hesham Ali, and Mahmoud Badawy. Internet of Things (IoT): Definitions, Challenges, and Recent Research Directions. *International Journal of Computer Applications*, 128:975–8887, 10 2015.

[73] Ralph Heredia. 4 Layers of IoT Architecture. https://www.zipitwireless.com/blog/4-layers-of-iot-architecture-explained. [Accessed 16-Feb-2023].

[74] Linghuan Hu, W Eric Wong, D Richard Kuhn, Raghu N Kacker, and Shuo Li. CT-IoT: a combinatorial testing-based path selection framework for effective IoT testing. *Empirical Software Engineering*, 27:1–38, 2022.

[75] IEEE. IEEE Guide for Software Verification and Validation Plans. *IEEE Std 1059-1993*, pages 1–87, 1994.

[76] IEEE. ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, pages 1–541, 2017.

[77] IEEE. IEEE Standard for an Architectural Framework for the Internet of Things (IoT). *IEEE Std 2413-2019*, pages 1–269, 2020.

[78] Eclipse IoT. IOT and Edge Key findings 2019. https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2019.pdf. Accessed: 2022-11-10.

[79] Eclipse IoT. IOT and Edge Key findings 2020. https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2020.pdf. Accessed: 2022-11-10.

[80] Eclipse IoT. IoT and Edge Key findings 2021. https://outreach.eclipse.foundation/iot-edge-developer-2021. Accessed: 2022-11-10.

[81] Eclipse IoT. IoT and Edge Key findings 2022. https://outreach.eclipse.foundation/iot-edge-developer-survey-2022. Accessed: 2022-11-10.

[82] ISO. ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 1:General concepts. *ISO/IEC/IEEE 29119-1:2022(E)*, pages 1–60, 2022.

[83] KS Jasmine. Non-functional Testing-Based Framework for Developing Reliable IoT Products. In *Next Generation of Internet of Things: Proceedings of ICNGIoT 2021*, pages 81–90. Springer, 2021.

[84] Minani Jean Baptiste, Yahia El Fellah, Sanam Ahmed, Fatima Sabir, Naouel Moha, and Yann-Gaël Guéhéneuc. An Exploratory Study on Code Quality, Testing, Data Accuracy, and Practical Use Cases of IoT Wearables. In *7th Conference on Cloud and Internet of Things (CIoT)*, pages 1–5. IEEE, 2024.

[85] Minani Jean Baptiste, Fatima Sabir, Naouel Moha, and Yann-Gaël Guéhéneuc. A Systematic Review of IoT Systems Testing: Objectives, Approaches, Tools, and Challenges. *IEEE Transactions on Software Engineering*, pages 1–29, 2024.

[86] Mohan Krishna Kagita, Giridhar Reddy Bojja, and Mohammed Kaosar. A framework for intelligent IoT firmware compliance testing. *Internet of Things and Cyber-Physical Systems*, 1:1–7, 2021.

[87] Teeba Kh and Ibrahim Hamarash. Model-based quality assessment of Internet of Things software applications: A systematic mapping study. *International Journal of Interactive Mobile Technologies*, 2020.

[88] Teeba Ismail Kh. and Ibrahim Ismael Hamarash. Model-Based Quality Assessment of Internet of Things Software Applications: A Systematic Mapping Study. *Int. J. Interact. Mob. Technol.*, 14:128–152, 2020.

[89] Nour Khezemi, Jean Baptiste Minani, Fatima Sabir, Naouel Moha, Yann-Gaël Guéhéneuc, and Ghizlane El Boussaidi. A Systematic Literature Review of IoT System Architectural Styles and Their Quality Requirements. *IEEE Internet of Things Journal*, 11(23):37599–37616, 2024.

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

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

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

[93] Shubha Chaturvedi Kiran Bhagnani, Gautam Chaturvedi. Taxonomy of Testing Techniques. *International Journal of Engineering and Computer Science*, 3(10), Oct. 2014.

[94] Barbara Ann Kitchenham, Lech Madeyski, and David Budgen. SEGRESS: Software Engineering Guidelines For Reporting Secondary Studies. *IEEE Transactions on Software Engineering*, 2022.

[95] Klima, Matej and Bures, Miroslav and Ahmed, Bestoun S and Bellekens, Xavier and Atkinson, Robert and Tachtatzis, Christos and Herout, Pavel. Specialized path-based technique to test Internet of Things system functionality under limited network connectivity. *Internet of Things*, 22:100706, 2023.

[96] Chorng-Shiuh Koong, Chihhsiong Shih, Pao-Ann Hsiung, Hung-Jui Lai, Chih-Hung Chang, William C Chu, Nien-Lin Hsueh, and Chao-Tung Yang. Automatic testing environment for multi-core embedded software—ATEMES. *Journal of systems and software*, 85(1):43–60, 2012.

[97] Kuldeep Kuldeep. https://artoftesting.com/test-artifacts-deliverables.

[98] Dennis Kundisch, Jan Muntermann, Anna Maria Oberländer, Daniel Rau, Maximilian Röglinger, Thorsten Schoormann, and Daniel Szopinski. An update for taxonomy designers: methodological guidance from information systems research. *Business & Information Systems Engineering*, pages 1–19, 2021.

[99] Barbara H Kwasnik. The role of classification in knowledge representation and discovery. *Library Trends*, 1999.

[100] P. Ladisa, H. Plate, M. Martinez, and O. Barais. Taxonomy of Attacks on Open-Source Software Supply Chains. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1509–1526. IEEE Computer Society, May 2023.

[101] Xabier Larrucea, Annie Combelles, John Favaro, and Kunal Taneja. Software Engineering For The Internet Of Things. *IEEE Software*, 34(1):24–28, 2017.

[102] Seungjin Lee, Jaeeun Park, Hyungwoo Choi, and Hyeontaek Oh. Energy-Efficient AP Selection Using Intelligent Access Point System to Increase the Lifespan of IoT Devices. *Sensors*, 23(11), 2023.

[103] Maurizio Leotta, Davide Ancona, Luca Franceschini, Dario Olianas, Marina Ribaudo, and Filippo Ricca. Towards a runtime verification approach for internet of things systems. In *Current Trends in Web Engineering: ICWE 2018 International Workshops, MATWEP, EnWot, KD-WEB, WEOD, TourismKG, Cáceres, Spain, June 5, 2018, Revised Selected Papers 18*, pages 83–96. Springer International Publishing, 2018.

[104] Maurizio Leotta, Diego Clerissi, Dario Olianas, Filippo Ricca, Davide Ancona, Giorgio Delzanno, Luca Franceschini, and Marina Ribaudo. An acceptance testing approach for Internet of Things systems. *IET Software*, 12(5):430–436, 2018.

[105] Maurizio Leotta, Filippo Ricca, Diego Clerissi, Davide Ancona, Giorgio Delzanno, Marina Ribaudo, and Luca Franceschini. Towards an Acceptance Testing Approach for IoT Systems. In *ICWE Workshops*, pages 125–138. Springer, 2017.

