IoT Bugs and Development Challenges

by

Amir Makhshari

BSc., Amirkabir University of Technology, Iran, 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF APPLIED SCIENCE
(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

August 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

IoT Bugs and Development Challenges

submitted by **Amir Makhshari** in partial fulfillment of the requirements for the degree of **Master of Applied Science** in **Electrical and Computer Engineering**.

Examining Committee:

Ali Mesbah
Supervisor, Electrical and Computer Engineering Department

Karthik Pattabiraman, Electrical and Computer Engineering Department
Committee Member

Sid Fels, Electrical and Computer Engineering Department
Committee Chair

Abstract

IoT systems are rapidly adopted in various domains, from embedded systems to smart homes. Despite their growing adoption and popularity, there has been no thorough study to understand IoT development challenges from the practitioners' point of view. We provide the first systematic study of bugs and challenges that IoT developers face in practice, through a large-scale empirical investigation. We collected 5,565 bug reports from 91 representative IoT project repositories and categorized a random sample of 323 based on the observed failures, root causes, and the locations of the faulty components. In addition, we conducted nine interviews with IoT experts to uncover more details about IoT bugs and to gain insight into IoT developers' challenges. Lastly, we surveyed 194 IoT developers to validate our findings and gain further insights. We propose the first bug taxonomy for IoT systems based on our results.

We highlight frequent bug categories and their root causes, correlations between them, and common pitfalls and challenges that IoT developers face. We recommend future directions for IoT areas that require research and development attention.

Preface

The work presented in this thesis was conducted by the author, Amir Makhshari, under the supervision of Professor Ali Mesbah. I was responsible for devising the approach and the experiments, implementing the tools, running the experiments, evaluating and analyzing the results, and writing the manuscript. My collaborator guided me with the creation of the methodology and the analysis of results, as well as editing and writing portions of the manuscript. Chapter 2 and 3 of this thesis is an extended version of the paper we published in the Proceedings of the 43rd International Conference on Software Engineering (ICSE'21) [50].

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor, Dr. Ali Mesbah, for his careful supervision, unhesitating support, and considerate guidance throughout the course of this research. Without his critical reviews and intellectual inputs, the completion of this thesis would not have been possible for me. I am also sincerely thankful to Dr. Karthik Pattabiraman and Dr. Sid Fels for accepting to be a part of my defence committee.

I would also like to thank my friends and colleagues at Software Testing and Analysis Lab who always helped me with their constructive feedback. Last but certainly not least, I would like to thank my family for their love and constant support.

Chapter 1

Introduction

Internet of Things (IoT) envisions a self-configuring, adaptive, and complex network that interconnects smart objects, embedded with sensors or actuators, to the internet through the use of communication protocols [51]. By 2020, Gartner estimates that smart inter-connected devices will outnumber humans 4-to-1 [34] and it is estimated that by 2025, there will be over 75.44 billion smart things worldwide [63]. These smart "*things*" can be programmed and remotely controlled to collect their data or to control their actions. Programming physical devices with constraint resources, dealing with diverse network protocols, and the integration of diverse entities in IoT systems, add unique characteristics to the challenges of developing such systems. Driven by the above considerations, the concept of bugs in IoT is more complicated than traditional software.

Previously, some studies have investigated the characteristics of IoT repositories [19], and discussed some challenges of IoT systems [17, 33, 68]. Existing research on bug categorization is limited to specific sub-domains of IoT such as bugs in smart aquaculture systems [14], bugs in IoT device operating systems [45], or bugs uncovered during the deployment of IoT systems [33].

While more mature software domains have benefited from empirical and qualitative studies on their bugs and developer challenges [37, 40, 79], no such study is available for IoT to the best of our knowledge.

Overall, existing work on IoT are either not generalizable or they do not consider experiences of IoT developers to draw conclusions about the characteristics

of IoT development.

In this paper, we provide a generalized and systematic overview of bugs and developer challenges in IoT systems. In order to do so, we mine IoT GitHub repositories and collect (5,565) bug reports from 91 representative IoT projects. By applying Root Cause Analysis (RCA), we categorize a subset of 323 bug reports considering the observed failures, root causes, and locations of the bugs. We propose the first taxonomy of bugs in IoT systems, which is constructed by analyzing these real-world IoT bugs. To complement the taxonomy and study the challenges of IoT developers, we conducted semi-structured interviews with nine IoT practitioners that have hands-on experience in different layers of IoT. Lastly, we validated our findings through an online survey that was completed by 194 IoT developers.

1.1 Contributions

We conducted a systematic empirical study involving more than 90 real-world IoT projects and 200 IoT developers to characterize IoT bugs and development challenges. The detailed list of thesis contributions is as follows:

- We provide the first benchmark dataset of representative IoT repositories for future software engineering research in the context of IoT. This dataset consists of 91 IoT projects collected by mining all 8,774 IoT repositories on GitHub, and carefully analyzing the resulting projects. These repositories are in various programming languages and cover all different layers of IoT. This dataset is publicly available in our replication package [49].

- We provide a benchmark dataset of 5,565 valid IoT bugs collected from representative IoT projects by mining GitHub issues. This dataset is publicly available [49] and can be used for future research on IoT bugs.

- We propose the first multi-faceted and comprehensive bug taxonomy for IoT systems by carefully analyzing a subset of 323 IoT bugs, and considering inputs from IoT developers through semi-structured one-on-one interviews with IoT developers, and an online survey that involved a large and diverse set of IoT developers.

- We apply Root Cause Analysis (RCA) to each analyzed bug report and provide categories of IoT bugs' root causes.
- We investigate all bug categories based on their frequency, perceived severity, and their correlations using analyzed data from GitHub discussions, interviews and our online survey.
- We characterize the state-of-the-practice challenges faced by IoT developers using all our various input sources.
- We provide real-world concrete examples and contextual data from through GitHub discussions, interviews, and survey comments in order to give more detailed insights into each development challenges.

Our findings show that general development issues, device management issues, and messaging issues are the most frequent bug categories in the context of IoT. Furthermore, the most frequent root causes of IoT bugs are software programming faults, semantic faults, and dependency faults. In terms of challenges, our findings highlight the obstacles of IoT developers in various areas such as testing, debugging, dealing with heterogeneity, and security in IoT.

1.2 Thesis Organization

In chapter two of the thesis, we provide detailed background information about different layers of IoT system. We also provide an in-depth case study of a real-world IoT bug that we collected from GitHub by going through the steps IoT developer gone through to debug the issue and the challenges they faced throughout the process. We use this example as the motivation for our study. In chapter 3, we provide our bug analysis methodology and our findings regarding IoT bugs. Chapter 4 starts with a description of our methodology and discuss our findings regarding IoT development challenges. Chapter 5 discusses the related work, and chapter 6 includes conclusions, future work and possible research directions.

Chapter 2

Background

2.1 IoT Architecture

Figure Figure 2.2 shows a typical architecture of an IoT system which is based on the architectures defined in previous studies [15, 25, 51, 68].

2.1.1 Device Layer

The device layer at the bottom, which is the closest component to the physical world, includes smart programmable things interacting with the environment through their embedded sensors and actuators. Some IoT devices employ light embedded operating systems (e.g. contiki, RIOT, TinyOS) that have support for various programming languages allowing developers to write embedded code, such as application-specific logics for IoT devices, on top of the device OS [38]. IoT operating systems aim to hide the minor details of the device hardware and provide basic OS tasks, like memory management and real-time scheduling, efficiently[38]. On the other side of spectrum, bare metal IoT devices run the embedded code directly on their hardware processor. Underlying details of the device hardware is usually abstracted by hardware abstraction layer (HAL) to enable access for the operating system to various hardware components like GPIO, and serial interfaces[25]. In addition, IoT devices should have a connectivity capability which includes drivers, external libraries, and communication protocols to allow them communicate over a

local wired or wireless communication, which allows devices to be remotely controlled by external entities.

2.1.2 Gateway Layer

This layer contains gateway devices with fewer resource constraints with the ability to handle telemetry data collection, processing, and routing locally on the edge. Gateway devices can handle the device-device and device-cloud interoperability by interpreting diverse communication protocols such as MQTT, CoAP, and HTTP [75].

The IoT gateway devices have less resource constraint and the ability to handle telemetry data collection, processing, and routing locally on the edge. These devices can offer secure communication with remote systems since they can employ full-stack communication protocols and security mechanisms[9].

There are different types of IoT devices based on their processing, power, and storage capabilities [9] which affects the way they communicate with the cloud systems over the internet. Gateway-connected devices, as the name suggested in [85], are designed to do a limited functionality due to their constrained resources, thus, they rely on gateway devices for specific application purposes, data management, or secure communication with remote cloud systems [9]. On the other hand, cloud-connected devices have the ability to talk directly to the cloud by employing protocols designed specifically for constrained devices like Constrained Application Protocol (CoAP). They should have enough resources to do basic data pre-processing and to apply security mechanisms by themselves without the help of a gateway device [68, 85]. Gateway devices can handle the device-device and device-cloud interoperability by interpreting diverse communication protocols of both sides[15]. Both types of IoT devices allow higher level components to control, monitor, and upgrade them remotely [25].

2.1.3 Cloud and Application Layer

Remote IoT cloud servers accumulate and process all telemetry data, and communicate with a large number of heterogeneous gateway or cloud-connected devices to control and monitor them remotely. IoT cloud servers rule engine lets users write automation logic between IoT devices to define interoperability behaviours of the

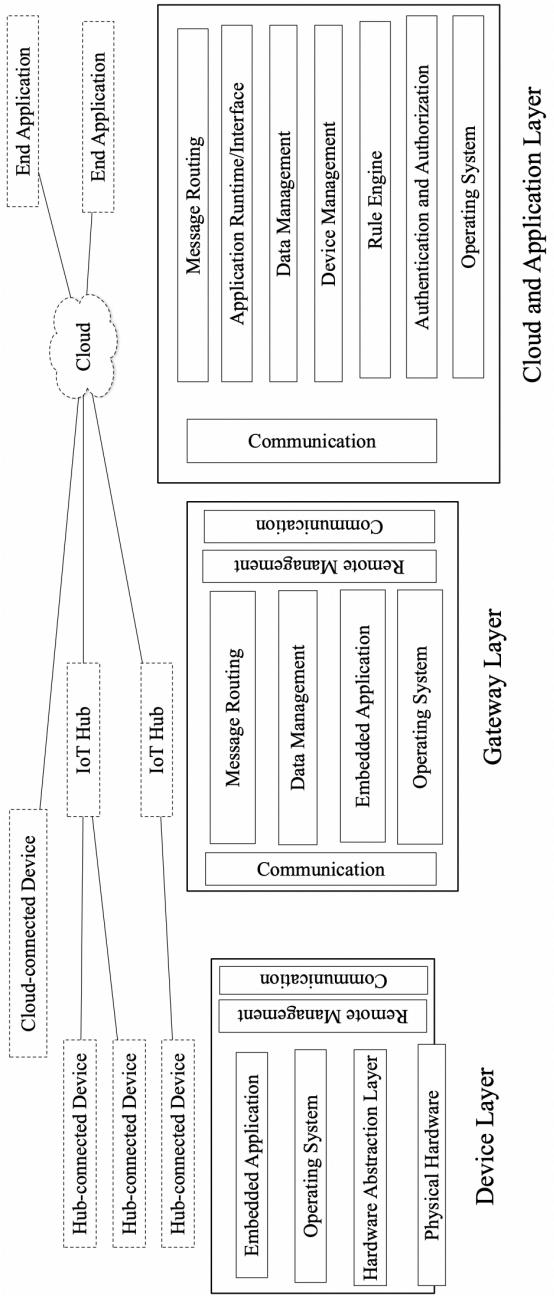

Figure 2.1: A typical layered architecture of IoT systems and software stack for each layer

IoT system [85].

End users and end applications can monitor the telemetry data and device status, and send commands to IoT devices to control and configure them remotely from the cloud. Usually, external end users and end applications should be authenticated and authorized in the cloud to be able to use these services. Some IoT cloud providers offer an interface for third-party applications, and allow users to run custom code in the application runtime environment on top of the cloud services[15] and also allow them to have direct access to the analytical data via various visualizations and customized dashboards.

2.2 Motivation

Dealing with bugs and their causes in IoT systems is not a trivial task for developers. Below, we will do an in-depth investigation of a real bug report in an IoT system as an example of how dealing with bugs can be time-consuming for IoT developers. In addition to discussing the bug, we also showcase some challenges IoT developers have went through while debugging.

Figure 2.2 shows (on the right side) the steps that IoT developers have taken to find the root cause of the bug report PYTRADFRI/135 [23]. This bug occurred in a smart home environment where different devices should be connected to a home automation server with the help of a gateway device.

Based on developers' discussions, the first manifestation of this bug happens at the application layer. A light bulb device (D3) is mistakenly recognized as a sensor device (F_1) as the user adds it, and also the Android app crashes (F_2) once it tries to turn the light bulb off (step 1 and 2). Initially, the developers suspected the root cause of the bug to be the incompatibility of a gateway library with the home automation server (step 3).