[106] Maurizio Leotta, Filippo Ricca, Diego Clerissi, Davide Ancona, Giorgio Delzanno, Marina Ribaudo, and Luca Franceschini. Towards an acceptance testing approach for Internet of Things systems. In *Current Trends in Web Engineering*, pages 125–138. Springer, 2018.

[107] Hareton KN Leung and Peter WL Wong. A Study of User Acceptance Tests. *Software Quality Journal*, 6:137–149, 1997.

[108] Jia Li, Behrad Moeini, Shiva Nejati, Mehrdad Sabetzadeh, and Michael McCallen. A Lean Simulation Framework for Stress Testing IoT Cloud Systems. *IEEE Transactions on Software Engineering*, 2024.

[109] Francesca Lonetti, Antonia Bertolino, and Felicita Di Giandomenico. Model-Based Security Testing in IoT Systems: A Rapid Review. *Information and Software Technology*, 164:107326, 2023.

[110] Francesca Lonetti, Antonia Bertolino, and Felicita Di Giandomenico. Model-based security testing in IoT systems: A Rapid Review. *Information and Software Technology*, page 107326, 2023.

[111] Eleanor E Maccoby and Nathan Maccoby. The interview: A tool of social science. *Handbook of social psychology*, 1(1):449–487, 1954.

[112] QA Madness. What is a test plan and how to write one?, Oct 2021.

[113] K. Mahmood, Jean Baptiste Minani, G. Rasool, F. Sabir, Y.-G. Guéhéneuc, and F. Jaffar. Exploring the Impacts of Antipatterns on Object-Oriented, Service-Oriented, and Mobile-Oriented Systems. *Software: Practice and Experience*, 2025. Under revision.

[114] Amir Makhshari and Ali Mesbah. IoT Bugs and Development Challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 460–472, 2021.

[115] Amir Makhshari and Ali Mesbah. IoT Bugs and Development Challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 460–472, 2021.

[116] Shona McCombes. Writing Strong Research Questions. `https://www.scribbr.com/research-process/research-questions/`, 2022. [Accessed 02-Jun-2023].

[117] Noha Medhat, Sherin Moussa, Nagwa Badr, and Mohamed F. Tolba. Testing techniques in iot-based systems. In *2019 Ninth International Conference on Intelligent Computing and Information Systems*, pages 394–401, 2019.

[118] Noha Medhat, Sherin M Moussa, Nagwa Lotfy Badr, and Mohamed F Tolba. A framework for continuous regression and integration testing in iot systems based on deep learning and search-based techniques. *IEEE Access*, 8:215716–215726, 2020.

[119] Jean Baptiste Minani, Yahia El Fellah, Ahmed Sanam, Fatima Sabir, and Naouels Moha. An Exploratory Study on Code Quality, Testing, Data Accuracy, and Practical Use Cases of IoT Wearables. In *7th edition of Conference on Cloud and Internet of Things 2024 (CIoT'24)*, 2024.

[120] Jean Baptiste Minani, Yahia El Fellah, Fatima Sabir, Naouel Moha, Yann-Gaël Guéhéneuc, Martin Kuradusenge, and Tomoaki Masuda. IoT systems testing: Taxonomy, empirical findings, and recommendations. *Journal of Systems and Software*, page 112408, 2025.

[121] Jean Baptiste Minani, Naouel Moha, Yann-Gaël Guéhéneuc, and Fatima Sabir. FUNEETIS: Approach for Functional End-to-End Testing of IoT Systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2025.

[122] Jean Baptiste Minani, Naouel Moha, Yann-Gaël Guéhéneuc, and Fatima Sabir. TISSEA: A Framework for Testing IoT Systems Based on Technical Software Engineering Aspects. *IEEE Internet of Things Journal*, 2025.

[123] Jean Baptiste Minani, Fatima Sabir, Yahia El Fellah, and Naouel Moha. Practical guidance for IoT systems testing: A taxonomy. In *Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things*, pages 57–64, 2024.

[124] Jean Baptiste Minani, Fatima Sabir, Yahia El Fellah, and Naouel Moha. Towards an automated approach for testing IoT devices. In *Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things*, pages 22–29, 2024.

[125] Jean Baptiste Minani, Fatima Sabir, Naouel Moha, and Yann-Gaël Guéhéneuc. A multi-method study of internet of things systems testing in industry. *IEEE Internet of Things Journal*, pages 1–1, 2023.

[126] Jean Baptiste Minani, Fatima Sabir, Naouel Moha, and Yann-Gaël Guéhéneuc. A systematic review of IoT systems testing: Objectives, approaches, tools, and challenges. *IEEE Transactions on Software Engineering*, 2024.

[127] Jean Baptiste Minani, Fatima Sabir, Naouel Moha, and Yann-Gaël Guéhéneuc. A Multi-Method Study of Internet of Things Systems Testing in Industry. *IEEE Internet of Things Journal*, pages 1–1, 2023.

[128] Jean Baptiste Minani, Fatima Sabir, Naouel Moha, and Yann-Gaël Guéhéneuc. A Systematic Review of IoT Systems Testing: Objectives, Approaches, Tools, and Challenges. *IEEE Transactions on Software Engineering*, 50(4):785–815, 2024.

[129] Jean Baptiste Minani, El Fellah Yahia, Trabelsi Imen, Naouel Moha, and Yann-Gaël Guéhéneuc. TGenAI: LLM-based Approach for Functional Test Cases Generation for IoT System. *Under review, Information and Software Technology (IST)*, 2024.

[130] Naemah Mubarakah et al. Software Engineering Taxonomy Reviews. In *2020 4rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM)*, pages 63–67. IEEE, 2020.

[131] Ghadeer Murad, Aalaa Badarneh, Abdallah Qusef, and Fadi Almasalha. Software Testing Techniques in IoT. In *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, pages 17–21. IEEE, 2018.

[132] Krishna Nadiminti, Marcos Dias De Assunçao, and Rajkumar Buyya. Distributed systems and recent innovations: Challenges and benefits. *InfoNet Magazine*, 16(3):1–5, 2006.

[133] SR Nagalakshmi and Meenakshi D'Souza. Coverage Criteria Based Testing of IoT Applications. In *International Conference on Distributed Computing and Intelligent Technology*, pages 101–116. Springer, 2024.

[134] Deepika Navani, Sanjeev Jain, and Maninder Singh Nehra. The Internet of Things (IoT): A Study of Architectural Elements. In *2017 13th International Conference on Signal-Image Technology And Internet-Based Systems (SITIS)*, pages 473–478, 2017.

[135] Mukrimah Nawir, Amiza Amir, Naimah Yaakob, and Ong Bi Lynn. Internet of Things (IoT): Taxonomy of security attacks. In *2016 3rd international conference on electronic design (ICED)*, pages 321–326. IEEE, 2016.

[136] Marlen Niederberger and Julia Spranger. Delphi technique in health sciences: a map. *Frontiers in public health*, 8:457, 2020.

[137] Premal Nirpal and Karbhari Kale. A Brief Overview Of Software Testing Metrics. *International Journal on Computer Science and Engineering*, 3, 01 2011.

[138] Node-RED. Node-RED. `https://nodered.org/`. [Accessed 10-Feb-2023].