However, further investigation of the failure in the edge layer (step 4), revealed that the gateway misidentifies D3 as a remote controller (F_3). Since the gateway in this system relies mainly on a specific format of response data from devices to identify their types properly, potential inconsistencies of the payload data from the subject device were also investigated (step 5). In addition, developers should consider the device type, model, manufacturer, and the firmware version to detect

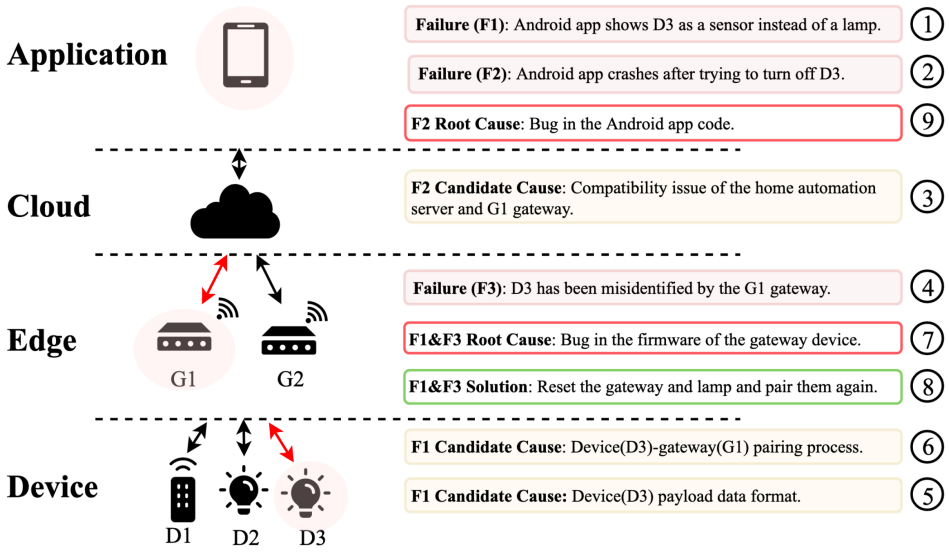

Figure 2.2: The steps IoT developers have went through to find the root cause of a real IoT bug occurred in [23].

possible breaking changes in the device firmware by the device manufacturer.

But, further investigation eliminated the possibility of the device payload data as the root cause as some users reported that the same setting has not caused any issue for them. As the next step developers tried pairing in different circumstances (step 6), since the device battery and distance to the gateway could affect the pairing process. After none of the candidate causes showed as the potential reason for the bug, the root cause was identified as an external bug in the firmware of the G2 gateway device (step 7) and resetting both the device and the gateway and pairing them again solved the issue (step 8). But surprisingly F_2 continued to exist. After monitoring the data that the app receives via Wireshark, the root cause of F_2 was identified as a fault in the Android app code (step 9).

This example shows how one bug in the firmware of the gateway device, can make a lamp in the device layer useless, and cause observable failures in edge and application layers. Distribution of IoT failures and their candidate causes in different layers of IoT required a wide range of development skills from developers, as we could see that developers have to both investigate low-level firmware code and debug high-level application code to find the root cause of a single bug. Also dealing with behaviours of diverse devices such as comparing their payload, and depending on naive debugging practices such as monitoring network traffic, make it extremely hard for developers to deal with IoT bugs. Another challenging factor that we observed in this case study, is the interference of physical factors like the battery and the distance of the IoT devices from the gateway devices, which makes the debugging process rather time-consuming and complex.

Despite these compelling challenges, previous studies [14, 17, 33, 45, 68] do not take into consideration real-world experiences of IoT developers. In this paper, we aim to provide a systematic and generalized understanding of bugs and challenges in IoT by mainly considering IoT developers' experiences.

Chapter 3

IoT Bugs

3.1 Methodology

Our goal from this study is to characterize software bugs in IoT systems. To this end, we address the following research questions in this study:

- `RQ1`: What are the classes of bugs in IoT systems?
- `RQ2`: What are the root causes of IoT bugs?
- `RQ3`: What are the perceived severity and frequency of IoT bugs by real IoT developers?
- `RQ4`: What IoT bugs are more likely to happen together and have higher correlation?

In order to answer these questions, we conduct an empirical investigation consisting of two phases. In the first phase (RQ1), we analyze 323 issues and pull requests (PR) from open-source IoT projects. We use the findings to form the first taxonomy of bugs in IoT systems. In the second phase (RQ2), we conduct a qualitative study through (1) semi-structured interviews with IoT developers to discover new categories of bugs, (2) a survey of IoT developers to validate the findings about IoT bugs and gain new insights. All our quantitative and qualitative data is available online [49]

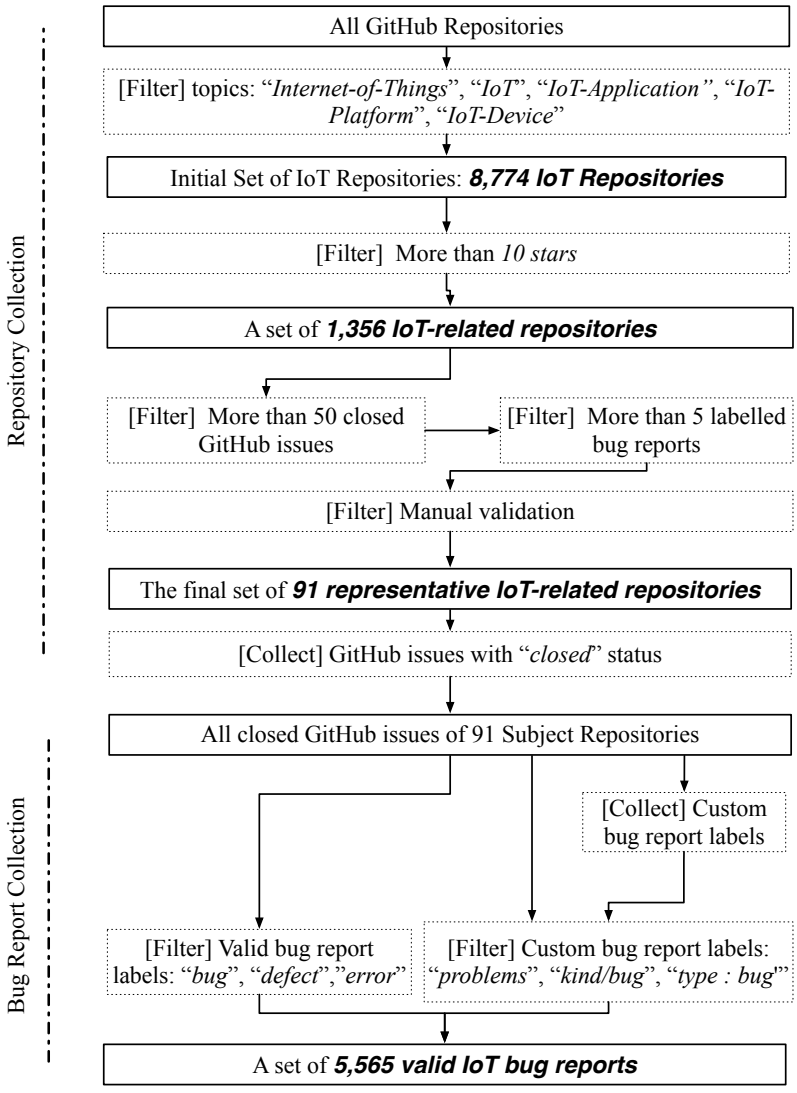

Figure 3.1: Our approach to collect subject IoT repositories and their valid bug reports.

3.1.1 IoT Bug Categorization

Collecting bug reports

Figure Figure 3.1 shows the steps we took to find out subject IoT repositories and collect valid bug reports from them. The initial step is to find repositories that are representative of IoT projects. We employed the “GitHub topic feature” to find IoT-related repositories. According to GitHub’s official website [1], Topics are labels that create subject-based connections between GitHub repositories. Repository owners can use multiple tags for their repository to help users in exploring projects based on their technology, subject area, or domain.

We searched among topics with related keywords such as "internet-of-things" and "IoT" and we added the top three topics “IoT-application”, “IoT-platform”, and “IoT-device” from the results to our list of targeted topics. Initially, we collected 8,774 repositories using these five topics in January 2020. Inspecting a random sample of 30 repositories showed that 86% of them have less than 5 issued bug reports, while 50% of them did not have a single bug report. A previous study [8] found that 3 out of 4 developers check the stars metric before using or contributing to a GitHub project. Thus, In order to reduce the number of GitHub API requests, we excluded repositories with less than 10 stars [8], resulting in 1,356 repositories to consider.

To only consider valid bugs, we looked for issued bug reports with only “closed” status and with “bug”’, “defect”, or “error” labels. Moreover, to select only representative IoT repositories for our study, we manually analyzed repositories that have more than five labeled issues or more than 50 closed issues based on the information in their readme page, issued bug reports, and their website (when available). We then excluded projects that were not representative of IoT systems such as user interface, documentation, or outdated repositories. Our final project list contained 91 open source IoT repositories to analyze. For five repositories that use custom labels (e.g., “problems”, “kind/bug”, “type : bug”), we manually added their labels to our search keywords. In the end, we collected *5,565 bug reports* from the 91 IoT repositories.

Figure Figure 3.2 shows the demographics of our subject repositories. The se-

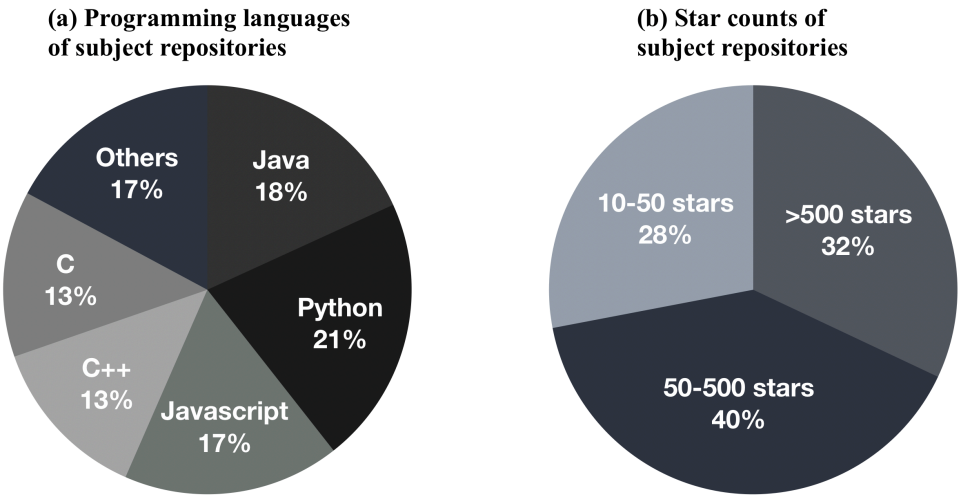

Figure 3.2: (a) Programming languages of subject repositories and (b) their star count.

lected IoT repositories together cover all the layers of the IoT systems' architecture depicted in Figure 2.2. The most popular programming languages among the subject repositories are Python (21%), Java (18%), JavaScript (17%), C (13%), and C++ (13%). A few of them use other programming languages such as Go, Ruby, and C#. The selected repositories are also diverse in terms of the number of stars and forks they have. In February of 2020, 32% of our subject GitHub repositories had more than 500 stars, 40% between 50 to 500 stars, and 28% between 10 to 50 stars.

Labeling

Figure Figure 3.3 shows the steps we took to label bug reports. For each bug report in our dataset, we created a JSON object containing failure(s), cause(s) of the failure, and location(s) of the faulty code. Failure refers to any observable unexpected behavior of the system that is against the correct functionality of that system [7, 70]. To explore the causes of the failure, we did RCA on each bug report using the *five whys technique* [65]. Based on this technique, multiple causes can contribute together or at different levels to a visible failure in the system. Following this approach, we started from the failure and repeatedly asked "why" until we

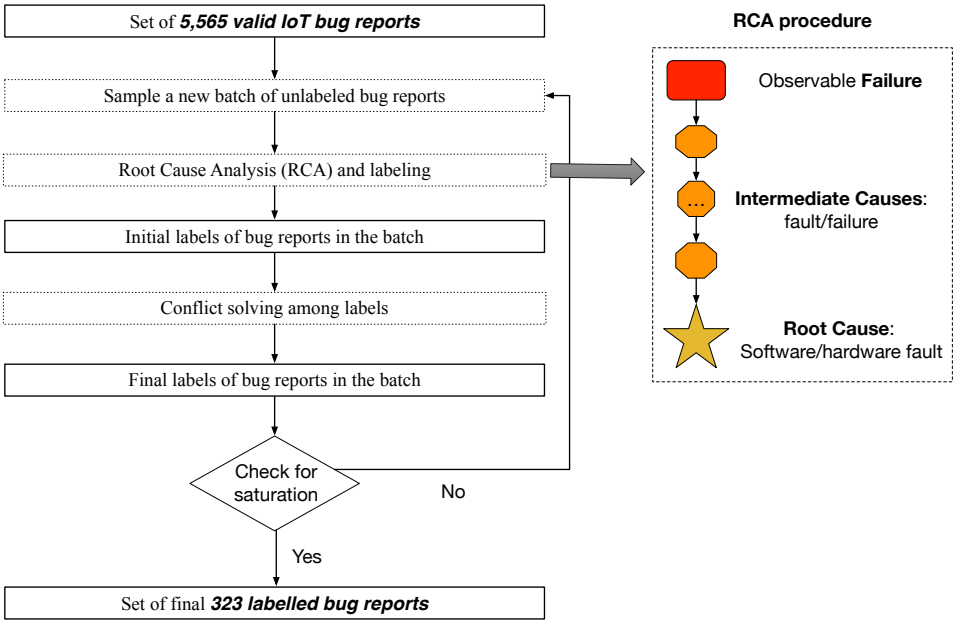

Figure 3.3: (a) Programming languages of subject repositories and (b) their star count.

reached the root cause of the problem. In the case of software failures, root causes are often developers' faults in the design or implementation of the IoT system. To label the location(s) of faults, we used the architecture defined in Chapter 2 as our reference.

Bug reports were manually labeled by the author(s) individually following the open coding procedure [64]. We followed an iterative process for labeling where in each iteration, we randomly sampled new instances from the collected bug reports and labeled them. After each labeling iteration, all potential conflicts in labels between the author(s) were resolved. We continued this process until bug categories reached a state of saturation where no new category appeared [22].

We also flagged and discarded issues and PRs that could not represent a bug or a bug-fix such as enhancements or how-to-use questions. We examined the entire discussion among developers, as well as the fix commit data (e.g. the commit message and the code diff) to label each bug report. At the end, we labeled *323 bug reports*.