[139] Khezemi Nour, Jean Baptiste Minani, Sabir Fatima, Naouel Moha, Yann-Gaël Guéhéneuc, and El Boussaidi Ghizlane. A Systematic Literature Review of IoT System Architectural Styles and Their Quality Requirements. *IEEE Internet of Things Journal*, 2024.

[140] Dario Olianas, Maurizio Leotta, and Filippo Ricca. An Approach and a Prototype Tool for Generating Executable IoT System Test Cases. In *International Conference on the Quality of Information and Communications Technology*, pages 383–398. Springer, 2020.

[141] Dario Olianas, Maurizio Leotta, and Filippo Ricca. An approach and a prototype tool for generating executable iot system test cases. In *International Conference on the Quality of Information and Communications Technology*, pages 383–398. Springer, 2020.

[142] Dario Olianas, Maurizio Leotta, and Filippo Ricca. MATTER: A tool for generating end-to-end IoT test scripts. *Software Quality Journal*, 30:1–35, 2022.

[143] Dario Olianas, Maurizio Leotta, and Filippo Ricca. MATTER: A tool for generating end-to-end IoT test scripts. *Software Quality Journal*, 30(2):389–423, 2022.

[144] Yazan Otoum, Dandan Liu, and Amiya Nayak. DL-IDS: a deep learning–based intrusion detection framework for securing IoT. *Transactions on Emerging Telecommunications Technologies*, 33(3):e3803, 2022. e3803 ett.3803.

[145] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. The PRISMA 2020 Statement: An Updated Guideline For Reporting Systematic Reviews. *International journal of surgery*, 88:105906, 2021.

[146] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. The PRISMA 2020 Statement: An Updated Guideline For Reporting Systematic Reviews. *International journal of surgery*, 88:105906, 2021.

[147] Matthew J Page, David Moher, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. PRISMA 2020 Explanation And Elaboration: Updated Guidance And Exemplars For Reporting Systematic Reviews. *bmj*, 372, 2021.

[148] N D Patel, B M Mehtre, and Rajeev Wankar. Simulators, Emulators, and Test-beds for Internet of Things: A Comparison. In *2019 Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 139–145, 2019.

[149] Murray G Patterson. What is energy efficiency?: Concepts, indicators and methodological issues. *Energy policy*, 24(5):377–390, 1996.

[150] Yusuf Perwej, Kashiful Haq, Firoj Parwej, M Mumdouh, and Mohamed Hassan. The internet of things (IoT) and its application domains. *International Journal of Computer Applications*, 975(8887):182, 2019.

[151] Luis Eduardo Pessoa, Cristovao Freitas Iglesias Jr, and Claudio Miceli. RITA: Automatic Framework for Designing of Resilient IoT Applications. *arXiv preprint arXiv:2411.18324*, 2024.

[152] Roberto Pietrantuono, Massimo Ficco, and Francesco Palmieri. Testing the resilience of MEC-based IoT applications against resource exhaustion attacks. *IEEE Transactions on Dependable and Secure Computing*, 21(2):804–818, 2023.

[153] Pedro Martins Pontes, Bruno Lima, and João Pascoal Faria. Izinto: a pattern-based IoT testing framework. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, page 125–131, NY, USA, 2018. Association for Computing Machinery.

[154] Pedro Martins Pontes, Bruno Lima, and João Pascoal Faria. Izinto: A Pattern-Based IoT Testing Framework. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, pages 125–131, 2018.

[155] Chittaranjan Pradhan, Sunil A Kinange, Jayavel Kanniappan, and Rajesh Kumar Jayavel. IoT automation test framework for connected ecosystem. In *International Conference on Intelligent Vision and Computing*, pages 309–320. Springer, 2021.

[156] T. Punter, M. Ciolkowski, B. Freimut, and I. John. Conducting on-line surveys in software engineering. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 80–88, 2003.

[157] Claudia Raibulet. Towards a Taxonomy for the Evaluation of Self-* Software. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 22–23, 2018.

[158] Paul Ralph. Toward methodological guidelines for process theories and taxonomies in software engineering. *IEEE Transactions on Software Engineering*, 45(7):712–735, 2018.

[159] Tariq Aziz Rao and EU Haq. Security challenges facing IoT layers and its protective measures. *International Journal of Computer Applications*, 179(27):31–35, 2018.

[160] S Reid. Software and Systems Engineering Software Testing Part 1: Concepts and Definitions. Technical report, ISO/IEC/IEEE 29119-1, 2013.

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

[162] Robert Roggio, Jamie Gordon, and James Comer. Taxonomy of Common Software Testing Terminology: Framework for Key Software Engineering Testing Concepts. *Journal of Information Systems Applied Research*, 7(2):4, 2014.

[163] Philipp Rosenkranz, Matthias Wählisch, Emmanuel Baccelli, and Ludwig Ortmann. A distributed test system architecture for open-source IoT software. In *Proceedings:2015 Workshop on IoT challenges in Mobile and Industrial Systems*, pages 43–48. INRIA, 2015.

[164] Tobias Schnabel and Jennifer Neville. Prompts As Programs: A Structure-Aware Approach to Efficient Compile-Time Prompt Optimization. *arXiv preprint arXiv:2404.02319*, April 2024.

[165] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Transactions on Software Engineering*, 50(1), 2024.

[166] Selenium. Selenium. `https://www.selenium.dev/`. [Accessed 10-Feb-2023].

[167] Inc. Amazon Web Services. IoT Device Simulator. `https://aws.amazon.com/solutions/implementations/iot-device-simulator/`, 2021. [Accessed 15-Feb-2023].

[168] Pallavi Sethi and Smruti R Sarangi. Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering*, 2017:1–25, 2017.

[169] Sanjay Kumar Singh and Amarjeet Singh. *Software Testing*. Vandana Publications, 2012.

[170] Yogesh Singh, Arvinder Kaur, and Bharti Suri. An empirical study of product metrics in software testing. *Innovative techniques in instruction technology, e-learning, e-assessment, and education*, pages 64–72, 2008.

[171] Holm Smidt, Matsu Thornton, and Reza Ghorbani. Smart application development for IoT asset management using graph database modeling and high-availability web services. *Hawaii*, 01 2018.

[172] IEEE Computer Society. IEEE Standard for Software and System Test Documentation. *IEEE Std 829-2008*, pages 1–150, 2008.

[173] ISO Standards. IEEE/ISO/IEC International Standard - Software and systems engineering–Software testing: Test techniques. *ISO/IEC/IEEE 29119-4:2021(E)*, pages 1–148, 2021.

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

[175] Antero Taivalsaari and Tommi Mikkonen. On the development of IoT systems. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 13–19, 2018.

[176] Teik-Boon Tan and Wai-Khuen Cheng. Software Testing Levels in Internet of Things (IoT) Architecture. In Chuan-Yu Chang, Chien-Chou Lin, and Horng-Horng Lin, editors, *New Trends in Computer Technologies and Applications*, pages 385–390, Singapore, 2019. Springer, Springer Singapore.