Table 3.1: Interview Participants

ID	Role	IoT Systems Type	Projects Domain	Dev Exp (yr)	IoT Dev Exp (yr)
P1	Software and hardware lead	Full-stack	Smart home	13	7
P2	Hardware lead	Hardware	Education	15	5
P3	Software dev	Full-stack	Smart home	5	4
P4	Software dev	Middleware	Smart city	3	3
P5	Software lead	Full-stack	Smart home	20	17
P6	Software dev	Cloud	Not domain-specific	10	3
P7	Software and hardware dev	Full-stack	Smart home	20	3
P8	Software lead	Full-stack	Smart home	11	4
P9	Software lead	Cloud	Not domain-specific	20	12

3.1.2 Interviews

While manual analysis of bug reports and developers' discussions provided useful insights into the characteristics of IoT development, there was still a possibility that our bug categories are not generalizable. To mitigate this issue, we conducted semi-structured interviews with IoT developers to reveal new bug categories to complement and validate our results.

Participants

To collect interview participants, we employed GitHub as it provides a diverse pool of developers and their contributions to different projects.

To collect candidate interview participants, we followed the steps shown in figure Figure 3.4. We used the 1,356 IoT repositories we collected in section **??**, as the IoT projects with more than 10 stars, to sample interview participants. After collecting all the 11,129 contributors to these 1356 repositories, we removed duplicate contributors, resulting in 6,878 contributors. Duplicate contributors are the IoT developers involved in more than one repository from the set of subject repositories and thus appeared more than once in our initial dataset. The next step is to find the valid email addresses of these unique contributors.

GitHub API does not provide the email addresses of the developers directly. We searched through the information returned by GitHub public API from the requests regarding contributors' commits to extract their email addresses. This was the only way we could find to extract the email addresses of contributors to IoT repositories. A manual run of this method showed that the email addresses of a handful of the contributors are unavailable or invalid. First of all, we removed the contributors that their email address has not appeared in any of the GitHub API's responses

since our goal was to contact developers only via their email addresses. Moreover, we removed the contributors that their email address contains the string "no-reply" and its invariants. After all the filtering, we could obtain the valid email addresses of 1,847 unique contributors to popular IoT repositories.

For interviews, we only added the top three contributors to the set of candidate interviewers. We used purposive sampling [73] to recruit developers with adequate experience in developing IoT systems.

We contacted candidates through emails and conducted interviews until we reached data saturation, where we had sufficient data to replicate the study and further data collection is unnecessary [22]. We relied on this widely-applied methodological principle to decide when to stop interviewing [27, 53] as it is also used in other qualitative studies in software engineering [6, 66]. We interviewed people with different development backgrounds and experiences before deciding about the data saturation to consider variability in experimental results across different populations [31].

Table 3.1 presents all nine interview participant's experience and field of expertise in IoT development. For a high-level picture of their general development background, the lowest value is three years and the highest value is 20 years (avg=13, sdv=6.4). In terms of IoT development experience, the lowest value is three years and the highest value is 17 years (avg=6.4, sdv=4.6). Participants' IoT development experience covers all sections of IoT systems spanning from hardware to middleware, cloud, and end-applications. In addition, their projects cover a variety of domains such as smart home and Industrial IoT (IIoT).

Protocol Since our goal for conducting interviews was to be open to new data, and we did not have the definitive structure of bug categories, we conducted interviews following a semi-structured approach. Interviews started with some questions about participants' IoT development background and their field of expertise in IoT. This information could help us improvising insightful questions during the technical section of the interviews. The technical section had a combination of both open-ended and specific questions about different categories of bugs and challenges in IoT development. Our strategy was to start with open-ended questions to avoid biasing participants toward our findings, and then we gradually shifted to more structured and predefined questions during the interview process. With par-

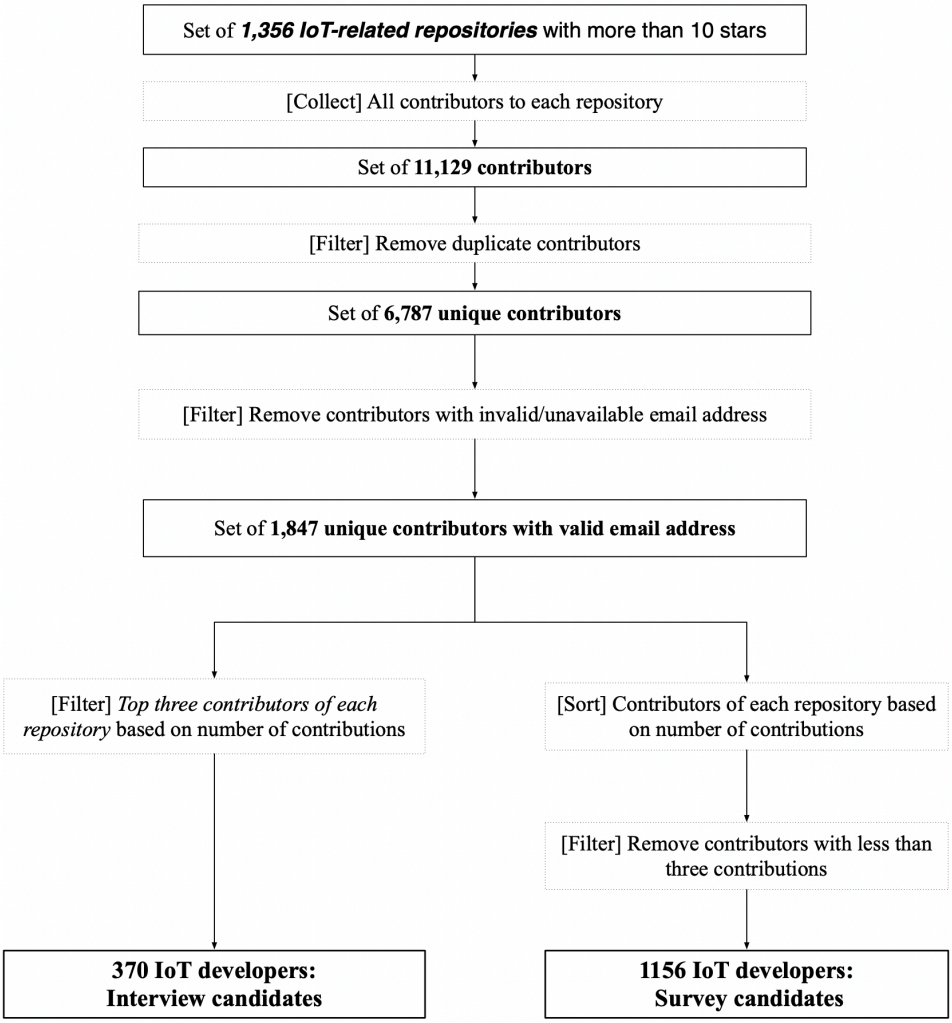

Figure 3.4: Our methodology for finding candidates for interviews and survey.

ticipants' consent, we recorded the audio and video of all the interviews for later analysis. All the interviews were conducted remotely through Zoom. Interviews took around 43 minutes on average ranging from 31–70 minutes. We used Descript, an automated speech detection tool, to generate transcribes of the interviews and we did manual corrections afterward in case of any mistakes in the automatically generated transcripts.

Analysis As the primary objective of this study is to generate theories from the experiences of IoT practitioners instead of using pre-conceived theories, we followed the grounded theory methodology [16] to ensure the quality of the generated theory. Our analysis steps consist of iteratively (i) collecting qualitative data from the interviews (ii) analyzing the interview transcript line by line and assigning labels (tags) to distinct units of meanings, and (iii) identifying emerging categories and relating categories to their subcategories while continuously comparing all the previously analyzed data with the emerging theories. These steps were repeated for each interview. On average, we extracted 18 tags per interview. Potential conflicts in the labels were resolved after each iteration by the authors.

3.1.3 Validation Survey

In order to make sure that our findings are generalizable, comprehensive, and representative, we involved more IoT developers through an online survey.

Participants Figure Figure 3.4 shows how we collected the survey participant candidates. Firstly, we sorted the collected unique contributors with valid email addresses based on the number of contributions they had to the 1,356 subject IoT projects. After removing the contributors with less than three contributions, we started to send out our survey to the resulting candidate participants until we reached saturation in our results. We gave higher priority to the developers with more contributions by sending our survey first to the developers with higher contributions.

We also sent out our survey to the IoT developer groups in social media platforms such as Linkedin and Facebook, and online forums. Our survey was online between 19 July and 19 August 2020. The survey, as distributed to participants, is available in our artifact package [49].

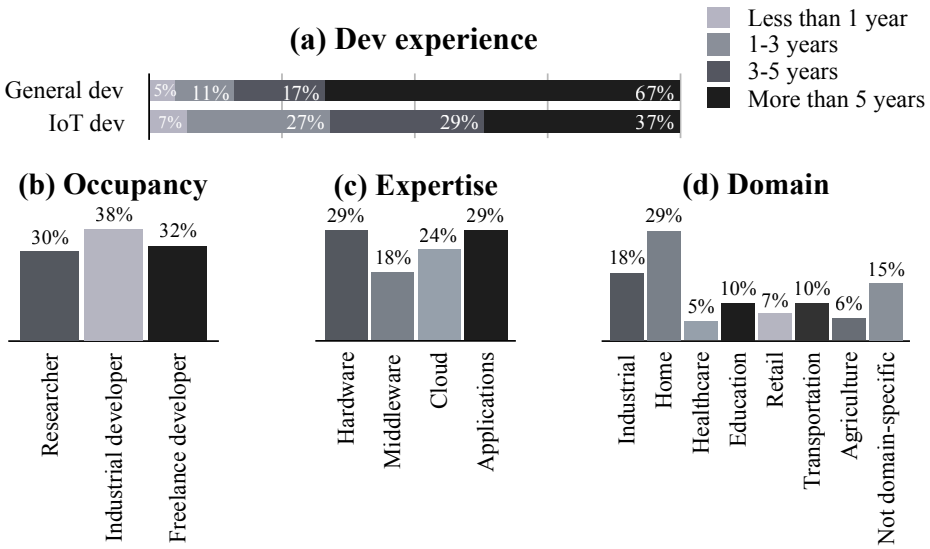

Figure 3.5: Development backgrounds of the survey participants

Protocol The survey has three sections. In the first section, we collect participants' background in IoT, and in general development, which is depicted in Figure 3.5. The second section concerns the challenges of developing IoT systems with questions aimed at correlating our findings with the participants' own experiences.

The third section is about the frequency and severity of bug categories based on participants' previous experience in IoT development. At the end of each section, there are open-ended questions to allow them to share their comments about our results and mention new categories.

Analysis Our survey was completed by 194 respondents, with a response rate of around 10% for valid responses. We received 95 comments through the open-ended questions sections. All of the survey respondents' comments are coded and analyzed following the same procedure discussed in Figure 3.1.2.

3.2 Findings: IoT Bug Categories (RQ1)

In this section, we describe our findings regarding IoT bugs.

3.2.1 Taxonomy of Bugs

We used all the tags collected by RCA of the bug reports in our dataset, to build a taxonomy of bugs in IoT systems. As our motivating example illustrates **??**, IoT bugs can be multi-faceted and manifest at different layers and locations. Therefore, we designed our bug taxonomy to accommodate all these bug characteristics. Considering various approaches suggested by Usma et. al. [76] for taxonomy construction, we followed the approach suggested by Kwasnik [44] as IoT bugs are multi-faceted and relatively a new and unexplored concept. Following their approach, we first defined facets of our classification as all failures and locations of failures. Then, we analyzed all bug reports based on these facets and built a hierarchical taxonomy that accommodates all the dimensions.

After we built the initial version of the taxonomy, we used the data from the interviews and the survey to complement and enhance the taxonomy. We reviewed all previously tagged data and re-tagged them after each alteration of the taxonomy. Figure 3.6 depicts our IoT bug taxonomy. Next, we describe the major bug categories in our taxonomy. We will use specific bugs as examples for each category. All these bug examples are available in our dataset, which is available online [49].

IoT Device This category of taxonomy covers bugs that are related to IoT device hardware and firmware.

Device hardware: Bugs in this subcategory are related to the physical aspects of IoT devices. Examples include bugs related to wiring issues, device pin status issues, or issues with physical sensors and actuators of the device. For example, PEDALINOMINI/34 is related to the device not differentiating between single and double presses of a hardware button.

Other common bugs in this category are those that are linked to the device's limitations in memory, power consumption, or processing capacity. One such instance provided by P_1 as he described a scenario where a device on low battery generated incorrect data to the cloud. There are various similar cases in our collected bug reports where the low battery of the device or the removal of the power source of the device causes failures. Even in some cases, enabling power saving mode cause unexpected behaviours such as variable lag. For example, in DEVICE-OS/1567, the power saving mode causes some variable lag, which led IoT devel-

opers to disable the power saving mode as they had not accounted for such delays.

Another group of device hardware issues is the problems regarding booting or rebooting the device. For example, heavy calculations and processing on the device (HOMIE-ESP8266/575, interviewee P_8) cause a device boot loop issue. Also, there are cases where the IoT device runs out of memory (ZWAVE2MQTT/141, interviewee P_7) which are other types of bugs related to this category. Another example of known hardware issues of IoT devices is the known timing issues of Raspberry Pi devices [78], which is also mentioned by an interviewee (P_8).

Device firmware: Firmware bugs consist of three subcategories. The first pertains to device firmware unexpected exception and hang issues. The second subcategory includes issues related to the configuration of the IoT device, which can be specified as an external instruction sent to the device for a specific purpose. This type of bug usually happens in the early stages of introducing an IoT device to the IoT network. Each device has to be configured properly in a way to be compatible with other hardware or software components and also be able to communicate with others on the network. Issues associated with configuring the device with WiFi credentials or with configuring the device with the correct firmware version are some common examples here. The third and most common sub-category is the firmware upgrade issue. There are various cases where poor practices for handling over-the-air (OTA) updates of the device firmware, stale updates, or updating the device firmware with the wrong binary have caused failures of the IoT system. In some cases, the stale update issues are related to device configuration issues as in WTHERMOSTATBECA/54, where the device needs to be re-configured with WiFi credentials after each firmware update, otherwise, future firmware updates would be stale.