[177] A Tanenbaum and MV Steen. Introduction to distributed systems. *Distributed Systems: Principles and Paradigms, Prentice Hall (Jan. 15, 2002)*, pages 1–33, 2015.

[178] Omer Tauqeer, Sadeeq Jan, Alaa Khadidos, Adil Khadidos, Fazal Khan, and Sana Khattak. Analysis of Security Testing Techniques. *Intelligent Automation and Soft Computing*, 29:291–306, 05 2021.

[179] Tewari, Ramanuj and Alsalami, Zaid and Alsailawi, HA and Kirubanantham, P and Dhabalia, Dharmesh and Saadoun, Osama Nazim and Abdulhussain, Zahraa N and Alwan, Ali Saad. Testing user Attitudes in IoT-load interface Based Healthcare attention management system. In *2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pages 831–834. IEEE, 2024.

[180] Haseeb Touqeer, Shakir Zaman, Rashid Amin, Mudassar Hussain, Fadi Al-Turjman, and Muhammad Bilal. Smart home security: challenges, issues and solutions at different IoT layers. *The Journal of Supercomputing*, 77(12):14053–14089, 2021.

[181] Imen Trabelsi, Brahim Mahmoudi, Jean Baptiste Minani, Naouel Moha, and Yann-Gaël Guéhéneuc. A Systematic Literature Review of Machine Learning Approaches for Migrating Monolithic Systems to Microservices. *IEEE Transactions on Software Engineering*, 2025. Under revision.

[182] Phillip Tracy. MQTT protocol minimizes network bandwidth for the internet of things. `https://www.rcrwireless.com/20161129/featured/mqtt-internet-of-things-tag31-tag99`. [Accessed 18-Feb-2023].

[183] Huynh Khanh Vi Tran, Michael Unterkalmsteiner, Jürgen Börstler, and Nauman bin Ali. Assessing test artifact quality—A tertiary study. *Information and Software Technology*, 139:106620, 2021.

[184] M. Unterkalmsteiner, R. Feldt, and T. Gorschek. A Taxonomy for Requirements Engineering and Software Test Alignment. *ACM Transactions on Software Engineering and Methodology*, 23(2):1–38, Apr 2014.

[185] Muhammad Usman, Ricardo Britto, Jürgen Börstler, and Emilia Mendes. Taxonomies in Software Engineering: A Systematic Mapping Study and a Revised Taxonomy Development Method. *Information and Software Technology*, 85:43–59, 2017.

[186] Sira Vegas, Natalia Juristo, and Victor R. Basili. Maturing Software Engineering Knowledge through Classifications: A Case Study on Unit Testing Techniques. *IEEE Transactions on Software Engineering*, 35(4):551–565, 2009.

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

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

[189] Isabel Karina Villanes, Erick Alexandre Bezerra Costa, and Arilo Claudio Dias-Neto. Automated mobile testing as a service (AM-TaaS). In *2015 IEEE World Congress on Services*, pages 79–86. IEEE, 2015.

[190] Baoping Wang, Linkai Zhu, Di Sun, Wennan Wang, Shiyang Song, and Sheng Peng. Towards an Automated Testing Framework for IoT Devices. In *Journal of Physics: Conference Series*, volume 2493, page 012023. IOP Publishing, 2023.

[191] Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel C. Briand. Automatic Generation of Acceptance Test Cases From Use Case Specifications: An NLP-Based Approach. *IEEE Transactions on Software Engineering*, 48(2):585–616, 2022.

[192] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Transactions on Software Engineering*, 50(4):911–936, 2024.

[193] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 2024.

[194] George R Wheaton. Development of a Taxonomy of Human Performance: A Review of Classificatory Systems Relating To Tasks And Performance. *Sage Journals*, 1968.

[195] Gary White, Vivek Nallur, and Siobhán Clarke. Quality of Service Approaches in IoT: A Systematic Mapping. *Journal of Systems and Software*, 132:186–203, 2017.

[196] www.thinxtream.com. IoT Testing Challenges and Approaches. `https://www.thinxtream.com/iot-testing-solutions-services.html`, 2020. [White Paper.Accessed 10-Feb-2023].

[197] Zhiyi Xue, Liangguo Li, Senyue Tian, Xiaohong Chen, Pingping Li, Liangyu Chen, Tingting Jiang, and Min Zhang. LLM4Fin: Fully Automating LLM-Powered Test Case Generation for FinTech Software Acceptance Testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1643–1655, 2024.

[198] Geeta Yadav, Kolin Paul, Alaa Allakany, and Koji Okamura. IoT-pen: A penetration testing framework for IoT. In *2020 International Conference on Information Networking (ICOIN)*, pages 196–201. IEEE, 2020.

[199] El Fellah Yahia, Jean Baptiste Minani, Naouel Moha, Gascon-Samson Julien, and Yann-Gaël Guéhéneuc. Analysis of Microservices-Based IoT Systems: Deployment Challenges, Industry Practices, and Performance Insights. *Under review, IEEE Internet of Things Journal*, 2025.

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

[201] Myoungsung You, Yeonkeun Kim, Jaehan Kim, Minjae Seo, Sooel Son, Seungwon Shin, and Seungsoo Lee. FuzzDocs: an automated security evaluation framework for IoT. *IEEE Access*, 10:102406–102420, 2022.

[202] Juliette C Young, David C Rose, Hannah S Mumby, Francisco Benitez-Capistros, Christina J Derrick, Tom Finch, Carolina Garcia, Chandrima Home, Esha Marwaha, Courtney Morgans, et al. A methodological guide to using and reporting on interviews in conservation science research. *Methods in Ecology and Evolution*, 9(1):10–19, 2018.

[203] Shengcheng Yu, Chunrong Fang, Yuchen Ling, Chentian Wu, and Zhenyu Chen. LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, pages 206–217, 2023.

[204] Tao Yue, Lionel C Briand, and Yvan Labiche. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):1–38, 2013.

[205] Tao Yue, Lionel C Briand, and Yvan Labiche. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):1–38, 2013.

[206] Muhammad Nouman Zafar, Wasif Afzal, and Eduard Enoiu. Towards a workflow for model-based testing of embedded systems. In *Proceedings of the 12th International Workshop on Automating TEST Case Design, Selection, and Evaluation*, pages 33–40, 2021.

[207] Justyna Zander and Ina Schieferdecker. A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains. In *Model-Based Testing for Embedded Systems*, 2011.

[208] Shicheng Zhu, Shunkun Yang, Xiaodong Gou, Yang Xu, Tao Zhang, and Yueliang Wan. Survey Of Testing Methods And Testbed Development Concerning Internet Of Things. *Wireless Personal Communications*, 123(1):165–194, 2022.

Primary Studies (PSs) for SLR

[209] Mahmoud AbdelHafeez and Mohamed AbdelRaheem. Assiut IoT: A Remotely Accessible Testbed For Internet Of Things. In *2018 IEEE Global Conference on Internet of Things (GCIoT)*, pages 1–6. IEEE, 2018.