Compatibility When a bug occurs only on a specific type of device, communication protocol, or third-party component, it falls under the compatibility category. For instance, a common device incompatibility issue happens when certain devices represent their telemetry data in different formats, leading to the other components not being able to process their data. Other common bugs are linked to compatibility issues of certain combinations of sensors and development boards, e.g., incompatibility of the DHT temperature sensor with the ESP32 microcontroller in MONGOOSE-OS/277. Issues with the interoperability of different protocols is

another case. One example is MAINFLUX/1079, which is related to the interoperability between the HTTP and MQTT protocols. A common bad practice in IoT development related to these issues is developing protocol-specific or device-specific code. For instance, in DEVICE-OS/1938, the IoT platform relies on event components to report what protocols each event is intended for, in order to be able to run different functions for each protocol individually. However, sometimes developers have no other choice but to follow this error-prone approach, just to bypass the limitations of third-party devices. For instance, (P_2) mentioned a case where the incompatibility of the Raspberry Pi and some types of sensors had forced their developers to implement custom logic for the communication of these devices. Developers had to switch between Raspberry Pi's default implementation and their own custom implementation based on the sensor type, leading to many issues.

Communication with IoT devices Bugs that are related to the communication of IoT devices with each other or with other entities fall under this category. Generally, there are two types of bugs in this category:

Device Connectivity: Some of the connectivity issues are related to the network that the device relies on for connecting to the internet. One example is when the device cannot discover a valid and available network such as a local access point and therefore loses access to the internet. As it is also mentioned by P_9 "When the device location is changed to another room or another building, the device has to be reconfigured for the new access point."

```
// Initial Wifi Connection
  WiFi.connect();
  // asserting socket creation are done correctly
 const sock_handle_t sock = socket_create();
  if (!socket_handle_valid(sock))   Log.error("socket_create() failed");
 // Asserting that socket connection works without error
 const auto r = socket_connect();
  if (r != 0)  Log.error("socket_connect() failed");
// WiFi Reconnection and asserting the socket closure operation
 WiFi.off();
 WiFi.connect();
const auto r2 = socket_close(sock);
if (r2 != 0)  Log.error("socket_close() failed");
```

Listing 3.1: An example of testing a bug fix after WiFi reconnection in DEVICE-OS/1639

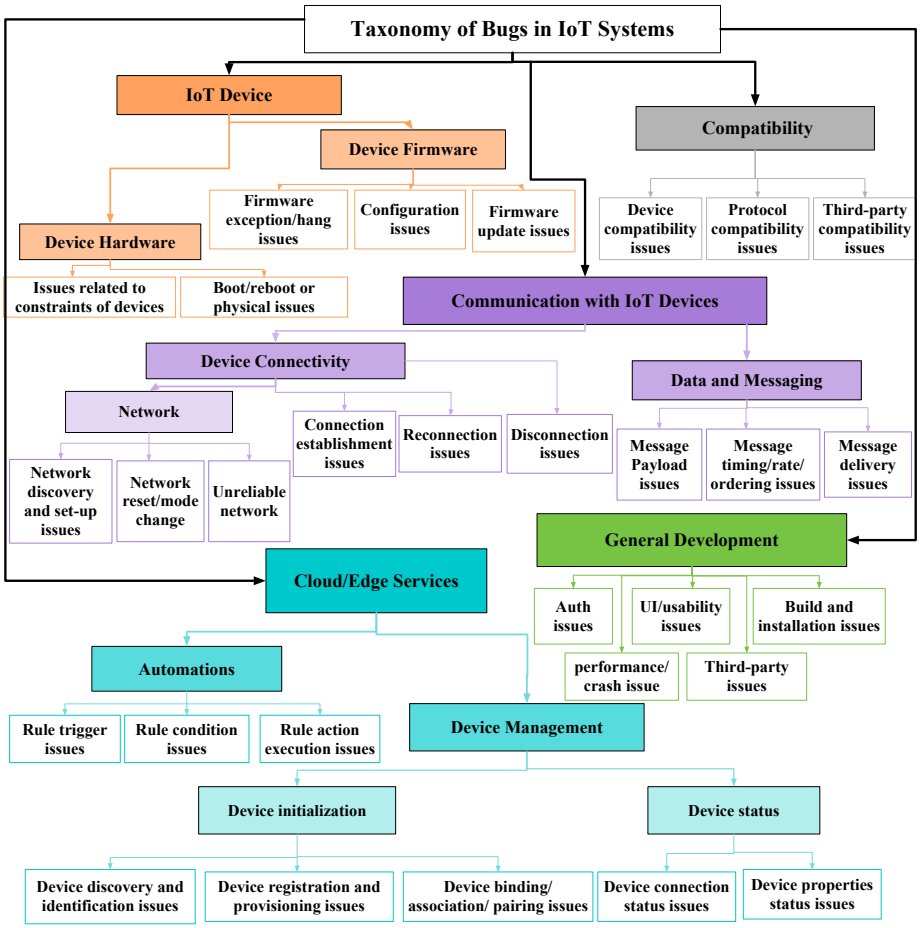

Figure 3.6: Taxonomy of IoT bugs.

In addition to the network discovery, not handling a network reset, or unstable and unreliable networks are other common issues that can lead to failures. However, Sometimes IoT devices fail to establish a valid connection to the gateway or remote cloud servers despite a valid network status. Failure in reconnecting, connection refreshing, and ensuring such connectivity failures do not cause propagated failures in other components are other pitfalls that IoT developers often deal with. Additionally, unexpected disconnection or connection closure issues are other manifestations of bugs in this category.

Listing Listing 3.1 shows a real-world example of the additional tests IoT developers write to check if a fix for a bug caused by WiFi reconnection is a good enough fix. As this example shows, IoT developers have to write code that is able to handle unexpected reconnections and disconnections in order to avoid connectivity bugs.

Additionally, two interviewees (P_6 and P_9) believe that connectivity bugs are the most serious and challenging bugs. As P_9 states "the weakest part of our IoT platform is to communicate with IoT devices."

Data and Messaging: This category includes bugs that are related to data and message sending in the IoT system. Typically, messages are either commands that are sent to IoT devices via the cloud, or they are telemetry data that are received from IoT devices in the edge, cloud, or applications. Some bugs cause failures in delivering these messages from the sender to the receiver. Some other messaging bugs are related to the timing of messages. For instance, various reported bugs are related to the rate and order of the messages. Additionally, some bugs are linked to the payload that is being delivered through the messages.

In some cases, payload size or format are the causes of failures. There are also cases where there are violations of payload integrity by messages being truncated or overwritten. For instance, in HOMIE-ESP8266/309, the IP address is truncated by one character in the MQTT response message.

Cloud/Edge Services This category includes bugs that are related to the services delivered by the remote cloud servers or gateway devices in the edge layer.

Device Management: To monitor and control each IoT device remotely, devices should be connected to a cloud server or a hub device, and report their status while listening to user commands. Device management (DM) issues include prob-

lems that cause failures in this process. The first class of DM issues happens in the stage of initializing the IoT device in the cloud or edge systems. One type of device initialization (DI) issue is when the IoT device is not properly or uniquely identified by the cloud or edge components and therefore causes further failures in the subject IoT system. Besides, if the IoT device fails to provide a recognizable identity and valid permissions to the cloud or edge, it would not be allowed to use remote services. Some examples of device registration and provisioning bugs are duplicate device certificates, issues with auto-provisioned devices, or failure in retrieving data from the provisioning service. Another class of DI bugs is problems with binding, association, and pairing of IoT devices. There are several cases where bugs are introduced to the IoT system just because devices are grouped together (such as devices in one room), due to not properly handling the association of a sensor device with a physical object. There is an example mentioned by one of our interviewees (P_5) where two switches were associated with one lamp and only one switch was working due to issues with addressing multi-instance devices with labels.

The second class of DM issues is related to problems with monitoring the status of IoT devices. One type of device status is the connectivity status, to check whether the device is online, which is also known as the heartbeat check. Some examples of bugs in this type are wrong device heartbeat rate, showing a lost connection as live and vice-versa, or not notifying other components when the device goes offline. Listing 3.2 shows some hearbeat issues that IoT developers have fixed in order to debug unexpected resets of the IoT devices bug occurred in HOMIE-ESP8266/242.

```
1 - There is too much time the client has sent a ping request without a response:
      Disconnect client to avoid half open connections.
2 - The server does not receive messages inside the keep-alive window: Send ping to
      the server to ensure the server will receive at least one message inside
      keep-alive window.
3 - The connection is one-sided and the server might not be connected: Send a ping
      message to the server to verify if the server is still there and the
      connection is not half-way.
```

Listing 3.2: The hearbeat issues related to the bug in HOMIE-ESP8266/242 and the developers' approach to check each of them.

Failure in retrieving the device status such as color and brightness for a light bulb, showing the status incorrectly, or failure in updating the device status are some other examples of DM issues.

Automation: This bug category is related to automation services that IoT cloud or edge platforms provide and it is classified into the trigger, condition, and execution issues. Rule trigger defines a condition under which a rule is initiated. Trigger failures usually cause a rule not to become triggered when it should be (SMARTHOME/5578) or vice-versa (HOME-ASSISTANT-CONFIG/2). Rule condition is the statement that should be checked when the rule is triggered. Examples of the rule condition issues are problems in retrieving the device state to check the rule condition since the condition usually relies on device latest status (TESLA-API/43). Issues in the execution of the rule action are the last and the most prominent automation issues. Some examples of these issues are crash after rule action execution, issues in handling asynchronous behavior and threads in rules, and having problems with the output of the rule being unpredictable or nondeterministic.

General Development This category captures common development bugs. Some common issues are problems with installing, compiling, or building a project as well as unexpected crashes or performance issues in the IoT project. The general development category also includes bugs in the authentication or authorization process. One of the IoT-specific authorization issues are problems with generating, signing, or maintaining the certificates that devices have to present for using cloud or edge services (AZURE-IOT-SDK-C/657). Other sub-categories of general development bugs are UI-related, usability, or external issues.

3.2.2 Root causes of Bugs

Figure 3.7 shows the distribution of bug categories and root causes. In this section we will describe each category of root causes.

General software programming faults (SWP): These faults are the most dominant root causes of the bugs. These faults are often syntactical mistakes by IoT developers. Examples of such faults would be accessing null pointers or empty memory locations, typos, missing/wrong conditions/loops, or missing/wrong values in code, infinite loop, and wrong data type. Also, we consider all UI-related

	SWP	HWP	SEM	CNF	DEP	MEM	CON	EC	TM
Device: Hardware	5	4	1	1	1	2	1	1	5
Device: Firmware	2	4	2	5	6	4	4	1	2
Communication: Connectivity	6	2	3	7	6	3	3	1	11
Communication: Messaging	15	7	10	1	5	3	3	7	11
Cloud: DM	25	3	19	13	7	3	3	8	13
Cloud: Automation	2	0	8	1	0	0	2	0	0
Compatibility	4	1	7	6	6	1	1	2	4
DEV	62	2	10	25	27	8	4	14	5

Figure 3.7: Distribution of bug categories (vertical) and root causes (horizontal)

faults to be in this category, like faults in the front-end code of a web application or an Android or iOS application. The top three issues caused by these faults are messaging issues, device management issues, and general development issues.

Semantic programming faults (SEM): These faults are the second most dominant root causes of IoT bugs. After general development issues, top issues caused by SEM faults are device management issues, automation issues, and compatibility issues. Some semantic mistakes that IoT developers make are wrong control flow, faulty logic of functionalities, or erroneous return values.

Among SEM-related faults, wrong return values are one of the most challenging ones to deal with for developers. One example of faulty return values is the mistake in AZURE-IOT-SDK-C/481, when the developer wants to get a device with a certain ID. If the device ID does not exist, the expected behaviour is to create a new device with that ID and return the newly created device. However, if a device with a certain ID does not exist, the IoT platform mistakenly returns an error code which is a fault in the return value. Also, figureãutoreffig:sem is about bug report SYNSE-SERVER/91, which occurred in an IoT plugin SDK. Developers are struggling to identify the expected behavior of the IoT system when facing empty messages from devices. Their observations suggest that empty messages from devices might sometimes mean that the device is misconfigured. As a result, developers have to check whether the empty message is from a previously configured device or not.

Some SEM faults are related to the automation logic of the IoT system, such as logical faults in automation apps, which are also discussed in recent studies [4]. An example of such mistakes is obvious in ENTITY-CONTROLLER/103, where developers have conflicting opinions on the expected behaviour of IoT devices when an overriding command is sent to devices. This issue gets more serious when the expected behaviour designed by developers as the correct behaviour, does not support certain user scenarios, which is the case in this bug example.

Dependency faults (DEP): The next frequent root cause is DEP faults, where developers use wrong versions of the software or firmware libraries, tools, devices, or protocols. Dependency faults are the main causes of compatibility issues, device firmware issues, and general development issues. It's important to note that dependency faults are different from compatibility bugs in nature since the latter

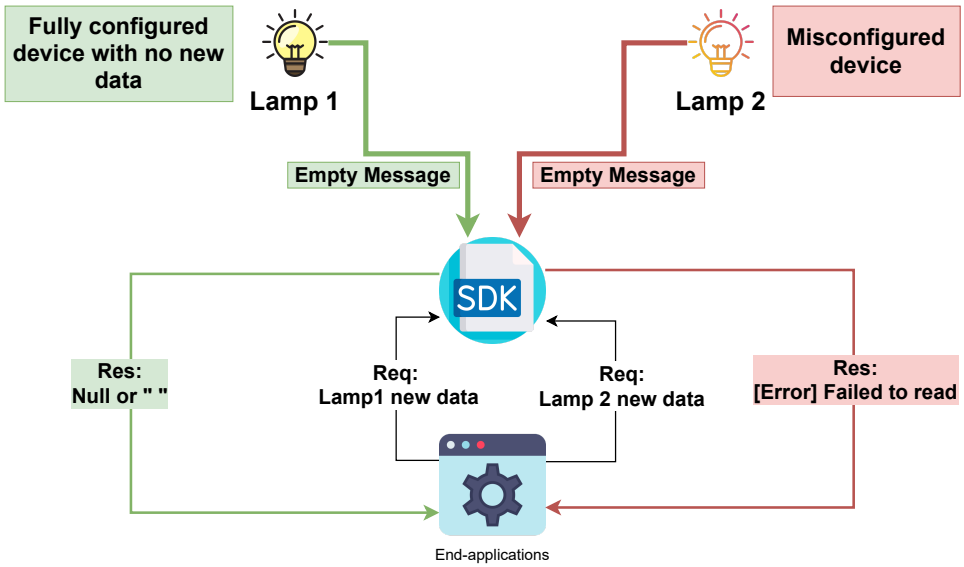

Figure 3.8: The IoT SDK should not treat the empty messages from devices 1 and 2 in the same manner. To avoid making an SEM fault by returning a wrong value to end-applications, the SDK developers have to check whether the requested IoT device has been previously configured or not. This figure shows the correct expected behavior.

is mostly an observed failure that can have different causes (as we can see in 3.7), where the first one is a fault by developers in choosing the right version. Developers have to follow the dependency constraints among hardware devices and software libraries to avoid these faults in their code.