[210] Menatalla Abououf, Rabeb Mizouni, Shakti Singh, Hadi Otrok, and Ernesto Damiani. Self-Supervised Online and Lightweight Anomaly and Event Detection for IoT Devices. *IEEE Internet of Things Journal*, 9(24):25285–25299, 2022.

[211] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, et al. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE, 2015.

[212] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. Model-Based Testing As A Service For IoT Platforms. In *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II 7*, pages 727–742. Springer, 2016.

[213] Nayef Abdulwahab Mohammed Alduais, Jiwa Abdullah, Ansar Jamil, Lukman Audah, and R Alias. Sensor Node Data Validation Techniques For Realtime IoT/WSN Application. In *2017 14th International Multi-Conference on Systems, Signals & Devices (SSD)*, pages 760–765. IEEE, 2017.

[214] Domenico Amalfitano, Nicola Amatucci, Vincenzo De Simone, Vincenzo Riccio, and Fasolino Anna Rita. Towards a Thing-In-the-Loop Approach For The Verification And Validation Of IoT Systems. In *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*, pages 57–63, 2017.

[215] Ilja Behnke, Lauritz Thamsen, and Odej Kao. Héctor: A Framework For Testing IoT Applications Across Heterogeneous Edge And Cloud Testbeds. In *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing companion*, pages 15–20, 2019.

[216] Stig Bosmans, Siegfried Mercelis, Joachim Denil, and Peter Hellinckx. Testing IoT Systems Using A Hybrid Simulation-Based Testing Approach. *Computing*, 101:857–872, 2019.

[217] Chiara Buratti, Andrea Stajkic, Gordana Gardasevic, Sebastiano Milardo, M Danilo Abrignani, Stefan Mijovic, Giacomo Morabito, and Roberto Verdone. Testing Protocols for the Internet of Things on the EuWIn Platform. *IEEE Internet of Things Journal*, 3(1):124–133, 2015.

[218] Miroslav Bures, Bestoun S Ahmed, Vaclav Rechtberger, Matej Klima, Michal Trnka, Miroslav Jaros, Xavier Bellekens, Dani Almog, and Pavel Herout. PatrIoT: IoT Automated Interoperability And Integration Testing Framework. In *2021 14th IEEE Conference on Software Testing,Verification and Validation (ICST)*, pages 454–459. IEEE, 2021.

[219] Miroslav Bures, Xavier Bellekens, Karel Frajtak, and Bestoun S Ahmed. A Comprehensive View On Quality Characteristics Of The IoT Solutions. In *3rd EAI International Conference on IoT in Urban Space*, pages 59–69. Springer, 2020.

[220] Woei-Kae Chen, Chien-Hung Liu, William W-Y Liang, and Ming-Yi Tsai. ICAT: An IoT Device Compatibility Testing Tool. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 668–672. IEEE, 2018.

[221] Loubna Chhiba, Abdelaziz Marzak, and Mustapha Sidqui. Quality Attributes for Evaluating IoT Healthcare Systems. In *Innovations in Smart Cities Applications Volume 5: The Proceedings of the 6th International Conference on Smart City Applications*, pages 495–505. Springer, 2022.

[222] Soumya Kanti Datta, Christian Bonnet, Hamza Baqa, Mengxuan Zhao, and Franck Le-Gall. Approach For Semantic Interoperability Testing In Internet Of Things. In *2018 Global Internet of Things summit (GIoTS)*, pages 1–6. IEEE, 2018.

[223] Soumya Kanti Datta, Christian Bonnet, Hamza Baqa, Mengxuan Zhao, and Franck Le-Gall. Developing And Integrating A Semantic Interoperability Testing Tool In F-Interop Platform. In *2018 IEEE Region Ten Symposium (Tensymp)*, pages 112–117. IEEE, 2018.

[224] Senay Tuna Demirel, Mehmet Demirel, Ibrahim Dogru, and Resul Das. InterOpT: A new testing platform based on oneM2M standards for IoT Systems. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2019.

[225] João Pedro Dias, Hugo Sereno Ferreira, and Tiago Boldt Sousa. Testing and Deployment Patterns for the Internet-of-Things. In *Proceedings of the 24th European Conference on Pattern Languages of Programs*, pages 1–8, 2019.

[226] John Esquiagola, Laisa Caroline de Paula Costa, Pablo Calcina, Geovane Fedrecheski, and Marcelo Zuffo. Performance Testing Of An Internet Of Things Platform. In *IoTBDS*, pages 309–314, 2017.

[227] K. Fizza, P.P. Jayaraman, A. Banerjee, D. Georgakopoulos, and R. Ranjan. Evaluating Sensor Data Quality in Internet of Things Smart Agriculture Applications. *IEEE Micro*, 42(1):51–60, 2022.

[228] Yi Gao, Jiadong Zhang, Gaoyang Guan, and Wei Dong. LinkLab: A Scalable And Heterogeneous Testbed For Remotely Developing And Experimenting IoT Applications. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 176–188. IEEE, 2020.

[229] V. Geetha Lekshmy and J.M. Kannimoola. Formal Verification Of IoT Protocol: In Design-Time And Run-Time Perspective. *Lecture Notes in Networks and Systems*, 145:873–884, 2021.

[230] Gleiston Guerrero-Ulloa, Miguel J Hornos, and Carlos Rodríguez-Domínguez. TDDM4IoTS:A Test-Driven Development Methodology For Internet Of Things (IoT)-Based Systems. In *Applied Technologies: First International Conference, ICAT 2019, Quito, Ecuador, December 3–5, 2019, Proceedings, Part I*, pages 41–55. Springer, 2020.

[231] Ligia Georgeta Guşeilă, Dragoş-Vasile Bratu, and Sorin-Aurel Moraru. Continuous Testing In The Development Of IoT Applications. In *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, pages 1–6. IEEE, 2019.

[232] Lorena Gutiérrez-Madroñal, Antonio García-Domínguez, and Inmaculada Medina-Bulo. Evolutionary Mutation Testing For IoT With Recorded And Generated Events. *Software: Practice and Experience*, 49(4):640–672, 2019.

[233] Lorena Gutiérrez-Madroñal, Inmaculada Medina-Bulo, and Juan José Domínguez-Jiménez. IoT-TEG: Test Event Generator System. *Journal of Systems and Software*, 137:784–803, 2018.

[234] Md Mahmud Hossain, Shahid Al Noor, Yasser Karim, and Ragib Hasan. IoTBed: A Generic Architecture for Testbed as a Service for Internet of Things-Based Systems. In *ICIoT*, pages 42–49, 2017.

[235] Linghuan Hu, W Eric Wong, D Richard Kuhn, Raghu N Kacker, and Shuo Li. CT-IoT: A Combinatorial Testing-Based Path Selection Framework For Effective Iot Testing. *Empirical Software Engineering*, 27:1–38, 2022.

[236] Jaeyoung Hwang, Abdullah Aziz, Nakmyoung Sung, Abbas Ahmad, Franck Le Gall, and Jaeseung Song. AUTOCON-IoT: Automated And Scalable Online Conformance Testing For IoT Applications. *IEEE Access*, 8:43111–43121, 2020.

[237] Koray İnçki and Ismail Ari. A Novel Runtime Verification Solution For IoT Systems. *IEEE Access*, 6:13501–13512, 2018.

[238] Koray Inçki, İsmail Arı, and Hasan Sözer. Runtime Verification Of IoT Systems Using Complex Event Processing. In *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, pages 625–630. IEEE, 2017.

[239] Alexander Kaiser and Sascha Hackel. Standards-Based IoT Testing With Open-Source Test Equipment. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 435–441. IEEE, 2019.

[240] Yasser Karim and Ragib Hasan. FogTestBed: A Generic Architecture For Testbed For Fog-Based Systems. In *2020 SoutheastCon*, pages 1–7. IEEE, 2020.

[241] Eunsook Eunah Kim and Sebastien Ziegler. Towards An Open Framework Of Online Interoperability And Performance Tests For The Internet Of Things. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[242] Hiun Kim, Abbas Ahmad, Jaeyoung Hwang, Hamza Baqa, Franck Le Gall, Miguel Angel Reina Ortega, and JaeSeung Song. IoT-TaaS: Towards A Prospective IoT Testing Framework. *IEEE Access*, 6:15480–15493, 2018.

[243] Jay Kiruthika and Souheil Khaddaj. Software Quality Issues And Challenges Of Internet Of Things. In *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 176–179. IEEE, 2015.

[244] Moez Krichen. Improving Formal Verification And Testing Techniques For Internet Of Things And Smart Cities. *Mobile networks and applications*, pages 1–12, 2019.

[245] Moez Krichen and Mariam Lahami. Towards A Runtime Testing Framework For Dynamically Adaptable Internet Of Things Networks In Smart Cities. *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*, pages 589–607, 2020.

[246] Daniel Kuemper, Thorben Iggena, Ralf Toenjes, and Elke Pulvermueller. Valid. IoT: A Framework For Sensor Data Quality Analysis And Interpolation. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 294–303, 2018.

[247] Daniel Kümper, ES Reetz, Marten Fischer, E Pulvermueller, and R Tönjes. From Semantic IoT-Service Descriptions To Executable Test Cases-Information Flow Of An Implemented Test Framework. In *VALID 2014-6th International Conference on Advances in System Testing and Validation Lifecycle*. International Academy, Research and Industry Association, IARIA, 2014.

[248] Rémy Leone, Federico Sismondi, Thomas Watteyne, and César Viho. Technical Overview of F-Interop. In *Interoperability, Safety and Security in IoT: Second International Conference, InterIoT 2016 and Third International Conference, SaSeIoT 2016, Paris, France, October 26-27, 2016, Revised Selected Papers 2*, pages 11–17. Springer, 2017.

[249] Maurizio Leotta, Davide Ancona, Luca Franceschini, Dario Olianas, Marina Ribaudo, and Filippo Ricca. Towards A Runtime Verification Approach For Internet Of Things Systems. In *Current Trends in Web Engineering: ICWE 2018 International Workshops, MATWEP, EnWot, KD-WEB, WEOD, TourismKG, Cáceres, Spain, June 5, 2018, Revised Selected Papers 18*, pages 83–96. Springer International Publishing, 2018.

[250] Maurizio Leotta, Diego Clerissi, Dario Olianas, Filippo Ricca, Davide Ancona, Giorgio Delzanno, Luca Franceschini, and Marina Ribaudo. An Acceptance Testing Approach For Internet Of Things Systems. *IET Software*, 12(5):430–436, 2018.

[251] Maurizio Leotta, Filippo Ricca, Diego Clerissi, Davide Ancona, Giorgio Delzanno, Marina Ribaudo, and Luca Franceschini. Towards An Acceptance Testing Approach For Internet Of Things Systems. In *Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17*, pages 125–138. Springer, 2018.

[252] Xinyao Liu, Baojiang Cui, Junsong Fu, and Jinxin Ma. HFuzz: Towards Automatic Fuzzing Testing Of NB-IoT Core Network Protocols Implementations. *Future Generation Computer Systems*, 108:390–400, 2020.

[253] Nhan Ly-Trong, Chuong Dang-Le-Bao, Dang Huynh-Van, and Quan Le-Trung. UiTIOT v3: A Hybrid Testbed For Evaluation Of Large-Scale IoT Networks. In *Proceedings of the 9th International Symposium on Information and Communication Technology*, pages 155–162, 2018.

[254] Amir Makhshari and Ali Mesbah. IoT Bugs And Development Challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 460–472. IEEE, 2021.

[255] A Malini, AB Yugakiruthika, Sarita Pappa Gunasekar, and R Preethi. Testing As A Service Focused on Semantic Interoperability: An Approach. In *2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)*, pages 1–5. IEEE, 2021.

[256] Erik Jan Marinissen, Yervant Zorian, Mario Konijnenburg, Chih-Tsun Huang, Ping-Hsuan Hsieh, Peter Cockburn, Jeroen Delvaux, Vladimir Rožić, Bohan Yang, Dave Singelée, et al. IoT: Source Of Test Challenges. In *2016 21th IEEE European test symposium (ETS)*, pages 1–10. IEEE, 2016.

[257] Qudsia Mateen, Mehreen Sirshar, et al. Software Quality Assurance in Internet of Things. *Int. J. Comput. Appl*, 109(9):16–24, 2015.

[258] Noha Medhat, Sherin Moussa, Nagwa Badr, and Mohamed F. Tolba. Testing Techniques in IoT Based Systems. In *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 394–401, 2019.

[259] Noha Medhat, Sherin M Moussa, Nagwa Lotfy Badr, and Mohamed F Tolba. A Framework For Continuous Regression And Integration Testing In IoT Systems Based On Deep Learning And Search-Based Techniques. *IEEE Access*, 8:215716–215726, 2020.

[260] Ghadeer Murad, Aalaa Badarneh, Abdallah Qusef, and Fadi Almasalha. Software Testing Techniques in IoT. In *2018 8th International conference on computer science and information technology (CSIT)*, pages 17–21. IEEE, 2018.

[261] Sang Nguyen, Zoran Salcic, Xuyun Zhang, and Akshat Bisht. A Low-Cost Two-Tier Fog Computing Testbed For Streaming IoT-Based Applications. *IEEE Internet of Things Journal*, 8(8):6928–6939, 2020.

[262] Fotis Nikolaidis, Manolis Marazakis, and Angelos Bilas. IoTier: A Virtual Testbed To Evaluate Systems For IoT Environments. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 676–683. IEEE, 2021.

[263] Michael Norris, Berkay Celik, Prasanna Venkatesh, Shulin Zhao, Patrick McDaniel, Anand Sivasubramaniam, and Gang Tan. IoTRepair: Systematically Addressing Device Faults In Commodity IoT. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 142–148. IEEE, 2020.

[264] Dario Olianas, Maurizio Leotta, and Filippo Ricca. MATTER: A Tool For Generating End-To-End IoT Test Scripts. *Software Quality Journal*, 30(2):389–423, 2022.

[265] Mani Padmanabhan. A Study On Transaction Specification Based Software Testing For Internet Of Things. In *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pages 1–6. IEEE, 2018.