Timing faults (TM): One of the most important root causes often leading to hardware, connectivity, and messaging issues are TM faults. Improper handling of time-outs or rate of operations, wrong time-out values for connection closures, or not handling asynchronous behaviors are among timing-related root causes. In some cases, the connection is monitored to be closed when the device is idle. Furthermore, finding the ideal time-out value for properly closing the connection to avoid possible failures has shown to be a faulty task for IoT developers.

One example of TM faults occurs in DEVICEHIVE-JAVA-SERVER/145 when users want to get a specific command of a certain device and instantly receive an empty array as the response when the device is still busy executing the same com-

mand from before. The expected behaviour is to wait until the device acknowledge the complete command execution within the defined time-out. Figure Figure 3.9 shows part of the bug fix for this issue, that enforces the device to first wait for the command to be processed.

```
@GET
@Path("/{deviceGuid}/command/{commandId}/poll")
@PreAuthorize("hasAnyRole('CLIENT', 'ADMIN', 'KEY') and hasPermission(null, 'GET_DEVICE_COMMAND')")
@ApiOperation(value = "Poll for commands ", notes = "Polls for commands based on provided parameters (long polling)")
@ApiOperation(value = "Waits for a command to be processed.",
        notes = "Waits for a command to be processed.<br>" +
                "<br>" +
                "This method returns a command only if it has been processed by a device.<br>" +
                "<br>" +
                "In the case when command is not processed, the method blocks until device acknowledges command execution.
        response = DeviceCommand.class)
@ApiResponses({
        @ApiResponse(code = 204, message = "Command was not processed during waitTimeout."),
        @ApiResponse(code = 400, message = "Command with commandId was not sent for device with deviceGuid
        @ApiResponse(code = 404, message = "Device or command was not found.")
})
void wait(
        @ApiParam(name = "deviceGuid", value = "Device GUID", required = true)
        @PathParam("deviceGuid")
        String deviceGuid,
        @ApiParam(name = "commandId", value = "Command Id", required = true)
        @PathParam("commandId")
        String commandId,
        @ApiParam(name = "waitTimeout", value = "Wait timeout")
        @ApiParam(name = "waitTimeout", value = "Wait timeout in seconds (default: 30 seconds, maximum: 60 seconds)
```

Figure 3.9: A timing fault while polling device commands occurred in DEVICEHIVE-JAVA-SERVER/145

Hardware programming faults (HWP): Some faults that are more specific to hardware programming such as interrupt handling, are assigned to the category of HWP faults. Examples include wrong pin/port mapping, improper socket operation, wrong bit assignment, and incorrect interrupt handling. Other faults in hardware code resemble those in software code, like referencing a null pointer or wrong return type.

Faults in handling exceptional cases (EC): These faults are another root cause for IoT bugs that include mistakes in handling corner cases (large or out of range data), not handling errors properly, or not handling changes of the requirements or changes in third-party components. Figure Figure 3.10 shows a EC fault in which the IoT developers did not count for large messages.

Memory faults (MEM): Faults related to incorrectly handling memory objects

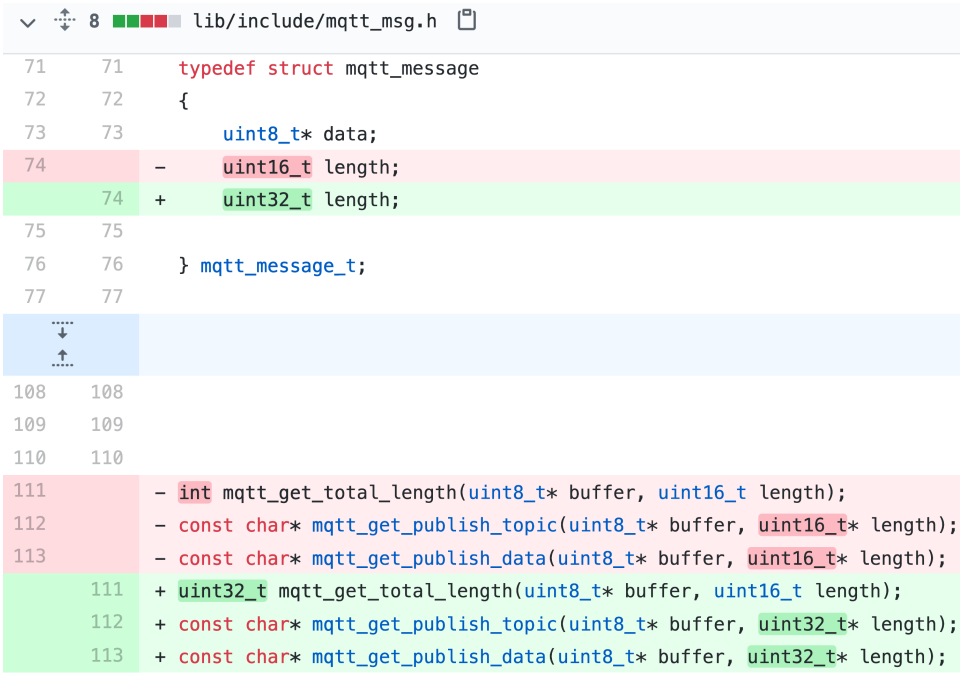

Figure 3.10: An EC fault (from ESP-MQTT/34) that allocates 16 bit instead of 32 bit for each MQTT message, which cause the system to break with large MQTT messages.

fall into this category. There are some previous studies focused specifically on memory-related bugs [39]. Also some other studies have categorized these bugs as a defect in software systems [71]. Memory bugs often happen as the result of race conditions on memory locations. A race condition is a situation in which the result of an operation depends on the interleaving of certain individual operations. Such situations can lead to different types of bugs like buffer overflow, stack smashing, memory leak, uninitialized read, and double free bugs, as it is also classified in bugBench [48].

Concurrency faults (CON): A concurrency fault exists if running two threads at the same time is not handled properly. These faults usually do not show up if the two programs run sequentially. Thus, these bugs happen only in multi-threading or

multi-processes environments and only when the operations are ill-synchronized. In the context of IoT, the communication of things and the cloud can be asynchronous, which means data can be transmitted intermittently rather than in a steady stream, because there are various parties (user, sensor, actuator) that can send data whenever they like. Based on the classification in bugBench, there are three types of concurrency bugs. Data race bugs happen when at least two threads access a shared variable at the same time. At least one thread tries to modify the variable. Atomicity-related bugs happen when one thread is unexpectedly interrupted by another thread. Deadlock is a situation when two or more threads wait for a resource and can never proceed anymore.

Configuration faults (CNF): Hardware devices and development tools need some particular configuration, and wrong provisioning of the proper configurations are considered CNF faults.

3.2.3 Correlations among bug categories

During our analysis, we observed some frequent patterns of certain bug categories appearing together more often. To study the correlations between bug categories, we used Lift [41], a statistical metric introduced by Han and Kamber that computes the probability of two categories appearing together. For each pair of bug category, a lift value of more than 1 shows a positive correlation, and a lift value below 1 reveals a negative correlation.

Table 3.2 shows the lift values of correlated bug category pairs. The top correlated bug categories are [hardware, firmware], [hardware, connectivity], and [firmware, connectivity]. This correlation analysis, besides helping IoT developers in debugging, gives insight into how intertwined IoT bugs can be in practice.

As the Figure Figure 3.11 shows the GitHub discussions for RPIEASY/128, an IoT developer observed a clear conflict between the device status logged from the device layer and the cloud layer. Further investigation revealed that a fault in the device driver code has caused the wrong device battery status to be sent to the IoT cloud.

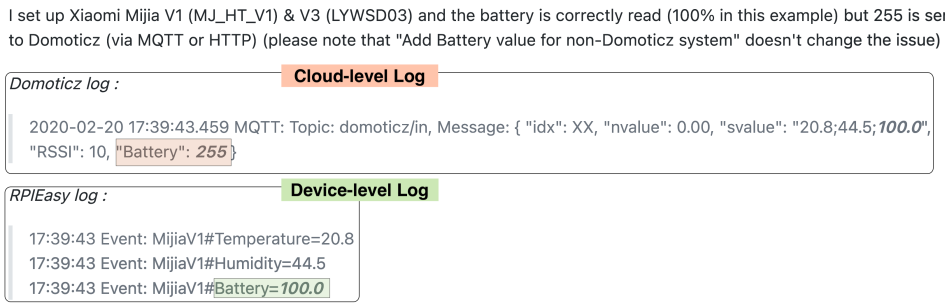

Figure 3.11: An example of how a fault in the device layer has propagated to a failure in the cloud layer

Table 3.2: Bug Categories with Positive Correlation

Bug Category	Bug Category	Lift Value
Device: Hardware	Device: Firmware	3.86
Device: Hardware	Communication: Connectivity	2.57
Device: Firmware	Communication: Connectivity	2.25
Compatibility	Cloud: Device Management	1.84
Compatibility	Communication: Messaging	1.44
Communication: Messaging	Device: Firmware	1.42
Cloud: Device Management	Automation	1.38
Cloud: Device Management	Device: Hardware	1.28
Communication: Connectivity	Communication: Messaging	1.22
Communication: Connectivity	Compatibility	1.14

3.2.4 Representativeness of bug categories

All the bug categories in our taxonomy have been faced by at least 82% of IoT developers, which shows the bug categories are representative of the real-world bugs in IoT systems. More than 97% of IoT developers have faced connectivity issues at least once. After this category, messaging, automation, and device management issues are among the most approved bug categories with all of them approved by more than 90% of IoT developers. Also, more than 81% of IoT developers have the experience of dealing with general development bugs. Regarding sub-categories, about 95% of IoT developers having dealt with device initialization issues. Bugs

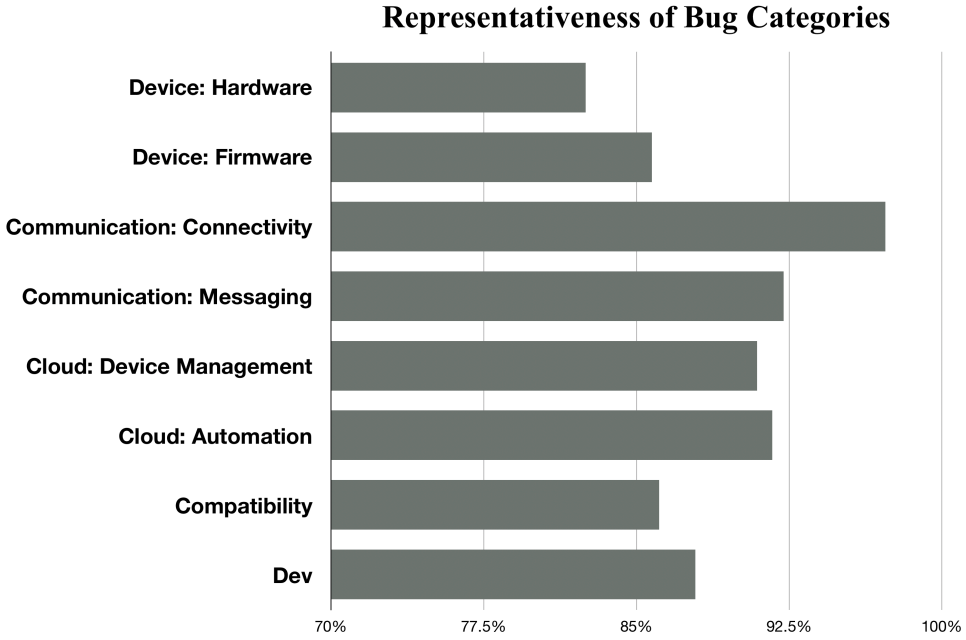

Figure 3.12: Survey result about how much IoT developers have experienced each bug category.

related to authentication and installation are the least experienced general development bugs.