[266] Ciprian Păduraru, Rareş Cristea, and Eduard Stăniloiu. RiverIoT: A Framework Proposal For Fuzzing IoT Applications. In *2021 IEEE/ACM 3rd International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT)*, pages 52–58. IEEE, 2021.

[267] M.R. Palattella, F. Sismondi, T. Chang, L. Baron, M. Vučinić, P. Modernell, X. Vilajosana, and T. Watteyne. F-Interop Platform And Tools: Validating IoT Implementations Faster. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11104 LNCS:332–343, 2018.

[268] Pedro Martins Pontes, Bruno Lima, and João Pascoal Faria. Izinto: A Pattern-Based IoT Testing Framework. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, pages 125–131, 2018.

[269] Svitlana Popereshnyak, Olha Suprun, Oleh Suprun, and Tadeusz Wieckowski. IoT Application Testing Features Based On The Modelling Network. In *2018 XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 127–131. IEEE, 2018.

[270] Brian Ramprasad, Joydeep Mukherjee, and Marin Litoiu. A Smart Testing Framework For IoT Applications. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 252–257. IEEE, 2018.

[271] Eike Steffen Reetz, Daniel Kuemper, Klaus Moessner, and Ralf Tönjes. How To Test IoT-Based Services Before Deploying Them Into Real World. In *European Wireless 2013; 19th European Wireless Conference*, pages 1–6. VDE, 2013.

[272] Paulo Alexandre Regis, Amar Nath Patra, and Shamik Sengupta. Low-Cost Wireless Testbed For Internet Of Things Ad Hoc Networks Prototyping And Evaluation. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE, 2020.

[273] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. SmartSantander: IoT Experimentation Over a Smart City Testbed. *Computer Networks*, 61:217–238, 2014.

[274] Hunor Sándor, Béla Genge, and Zoltán Szántó. Sensor Data Validation And Abnormal Behavior Detection In The Internet of Things. In *2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pages 1–5. IEEE, 2017.

[275] Ina Schieferdecker, Sascha Kretzschmann, Axel Rennoch, and Michael Wagner. IoT-Testware: An Eclipse Project. In *2017 IEEE international conference on software quality, reliability and security (QRS)*, pages 1–8. IEEE, 2017.

[276] Joanna Sendorek, Tomasz Szydlo, and Robert Brzoza-Woch. Software-Defined Virtual Testbed For IoT Systems. *Wireless Communications and Mobile Computing*, 2018:1–11, 2018.

[277] Martin Serrano, Amelie Gyrard, Michael Boniface, Paul Grace, Nikolaos Georgantas, Rachit Agarwal, Payam Barnagui, Francois Carrez, Bruno Almeida, Tiago Teixeira, et al. Cross-Domain Interoperability Using Federated Interoperable Semantic IoT/Cloud Testbeds And Applications: The FIESTA-IoT Approach. In *Building the Future Internet through FIRE*, pages 287–321. River Publishers, 2022.

[278] V. Sharma, J. Kim, S. Kwon, I. You, and H.-C. Chen. Fuzzy-Based Protocol For Secure Remote Diagnosis Of IoT Devices In 5G Networks. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 246:54–63, 2018.

[279] Manisha Singh and Gaurav Baranwal. Quality of Service (QoS) in Internet of Things. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6. IEEE, 2018.

[280] Manisha Singh, Gaurav Baranwal, and Anil Kumar Tripathi. QoS-Aware Selection of IoT-Based Service. *Arabian Journal for Science and Engineering*, 45(12):10033–10050, 2020.

[281] Prasannjeet Singh, Francesco Flammini, Mauro Caporuscio, Mehdi Saman Azari, and Johan Thornadtsson. Towards Self-Healing in the Internet of Things by Log Analytics and Process Mining. In *30th European Safety and Reliability Conference and the 15th Probabilistic Safety Assessment and Management Conference,(ESREL2020 PSAM15), 01–05 November 2020, Venice, Italy*, pages 4644–4651, 2020.

[282] Teik-Boon Tan and Wai-Khuen Cheng. Software Testing Levels in Internet of Things (IoT) Architecture. In *New Trends in Computer Technologies and Applications: 23rd International Computer Symposium, ICS 2018, Yunlin, Taiwan, December 20–22, 2018, Revised Selected Papers 23*, pages 385–390. Springer, 2019.

[283] Martin Tappler, Bernhard K Aichernig, and Roderick Bloem. Model-Based Testing IoT Communication Via Active Automata Learning. In *2017 IEEE International conference on software testing, verification and validation (ICST)*, pages 276–287. IEEE, 2017.

[284] Yuuichi Teranishi, Yuki Saito, Sakae Murono, and Nozomu Nishinaga. JOSE: An Open Testbed For Field Trials Of Large-Scale IoT Services. *Journal of the National Institute of Information and Communications Technology*, 62(2):151–159, 2016.

[285] Feng-Ke Tsai, Chien-Chih Chen, Tien-Fu Chen, and Tay-Jyi Lin. Sensor Abnormal Detection And Recovery Using Machine Learning For IoT Sensing Systems. In *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 501–505. IEEE, 2019.

[286] Adrien van Den Bossche, Réjane Dalcé, and Thierry Val. Locura4IoT: A Testbed Dedicated To Accurate Localization Of Wireless Nodes In The IoT. *IEEE Sensors Journal*, 22(6):5437–5446, 2021.

[287] Antonio Velez-Estevez, Lorena Gutiérrez-Madroñal, and Inmaculada Medina-Bulo. IoT-TEG 4.0: A New Approach 4.0 for Test Event Generation. *IEEE Transactions on Reliability*, 71(3):1368–1380, 2021.

[288] Michael A Walker, Douglas C Schmidt, and Abhishek Dubey. Testing At Scale Of IoT Blockchain Applications. In *Advances in Computers*, volume 115, pages 155–179. Elsevier, 2019.

[289] Li Yi, Junyan Ma, and Te Zhang. HATBED: A Distributed Hardware Assisted Testbed For Non-Invasive Profiling Of IoT Devices. In *Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*, pages 13–17, 2019.

[290] Wenbo Zhang, Jiaxing Wang, Guangjie Han, Shuqiang Huang, Yongxin Feng, and Lei Shu. A Data Set Accuracy Weighted Random Forest Algorithm For IoT Fault Detection Based On Edge Computing And Blockchain. *IEEE Internet of Things Journal*, 8(4):2354–2363, 2020.

[291] S. Ziegler, S. Fdida, C. Viho, and T. Watteyne. F-Interop: Online Platform Of Interoperability And Performance Tests For The Internet Of Things. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 190:49–55, 2017.