3.2.5 Frequency of bug categories

We asked our survey participants how frequently they face each bug (sub)category. The most frequent categories of bugs are general development issues (48%), device management issues (29%), and messaging issues (19%). Connectivity issues are the most frequent bug category with more than half of developers face it frequently. Hardware, and messaging issues are among the most frequent bug categories after connectivity issues. The compatibility issues are the least frequent bugs according to IoT developers' experiences. Regarding sub-categories, device initialization issues are the most frequent device management bugs. Also concerning the device status issues, bugs related to the status of connection have shown more frequency. Among device-related issues, bugs that are related to the constraints of IoT devices

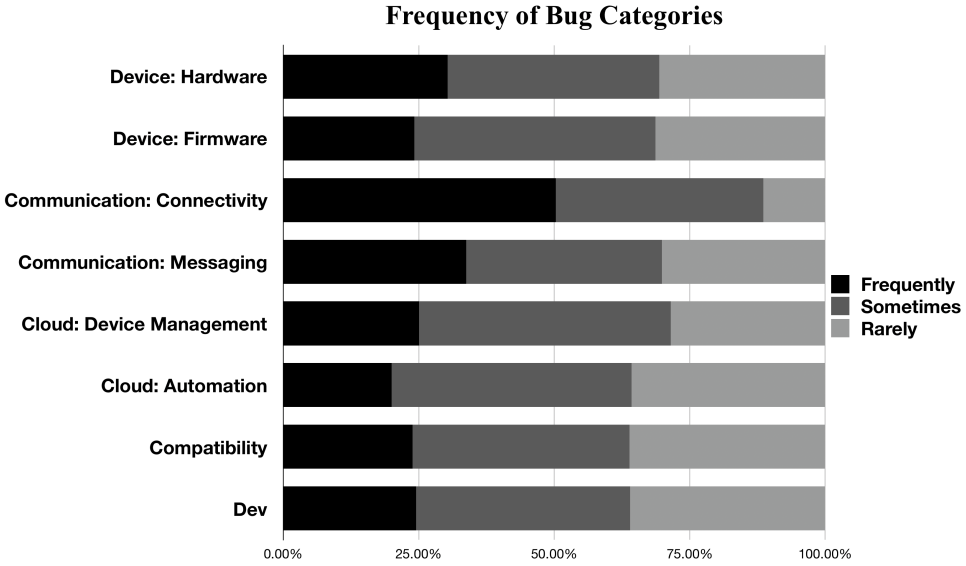

Figure 3.13: Survey result about how frequent each bug category is based on IoT developers' experience.

have the most frequency. Concerning general development issues, installation issues are the most frequent bugs.

3.2.6 Severity of bugs

We also asked survey participants what are the bug categories' perceived severity based on the impact they have on the IoT system and their fixing-time. Connectivity issues are the most severe bug category as well since more than 62% of IoT developers find it severe. Although device-related issues are the least experienced bug categories, they are the most severe bugs after connectivity issues. According to the survey respondents, automation issues are the least severe bugs Among sub-categories, device initialization issues has been found severe by more than half of IoT developers. Among device-related issues, bugs related to the status of device properties are more severe based on survey respondents. The device firmware exception issues are the most severe device-related bugs. Concerning general development issues, installation issues are the most severe bugs. Finally, nearly 30% of survey respondents find general development bugs severe.

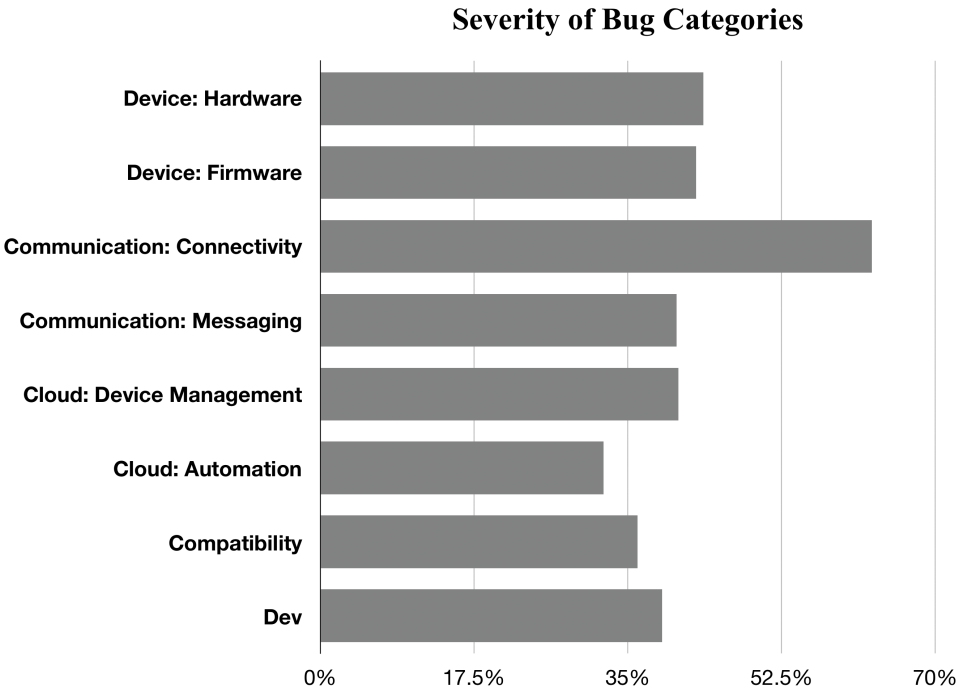

Figure 3.14: Survey result about how severe each bug category is based on IoT developers' experience.

Table 3.3 summarizes all the quantitative results we gathered from our online survey.

Taxonomy augmentation Regarding IoT bugs, we collected 79 tags from interviews and 18 tags from the survey comments. Device binding issues, performance issues, and third-party compatibility issues were not discovered before the interviews and were added by interviewers' experience. The survey comments did not reveal any new information to be added to the taxonomy. However, the extracted tags, from both the interviews and survey, helped us to characterize each bug category by providing contextual data.

Table 3.3: Survey Results: Bug Taxonomy

Bug Category	**Have faced**	**Frequency**			**Is Severe**
		Frequently	Sometimes	Rarely	
Device: Hardware	82.50%	30.30%	39.06%	30.64%	43.61%
Device: Firmware	85.75%	24.20%	44.48%	31.32%	42.78%
Communication: Connectivity	97.22%	50.29%	38.29%	11.42%	62.78%
Communication: Messaging	92.22%	33.74%	36.14%	30.12%	40.56%
Cloud: Device Management	90.93%	25.06%	46.43%	28.51%	40.75%
Cloud: Automation	91.67%	20.00%	44.24%	35.76%	32.22%
Compatibility	86.11%	23.88%	40.00%	36.12%	36.11%
Dev	87.89%	24.53%	39.44%	36.03%	38.89%

Chapter 4

Challenges

4.1 Methodology

In this study, we aim to characterize the development challenges of IoT developers in a systematic way. More precisely, we want to answer the following research questions:

- `RQ1`: What are the types of development challenges IoT developers face?
- `RQ2`: How frequent each of these development challenge are based on IoT developers' opinions?
- `RQ3`: What development challenges require more attention from the research community and the industry?

To answer these RQs, we devised the bug discussion analysis, IoT developer interviews and survey that we discussed in chapter 3 to characterize the development challenges in the context of IoT.

4.1.1 GitHub Issues Discussions

During the categorization of IoT bugs, we also read the entire GitHub discussion for each bug report and kept track of the challenges IoT developers are dealing with. We used the same tagging technique discussed in section Figure 3.1.2 for analyzing challenges we observe while reading through GitHub discussions. Our goal

was to come up with some examples and initial high-level categories of IoT development challenges . We aimed to use these initial data to lead the semi-structured interviews with IoT developers which will be discussed in the next section.

4.1.2 Interviews

During the interviews discussed in section subsection 3.1.2, we included open-ended questions about the challenges of IoT development as well. Although we had already tagged some possible challenges while reading through GitHub discussions during bug labeling, we employed these interviews to add new aspects of IoT development challenges to our data. Our semi-structured interviews helped us avoid being restricted to only code-level challenges and capture developers' high-level problems too. We could collect a more comprehensive set of IoT development challenges with inputs from developers of different IoT backgrounds during interviews. The participants, protocol, and analysis of interviews for the challenges section of interviews, was mostly the same as for the bugs section. The only major difference was that as included more open-ended questions about IoT development challenges.

By using the categories we extracted from interviews and the examples we collected during bug labelling, we already had enough data to form the initial categories of IoT development challenges utilizing the protocol discussed in section subsection 3.1.2.

4.1.3 Survey

We employed the survey discussed in section subsection 3.1.3 to validate the categories of challenges we had with a larger number of IoT developers. We also could collect more insights and examples for each category of IoT development challenges, which helped us to characterize them more comprehensively.

4.2 Findings

In this section, we provide our findings regarding the challenges faced by IoT developers.

4.2.1 Testing and Debugging Challenges

Relying on access to the real device According to seven interviewees, several GitHub issues (DEVICE-OS/1871, MYCONTROLLER/485, TESLA-API/43), and 74% of survey participants, IoT developers rely on access to devices to test and debug their IoT system, by tasks such as manual reset or device output monitoring ($P_{2,3,7}$).

```
// Develoeprs need to flash a real device in order to test the fix
1- Flash LTE device with this system firmware and tinker
//Manual digital write to the physical device
2- Make a function call particle call <device> digitalwrite D7,LOW
//Manual investigation of the device output
3- It should return 1 if successful
4- Wait 3 minutes
//repeat the scenario again
5- Make a function call particle call <device> digitalwrite D7,LOW
6- It should return 1 if successful
```

Listing 4.1: Steps to test a variable lag bug in DEVICE-OS/1567

Three interviewees (P_2, P_3, P_7) mentioned that in some scenarios, they need access to the physical hardware to reset the device or investigate the output of the device for debugging.

For instance, as it is shown in Listing 4.1, to test a fix for a variable lag related in an IoT device, developers need to do manual digital writes to the IoT device after some time intervals, and also investigate the output. GitHub discussions show that IoT developers go through these steps manually.

Another example in the same project is show in Listing 4.2, where the IoT developers have to apply physical changes to the device, like plugging or unplugging the battery, and also observe the physical state of the device in order to make sure that the fix completely fixes the bug.

```
// Develoeprs need battery-powered devices in order to test the fix
1- Flash the example application to a battery powered device
//Developers need to manually change the physical state of the device
2- Disconnect and reconnect the USB cable
//Manual investigation of the physical state of the device
3- Observe the LED status, whether it will enter SOS mode
```

Listing 4.2: Steps to test a USBSerial SOS issue in DEVICE-OS/1707

This challenge is also observable in other bug reports in GitHub (DEVICE-OS/1871, MYCONTROLLER/485, HOMIE-ESP8266/242). One example is a developer saying "*I need to check on my car to see what is returned by the API*" in discussions for TESLA-API/43. In some scenarios, devices are out of reach or in hard-to-access locations thus making remote debugging more essential. Four interviewees believed that practical simulation solutions are required for better IoT testing and debugging. As P_5, a middleware developer stated "*IoT device vendors do not provide a mock of their devices, and we have to do reverse engineering on the actual hardware devices rather than working with the simulated ones.*" Also as $P_{5,8,9}$ stated, current simulation solutions in IoT are not mature enough and they are only valid for limited scenarios, such as testing high-level controllers or small unit tests, rather than being suitable for all levels of testing such as system testing. For instance, P_8 states "*Device simulation can only help in either testing high-level controllers or small unit tests rather than helping with testing all functionalities of your system in integration with a Google NEST device as an example.*" In general, as $P_{5,9}$ also mentioned, compared to developing software within other ecosystems such as web platforms, where you can simulate server and browser in your local environment, simulation in IoT is not mature yet. Some relevant challenges mentioned in the survey are *affording to have all types of IoT devices*, *complex custom logic for effective IoT device mocking and simulation*, and *setting up test environments with IoT devices*.

Fault localization According to eight interviewees, nine survey comments, and also half of the survey participants, fault localization is a barrier due to lack of transparency in the operations of IoT systems. As P_7 mentioned "*there is no environment that logs everything.*" One contributing factor to this is the difficulty of tracing executions of numerous external components in IoT systems. P_3 mentioned using open-source solutions just to be able to log everything.

Another factor that impacts fault localization is the existence of hidden failures. As an example, P_7 mentioned "*It's hard to recognize on the app that the temperature the device is reporting now is for several minutes ago.*" Also P_5 states "*Usually, the bug is inconsequential until it is perceived at the end of the chain in the user interface.*"

Moreover, $P_{2,4}$ mentioned examples of failures that only show up after the de-

vice has worked for a specific amount of time (five minutes for P_2, several hours for P_4), which is also observed in GitHub issues (DEVICE-OS/1926, ZWAVE2MQTT/141, VSCP/207). This issue makes IoT failures more unpredictable and may hide developers' faults. Another issue toward fault localization is the lack of tools and developers' support. For instance, P_3 inspects device messages in bit-level by monitoring communications using Wireshark. P_2, as a hardware platform developer, said "*Since there is no feedback of errors or corruptions from devices, we've added some LEDs to them to track if something is working in the device level or not.*"

According to the survey results, more than half of the IoT developers agree with the lack of transparency as a challenge for debugging IoT, and nine IoT developers mentioned fault localization as one of the main IoT developers' challenges in the survey comments.

Reproducing IoT bugs In addition to observing several GitHub discussions (DITTO/414, TESLA-API/68), we collected four tags from interviews, and three survey comments regarding the challenge of reproducing IoT bugs. Besides the already mentioned factors which harden bug reproduction, such as limited access to devices or hidden failures, some bugs only happen with a specific device setting or with certain environments of the IoT system. IoT developers cannot reproduce these bugs unless they have exactly the same setting or environment. For instance, a survey comment indicated "*It's hard to reproduce some memory-related bugs in X86 devices when they have ASLR enabled.*" Also, we observed other examples such as TEMPERATURE-MACHINE/13: "*I can reproduce the bug with the help of ice packs taken at three different temperatures from my freezer.*"

Combinatorial explosion In addition to all the evolving components in traditional software, such as libraries and operating systems, there are more changing factors in IoT systems. Hardware devices produced by various manufacturers with different standards, device integration middlewares, and communication protocols are some examples of these extra changing factors in IoT. With all these components releasing new versions at a specific rate, a combinatorial explosion problem is likely to happen when developers want to cover all possible combinations with test cases. One relevant statement is "*I do not have all kinds of bulbs, remotes, and sensors, so I could be completely wrong!*" in a GitHub discussion (PYTRAD-

FRI/135). We could collect eight tags come from four interviewees and two tags from survey comments regarding the challenge of combinatorial testing. As one example, P_9 said "*We have to test with 10 or 15 different devices each time.*" Also, 80 percent of survey respondents agree with the combinatorial explosion as a testing challenge for IoT developers, and P_8 mentioned it as the most severe testing challenge.

Listing 4.3 shows how presence of physical devices can increase the test search space. In addition to regular tests that all systems have, IoT developers have to spend some additional time on writing and executing tests that cover interaction scenarios with physical devices.

```
1 - State changes are sane when the device is powered with USB cable in.
2 - State changes are sane when the device is powered with no USB cable in (
    powered through VIN).
3 - State changes are sane when USB cable is unplugged.
4 - State changes are sane when USB cable is plugged back.
5 - State changes are sane when Serial.end() is called and cable is unplugged and
    re-plugged.
6 - State changes are sane once Serial.begin() is called after.
7 - State changes are sane once Serial.end() is called, cable is unplugged, Serial
    .begin() is called, and cable is plugged back.
```

Listing 4.3: All different scenarios developers have to check, just to test a fix provided for a USB messaging bug in DEVICE-OS/1871

Testing and debugging edge-cases Covering large-scale scenarios (e.g. too many devices) and exceptional cases (e.g. temperatures below zero) add to the test coverage obstacles. This challenge is mentioned by four interviewees, three survey participants, and observed in several GitHub discussions (DEVICE-OS/1926, TEMPERATURE-MACHINE/13). As one example, P_4 said "*We should put effort to write proper tests against concurrency issues since we should be able to handle 140,000 HTTP requests per second because our IoT system is deployed in different cities.*" Additionally, this challenge is the most experienced testing challenge (83% of respondents).

Immature testing culture Figure 4.1 shows an over-reliant on IoT developers for testing as 64% of participants mentioned developers are the main testers in their IoT project. After developers, the main responsibility of testing is for QA teams (24%), third-party testing services (9%), and clients (1%). Furthermore, two% of

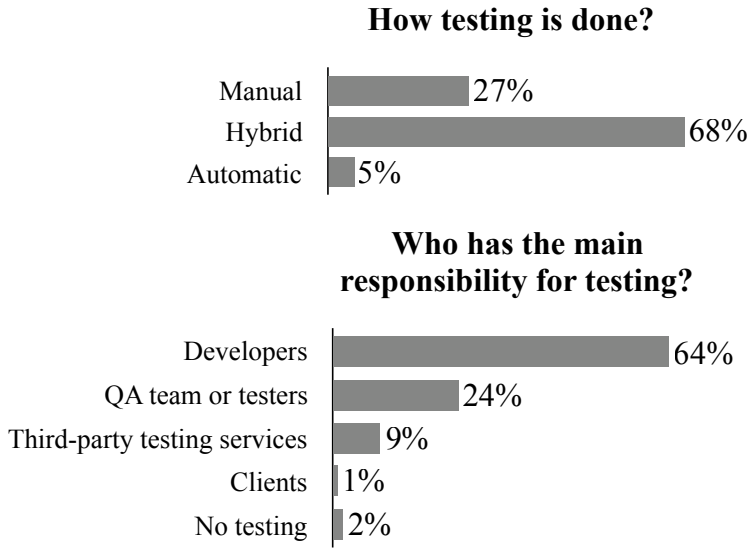

Figure 4.1: Survey responses about how testing is done in IoT projects.

participants have reported that they don'y have sophisticated testing.

Specially in the open-source community, developers rely on clients, who have a more diverse set of devices, for testing and debugging (ZWAVE2MQTT/141). We witnessed several examples of asking the users who reported the bug to test with a certain devices and report it back to the developers, since the developers don't have the actual devices. There are several quotes during interviews from open-source developers such as "*The platform is well-tested from the software side and usually, the bugs are because of unknown hardware issues.*" or another open-source IoT developer that mentioned "*We don't know what practices are good for hardware testing and lots of bugs are discovered by the community.*"

As P_6, a developer of a popular IoT project with near 7K stars, stated "*We do not have a QA team. it's up to developers to do testing, either manually or writing automated tests.*" Often, software developers do not have the skills to test the hardware side. P_9, a software developer of an IoT platform with 1.5K stars, told that the bottle-neck of their IoT platform is testing the hardware side since they do not have sufficient knowledge for tools and practices of hardware testing.

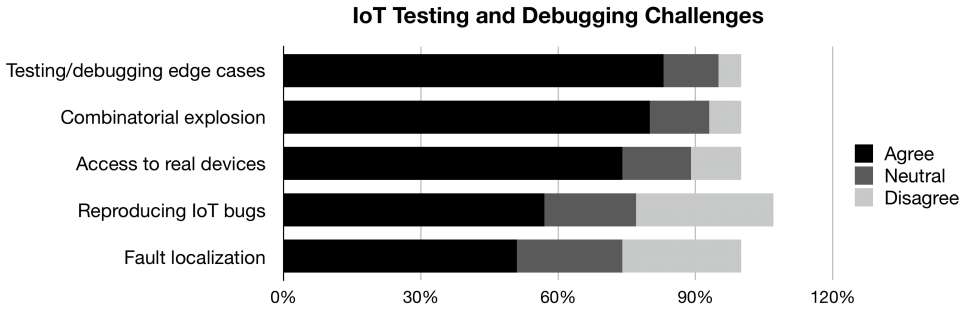

Figure 4.2: Survey responses about challenges of testing and debugging in IoT projects.

As Figure 4.1 shows, IoT testing highly depends on manual tasks as only 5% of participants reported testing completely automatic. Also, during interviews, four interviewees mentioned manual approaches for IoT testing. According to the survey respondents, the most adopted IoT testing approach is hybrid strategies. An example of such an approach is described by one IoT developer "*Services which don't interact with devices directly are tested automatically, but checking the entire platform with devices requires manual testing.*"

Based on our survey results, only 5% IoT developers use complete automated testing. Around 27% of IoT developers rely on complete manual testing while the rest (68%) use a hybrid approach.

4.2.2 Heterogeneity

Device and protocol fragmentation Some of the IoT developers reported developing separately for each device or protocol in order to fulfill interoperability ($P_{2\text{-}5}$, P_8). For instance, P_3 stated he has to develop a distinct adapter for talking with each particular device. He mentioned *"There is no guarantee that something that works with brand A also works with brand B."* On the other hand, P_6 noted that their platform is restricted to certain protocols instead of devices. Other developers mentioned fragmentation challenges by pointing out *fragmentation on the same platform* and *time to implement new technologies.* In addition, the majority of the interviewees (seven out of nine), 11 survey comments, and near half of the survey respondents find integration with a new IoT device or communication protocol

challenging.

The fragmentation issue can also be perceived by looking through GitHub issues. For instance, AZURE-IOT-SDK-C/815 happens due to minor timing differences between MQTT and AMQP protocols, and AZURE-IOT-SDK-C/213 is caused by different reliability of communication protocols as a developer says "*I've never faced any issue with Ethernet connectivity, since its very stable, but WiFi and Cellular face so many issues.*"

Third-party breaking changes Third-party changes challenge is mentioned by all interviewees (23 tags) and is agreed by 63% of survey participants and also there are several comments in the survey about it (eight tags). Three interviewees stated that third-parties make breaking changes without prior notice. Also, $P_{5,8}$ mentioned examples where the third-party system stopped supporting a device or a service which caused breakage in their IoT system. Four interviewees ($P_{2,4,5,8}$) explicitly mentioned that it's hard to keep pace with all the rapid changes from various third-parties such as device manufacturers.

Diversity of technologies, backgrounds, and requirements Challenges posed by the fundamental diversity of IoT technologies are the most repeated challenges among both interviews (30 tags) and survey comments (25 tags). Several participants mentioned that IoT development requires diverse development skills such as hardware programming and knowledge in dealing with network protocols.

Commonly, developers do not go through this learning curve: "*developers tend to use protocols which they are familiar with but sometimes better solutions exist and developers do not know/use them.*"

60% of IoT developers agreed that it's challenging for them to learn and interpret different technologies and network protocols to be able to develop an IoT system. One interesting theme emerged from interviews and the survey is making hardware and software developers with diverse backgrounds and skills work together ($P_{2\text{-}3,9}$), and nine survey comments). One challenge that is mentioned by participants is making embedded developers and cloud developers work independently with each other. The consequences of inadequate backgrounds of developers are best described in another comment as it says "*It causes a large number of devices to end up with security flaws, or the inability to deal with disconnections.*"

$P_{2,3,7,8}$ and several survey comments mentioned that it is hard to understand

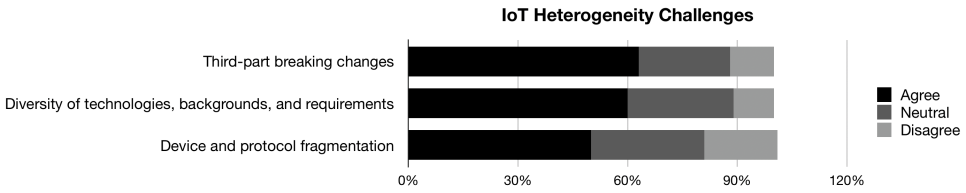

Figure 4.3: Survey responses about challenges related to hetereogenity in IoT projects.

low-quality documentation of certain device manufacturers and interpret complex response payloads from particular devices. $P_{2,3}$ and two survey comments also mentioned that user requirements, as well as users' backgrounds and skills, can be very disparate and it's challenging to develop a generalized IoT system that can support all possible use cases. For instance, P_2 mentioned that they had to include more pins on their hardware and add support for obscured sensors to cover all user requirements. Other challenges are *large search-space for selecting compatible devices or libraries* ($P_{5,7,8}$), and *dealing with diverse regulations and standards* ($P_{5,8}$, and three survey comments).

4.2.3 IoT security challenges

Also, from a total of 14 participants who mentioned security-related challenges, six of them posed it as the most important challenge. Below we will describe the security challenges aspect of IoT development that we could gather from our survey.

Complexity of IoT security First of all, 66% of IoT developers find security a complicated task. Our interview participants mentioned security issues rooted in the device firmware ($P_{1,3,4}$), network protocols ($P_{7,8}$), and automation rules (P_6). Another emerged theme from our data is related to the challenge of end-to-end security, from the IoT device to the cloud. Other challenges mentioned are *the complexity of the certification process*, *supporting different use cases while following security protocols*, and *existence of various attack surfaces*.

Handling device-level security One of the main challenges, also mentioned by P_7, is generating and storing access tokens within IoT devices that have processing and storage limitations. Similarly, near 60% of IoT developers think that

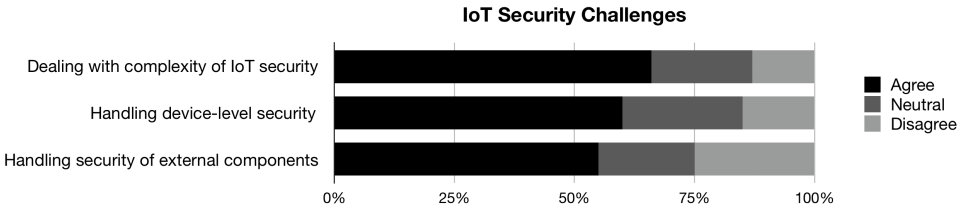

Figure 4.4: Survey responses about IoT security challenges.

device constraints make security tasks challenging. Some ($P_{8,9}$) believe that the security of the local communication between the device and IoT gateway is usually underestimated while it can be highly insecure. As P_9 argued, "*to make the development of the IoT system faster, developers don't consider the security of the local network.*"

Handling security of external components As the survey results suggest, more than half of the IoT developers are not confident about the security of the third-party components, such as operating systems and libraries, used in their IoT system.

4.2.4 Other challenges

Releasing updates for IoT devices Half of the interviewees believe that releasing software upgrades or security patches for already shipped devices ($P_{5,8}$) is inevitably challenging. Six IoT developers in the survey made comments such as *getting critical updates installed on already sold devices*" or *firmware updates in large deployments* regarding update challenges.

Programming for constrained devices 63% of the participants agreed that device constraints make IoT development harder. Most IoT developers struggle to design and implement software in a way to consume less processing power and energy. Device limitations in different layers have also been mentioned by our interviewees ($P_{2,3,6,8}$).

The listing Listing 4.2, which is already discussed in testing challenges, is also pointing out the obstacles caused by device constraints. This bug is an example of how IoT developers should be ready for different physical states of the devices' battery (plugged, unplugged), and consider all scenarios when writing code.

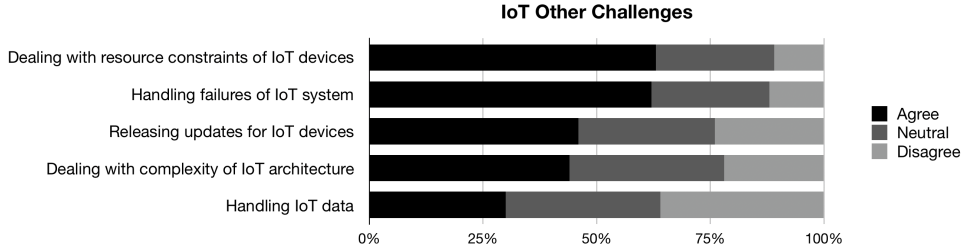

Figure 4.5: Survey responses about other IoT development challenges.

Handling failures An interesting theme that 62% of our participants agreed with is the challenge of handling failures in IoT systems, in a way to avoid losing data and making the system unavailable. As $P_{4,5}$ and five IoT developers from the survey described, developers have to design the system to be tolerable to failures and data losses. *Handling a backlog of sensor data in gateways or constraint devices in case of disconnections* (P_6, ZWAVE2MQTT/141), and *reducing mean-time-to-repair (MTTR) on already shipped devices* are some of the mentioned reliability challenges.

4.3 Discussion

IoT testing solutions are not adopted in practice Various IoT testing tools and methods [20] [60] have been proposed in the literature, such as device simulators [35] and emulators [47], IoT unit testing frameworks [42, 54], and IoT testbeds [2]. However, none of them seems to be adopted by IoT developers as only 9% of them mentioned using third-party services as their main testing approach. Besides, although IoT test automation frameworks exist [35], IoT testing is still carried out in a manual and ad-hoc manner, as 95% of the IoT developers in our study perform manual testing practices. Also, as it is mentioned by $P_{5,8}$, device simulation does not support simulating all types of devices. One possible future direction is having device simulators and emulators specifically crafted for each IoT device individually to virtualize their characteristics and bypass the need for the presence of the actual hardware device during testing. Also, as the importance of combinatorial testing in the context of IoT has been discussed previously in the literature [77], more focus is needed on combinatorial testing tools that consider the heterogeneous

nature of IoT devices and protocols.