Appendix A

Supplementary Information for SLR

Table A.1: Publication Sources

Document Type	Source Title	# PSs	% PSs
Journal	IEEE Access	4	4.8%
Journal	IEEE Internet of Things Journal	4	4.8%
Conference	ACM/IEEE International Conference on Internet of Things Design and Implementation (IoTDI)	3	3.6%
Book Chapter	EAI/Springer Innovations in Communication and Computing	2	2.4%
Conference	Global Internet of Things Summit (GIoTS)	2	2.4%
Conference	IEEE/ACM International Conference on Utility and Cloud Computing Companion	2	2.4%
Conference	IEEE International Conference on Software Quality Reliability and Security	2	2.4%
Conference	IEEE International Conference on Software Testing, Verification and Validation	2	2.4%
Conference	IEEE World Forum on Internet of Things	2	2.4%
Conference	International Conference on Safety and Security in IoT, International Conference on Interoperability in IoT	2	2.4%
Conference	International Conference on Smart City Applications	2	2.4%
Conference	International Conference on Web Engineering	2	2.4%
Conference	Springer Communications in Computer and Information Science (CCIS)	2	2.4%
Workshop	ACM International Workshop on the Internet of Safe Things	1	1.2%
Conference	ACM Multimedia Systems Conference	1	1.2%
Book Chapter	Advances in Computers	1	1.2%
Journal	Arabian Journal for Science and Engineering	1	1.2%
Conference	Asia-Pacific Software Engineering Conference	1	1.2%
Book Chapter	Building the Future Internet Through FIRE: 2016 FIRE Book	1	1.2%
Journal	Computer Networks	1	1.2%
Journal	Computing	1	1.2%
Journal	Empirical Software Engineering	1	1.2%
Conference	European Safety and Reliability Conference, Probabilistic Safety Assessment and Management Conference	1	1.2%
Workshop	European Test Workshop	1	1.2%
Conference	European Wireless Conference	1	1.2%
Journal	Future Generation Computer Systems	1	1.2%
Conference	IEEE Global Conference on Internet of Things (GCIoT))	1	1.2%
Conference	IEEE/ACM International Symposium on Cluster Cloud and Internet Computing	1	1.2%
Workshop	IEEE/ACM International Workshop on Software Engineering Research and Practices for the IoT	1	1.2%
Conference	IEEE International Conference on Intelligent Computing and Information Systems	1	1.2%
Conference	IEEE International Congress on Internet of Things	1	1.2%
Journal	IEEE Micro	1	1.2%
Conference	IEEE Region 10 Symposium	1	1.2%
Journal	IEEE Sensors Journal	1	1.2%
Conference	IEEE SoutheastCon	1	1.2%
Journal	IEEE Transactions on Reliability	1	1.2%
Journal	IET Software	1	1.2%
Conference	International Conference on Ad-Hoc Networks and Wireless	1	1.2%
Conference	International Conference on Advances in System Testing and Validation Lifecycle	1	1.2%
Conference	International Conference on Computer Science and Information Technology	1	1.2%
Conference	International Conference on Current Trends towards Converging Technologies	1	1.2%
Conference	International Conference on Industrial Engineering and Applications	1	1.2%
Conference	International Conference on Internet of Things as a Service	1	1.2%
Conference	International Conference on Internet of Things Smart Innovation and Usages	1	1.2%
Conference	International Conference on Internet of Things, Big Data and Security	1	1.2%
Conference	International Conference on Inventive Communication and Computational Technologies	1	1.2%
Conference	International Conference on Nascent Technologies in Engineering	1	1.2%
Conference	International Conference on Networking Sensing and Control	1	1.2%
Conference	International Conference on Perspective Technologies and Methods in MEMS Design	1	1.2%
Conference	International Conference on Sensing and Instrumentation in IoT Era	1	1.2%
Conference	International Conference on Software Engineering	1	1.2%
Journal	International Journal of Computer Applications	1	1.2%
Conference	International Multi-Conference on Systems, Signals, and Devices	1	1.2%
Conference	International Symposium on Distributed Computing and Applications for Business, Engineering, and Science	1	1.2%
Conference	International Symposium on Leveraging Applications of Formal Methods	1	1.2%
Conference	International Symposium on Networks, Computers, and Communications	1	1.2%
Workshop	International Symposium on Software Testing, and Analysis (ISSTA/ECOOP Workshops 2018)	1	1.2%
Journal	Journal of Systems and Software	1	1.2%
Journal	Journal of the National Institute of Information and Communications Technology	1	1.2%
Journal	Mobile Networks and Applications	1	1.2%
Conference	RoEduNet Conference: Networking in Education and Research	1	1.2%
Journal	Software - Practice and Experience	1	1.2%
Journal	Software Quality Journal	1	1.2%
Journal	Wireless Communications and Mobile Computing	1	1.2%
Workshop	Workshop on Benchmarking Cyber-Physical Systems and Internet of Things	1	1.2%
Total		**83**	**100.0%**

Table A.2: Authors With Two or More PSs

Author Name	# PSs	Affiliation	Author Name	# PSs	Affiliation
Franck Le Gall	6	Easy Global Market (EGM), France	Christian Bonnet	2	EURECOM, France
Maurizio Leotta	4	Università di Genova, Italy	Diego Clerissi	2	Università di Genova, Italy
Filippo Ricca	4	Università di Genova, Italy	Soumya Kanti Datta	2	EURECOM, France
Thomas Watteyne	4	INRIA, France	Giorgio Delzanno	2	Università di Genova, Italy
Abbas Ahmad	3	EGM, France	Ragib Hasan	2	University of Alabama at Birmingham, USA
Davide Ancona	3	Università di Genova, Italy	Jaeyoung Hwang	2	Sejong University, South Korea
Luca Franceschini	3	Università di Genova, Italy	Koray Incki	2	Özyeğin University, Turkey
Lorena Gutierrez-Madronal	3	University of Cádiz, Spain	Jaeseung Song	2	Sejong University, South Korea
Hamza Baqa	3	EGM, France	Yasser Karim	2	University of Alabama at Birmingham, USA
Daniel Kuemper	3	University of Applied Scie Osnabrueck, Germany	Moez Krichen	2	Al-Baha University, Saudi Arabia; University of Sfax, Tunisia
Inmaculada Medina-Bulo	3	University of Cádiz, Spain	Luis Sanchez	2	University of Cantabria, Spain
Dario Olianas	3	Università di Genova, Italy	Noha Medhat	2	Ain Shams University, Egypt
Marina Ribaudo	3	Università di Genova, Italy	Mengxuan Zhao	2	EGM, France
Ralf Toenjes	3	University of Applied Scie Osnabrueck, Germany	Sherin Moussa	2	Ain Shams University, Egypt
Bestoun S. Ahmed	2	Czech Technical University, Czech Republic	Elke Pulvermueller	2	Institute of Computer Science, University of Osnabrueck, Germany
Ismail Ari	2	Özyeğin University, Istanbul, Turkey	Manisha Singh	2	Banaras Hindu University, India
Nagwa L. Badr	2	Ain Shams University, Egypt	Federico Sismondi	2	IRISA, France
Gaurav Baranwal	2	Banaras Hindu University, India	Mohammed Tolba	2	Ain Shams University, Egypt
Xavier Bellekens	2	Czech Technical University, Czech Republic	Cesar Viho	2	IRISA, France
Miroslav Bures	2	Czech Technical University, Czech Republic	Sebastien Ziegler	2	Mandat International, Switzerland

✺ **INRIA**: Institut National de Recherche en Informatique et en Automatique.