Lack of device-level monitoring tool support Investigating the log data of IoT devices is a common debugging task for IoT developers. This task becomes even more important as the device status issues are among the most frequent bug categories. This bug category has appeared in around half of the bug reports in our dataset, and most IoT developers reported that they need to log communications or internal executions of the device as part of the debugging process for these bugs ($P_{1,2,3,4,7}$). There is no universal tool that receives log data from all types of devices, and developers often have to manually employ naive approaches to monitor device status and communications, such as serial print for each device separately ($P_{2,7}$) or using general-purpose tools like Wireshark ($P_{3,7}$). Existing logging solutions to track devices are believed to be inefficient as their limitations were discussed by several IoT developers. One IoT developer best mentioned it as "*even if some devices provide log libraries and tools, they should be manually aggregated or traced from each component separately to track an issue.*"

Fragmented and ever-changing ecosystem of IoT One of the most serious challenges of IoT development nowadays is the rapid obsolescence of hardware devices. As several IoT experts and blog posts [26, 67] describe it, the pace that IoT devices get obscured and stop being supported by the providers is increasing. New updates for IoT devices often make the older devices unusable while they also break IoT developers' implementations. Within this ever-changing ecosystem of IoT, developers have to struggle with maintaining their device-specific or protocol-specific code. IoT developers, not only have to afford all versions of devices to keep up with these changes but also they have to allocate much of their development effort into migrating from one version or ecosystem to the other. As this issue targets both IoT consumers and developers, in 2019, some countries put regulations on the minimum time that IoT providers can release updates after the device is bought [74]. Furthermore, some solutions such as contract-based testing were suggested by interviewees (P_5), to ensure continuous compatibility with third-party systems. However, none of these methods can be a long-term and universal solution as they are still dependent on the contracts and regulations in place.

4.3.1 Threats to Validity

Internal validity One internal threat to the validity of our study, similar to most qualitative studies, is researchers' bias in coding qualitative data. We mitigated this risk by having all authors of the paper involved in the tagging process and discuss any discrepancies in tags for all pieces of qualitative data from various sources (bug reports, interview transcripts, survey comments). For interviews specifically, we eliminated any personal preference on our results by triangulation; each piece of meaningful data from interview transcripts was tagged by one researcher who conducted the interview and one who was not present in the interview session.

External validity One external threat to the validity of our study is the generalization of studied IoT repositories. We minimized this issue by studying a large number of repositories (91 repositories) selected from all layers of IoT systems. Another risk to the validation of our study is the interview and survey participants not being representative of all IoT developers. However, we minimized this risk by recruiting interview and survey participants with different IoT-related field of expertise, years of experience, companies, and domains. In addition, our survey is filled out by 194 IoT developers with a diverse distribution of skills and experiences.

All our study material, including the bug dataset and interview and survey questions, is available online [49].

Chapter 5

Related Work

5.1 Related Work

Bugs and challenges of IoT systems Although a few previous studies have acknowledged some categories of bugs in IoT systems [14, 33, 45], no study is concerned about categorizing all types of real bugs in IoT systems using a systematic approach.

Liang et al. [45] classify performance and security bugs in IoT operating systems. They inspected bug databases of the top three IoT operating systems and classify performance and security bugs into three categories based on 23 bug samples. They indicate that security or performance bugs are either abusing storage or abusing CPU time in the OS. This paper is more concerned about the hardware bugs that cause resource waste or memory failures. Based on their results, root causes of these certain types of bugs in IoT operating systems are misuse of memory-related API in hardware code, lacking memory management, code redundancy, and logic defects. This paper only focuses on bugs in IoT operating systems and their categories are more related to hardware resources and devices' constraints and thus are not representative of all types of bugs in IoT.

Another example is a study by Zhou et al. about IoT logic bugs [84]. In this paper, they do a review of bugs related to the design logic of IoT systems by classifying them into seven categories and presents attack scenarios and cause analysis for each group. Their result indicates that some vulnerability bugs are

inherited from traditional computing systems. While constrained resources in IoT devices and integration of diverse entities from different layers of the IoT system architecture are reasons for some other logic bugs. Similarly, this study is not generalizable to all types of IoT bugs and is more specific to IoT security flaws.

In another study [33], Hnat et al. deployed over 350 sensors across 20 homes for over 8 months and observed a variety of failures. One example of such failures is process failures, due to plug disconnections, hardware failures, and driver crashes. Another failure they observed is link loss between sensors and hubs which are mostly because of concrete slab flooring, copper siding, and radio interference. Furthermore, sensor failures were observed caused by battery drain and plug disconnections. In general, a process downtime of up to 14% was observed during their experiments, whereas sensor-process link loss and sensor failures occurred for 1-2% of the time.

In a recent 2020 study [19], certain peculiarities of open-source IoT repositories were analyzed via examining how developers contribute to IoT repositories. This study compares the development of 30 IoT and non-IoT repositories via factors like programming languages, specialization of contributors, the evolution of files, and the number of dependencies they have. They conclude that software development in IoT is different and it requires a diverse set of skills in various areas. However, this study does not consider bugs and experiences of IoT developers to reach conclusions about the characteristics of IoT development.

A growing body of literature has investigated issues and design flaws that cause safety and security violations in IoT systems as well as security challenges in IoT [55, 80]. More specifically, in smart home ecosystem, security bugs related to the device firmware [32, 46, 57], communication protocols [21, 24, 61], smart apps, and safety of their interactions [4, 10, 11], as well as interactions of different components of IoT systems [85] have been studied. There exist taxonomies for describing characteristics of IoT systems with respect to security and privacy concerns [5] [13]. However, these papers do not present their taxonomy construction process, and they are focused on a different goal, namely security requirements and attacks.

Also several studies have investigated challenges of testing IoT systems [3, 20, 62, 77]. Various solutions for IoT testing have been proposed based on e.g.,

model-based testing [3], IoT mutation operators and test event generators [28, 29], and testing tools [20]. Also, tools and methodologies have been proposed to aid IoT developers in developing of IoT systems [18, 43, 52]. Morin et al. [52] poses heterogeneity as a significant hurdle for developers and proposes ThingML, a modelling language for IoT that is aligned with UML in addition to a code generator [30], to allow rapid development of IoT systems by abstracting the diverse technologies in IoT and help the development of distinct IoT components in a platform-independent approach. There have been some other studies toward model-driven IoT development and methodological frameworks for these systems to assist IoT developers [59].

The challenges of developing IoT systems have been discussed from different perspectives [15, 33, 68]. A previous study investigated the challenges of novice IoT developers to see what development tasks are more challenging for them [17] and developed a tool to help the novice developers [18]. However, no study has tried to study IoT developers' challenges systematically by interviewing and surveying IoT practitioners in the field.

Bug mining and developers' challenges Although mining IoT repositories has not received any attention in the literature, a growing number of studies have employed mining of software repositories or issue trackers to characterize bugs in Machine Learning systems [36, 37, 81, 82]. For instance, in a 2020 study [37], Jahangirova et al. mined DL repositories in GitHub and manually classified 271 issues and PR and 311 commits plus 477 StackOverflow discussions to classify faults in projects that use popular deep learning frameworks. They also did interviews and a validation survey, with both researchers and practitioners, to finally come up with a taxonomy of faults in using deep learning frameworks.

Additionally, [36] accompanies the same goal by investigating 2716 StackOverflow posts and 500 bug fix commits in GitHub. They classify DL bug types, root causes, and impacts as well as analysis on the stage the bug happens. They also give insight toward bugs commonality and trends along the time. There are further empirical investigations conducted to study bugs in DL or ML systems by analyzing bugs in terms of their fix-time and severity[69] [72] [82].

One prior study has followed this approach to identify bug categories in Blockchain systems [79]. In this study, Wan et al. used mining software repositories to classify

bugs in blockchain systems [79] by manually selecting 8 blockchain repositories and classifying 1108 bug reports into different categories. Same goals have been achieved by various studies using the same methodology for Big Data computing platforms [83] in which authors manually inspected 210 quality issues and classified the failures and hardware faults that cause them. Other studies towards the same goal using similar approaches are in the context of web applications [58] and service compositions [12]. In addition, developers' challenges have been investigated in different contexts such as mobile app development [40], and Blockchain development [86].

Chapter 6

Conclusions and Future Work

In this paper, we proposed the first bug taxonomy for IoT systems. We also presented a series of categories of challenges in these systems with a qualitative study. Our findings can help both researchers and practitioners in understanding real-world pain-points of IoT development in the wild and designing new techniques and tools. Our findings shed light on the most frequent and severe IoT bugs, their correlations, and their root causes, and thereby allowing these faults to be avoided or detected early in the development of IoT systems.

6.1 Future Work

Using our bug taxonomy for IoT fault seeding. Our bug taxonomy can be used by IoT mutation operators and test event generators as there have been recent studies regarding using these methods in IoT as well [28, 29]. Since we have characterized both the low-level programming errors and high-level system failures, as well as the location of the faults, our bug taxonomy is highly adaptable to seed various types of faults in IoT systems. We believe that the seeded faults using our bug taxonomy can be perfectly crafted to capture IoT developers' mistakes within various components as our taxonomy is constructed using real-world bugs in IoT projects.

Compared to the fault handling in other computing platforms such as distributed systems and cloud services, IoT systems raise unique challenges [27]. Due

to the heterogeneity of IoT devices, various devices often require different fault-handling techniques executed [56].

Listings

Bibliography

[1] *Classifying your repository with topics*. https://help.github.com/en/github/administering-a-repository/classifying-your-repository-with-topics. → page 12

[2] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. 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. → page 49

[3] A. Ahmad, F. Bouquet, E. Fourneret, F. Le Gall, and B. Legeard. Model-based testing as a service for IoT platforms. In *International Symposium on Leveraging Applications of Formal Methods*, pages 727–742. Springer, 2016. → pages 53, 54

[4] M. Alhanahnah, C. Stevens, and H. Bagheri. Scalable analysis of interaction threats in IoT systems. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 272–285, 2020. → pages 28, 53

[5] I. Alqassem and D. Svetinovic. A taxonomy of security and privacy requirements for the internet of things (IoT). In *2014 IEEE International Conference on Industrial Engineering and Engineering Management*, pages 1244–1248. IEEE, 2014. → page 53

[6] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M. A. Gerosa. How modern news aggregators help development communities shape and share knowledge. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 499–510. IEEE, 2018. → page 16

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004. → page 13

[8] H. Borges and M. T. Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018. → page 12

[9] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. *Internet Engineering Task Force (IETF): Fremont, CA, USA*, pages 2070–1721, 2014. → page 5

[10] Z. B. Celik, P. McDaniel, and G. Tan. Soteria: Automated IoT safety and security analysis. In *Annual Technical Conference (USENIX ATC)*, pages 147–158, 2018. → page 53

[11] Z. B. Celik, G. Tan, and P. D. McDaniel. IoTguard: Dynamic enforcement of security and safety policy in commodity IoT. In *NDSS*, 2019. → page 53

[12] K. M. Chan, J. Bishop, J. Steyn, L. Baresi, and S. Guinea. A fault taxonomy for web service composition. In *International Conference on Service-Oriented Computing*, pages 363–375. Springer, 2007. → page 55

[13] K. Chen, S. Zhang, Z. Li, Y. Zhang, Q. Deng, S. Ray, and Y. Jin. Internet-of-things security and vulnerabilities: Taxonomy, challenges, and practice. *Journal of Hardware and Systems Security*, 2(2):97–110, 2018. → page 53

[14] Y. Chen, Z. Zhen, H. Yu, and J. Xu. Application of fault tree analysis and fuzzy neural networks to fault diagnosis in the internet of things (IoT) for aquaculture. *Sensors*, 17(1):153, 2017. → pages 1, 9, 52

[15] A. Čolaković and M. Hadžialić. Internet of things (IoT): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144:17–39, 2018. → pages 4, 5, 7, 54

[16] G. Coleman and R. O'Connor. Using grounded theory to understand software process improvement: A study of irish software product companies. *Information and Software Technology*, 49(6):654–667, 2007. → page 18

[17] F. Corno, L. De Russis, and J. P. Sáenz. On the challenges novice programmers experience in developing IoT systems: A survey. *Journal of Systems and Software*, 157:110389, 2019. → pages 1, 9, 54

[18] F. Corno, L. De Russis, and J. P. Sáenz. Towards computational notebooks for IoT development. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–6, 2019. → page 54

[19] F. Corno, L. De Russis, and J. P. Sáenz. How is open source software development different in popular IoT projects? *IEEE Access*, 8: 28337–28348, 2020. → pages 1, 53

[20] J. P. Dias, F. Couto, A. C. Paiva, and H. S. 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. → pages 49, 53, 54

[21] B. Fouladi and S. Ghanoun. Honey, i'm home!!, hacking zwave home automation systems. *Black Hat USA*, 2013. → page 53

[22] P. I. Fusch and L. R. Ness. Are we there yet? data saturation in qualitative research. *The qualitative report*, 20(9):1408, 2015. → pages 14, 16

[23] Github. Lamps not identified as lamps with f/w 1.3.14, 2018. https://github.com/ggravlingen/pytradfri/issues/135. → pages viii, 7, 8

[24] R. Goyal, N. Dragoni, and A. Spognardi. Mind the tracker you wear: a security analysis of wearable health trackers. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 131–136, 2016. → page 53

[25] E. I. W. Group et al. The three software stacks required for IoT architectures, 2016. → pages 4, 5

[26] P. Group. Why can't my dishwasher program my tv? the problems with smart homes, February 2020. URL https://www.iotforall.com/smart-home-problems/. → page 50

[27] G. Guest, A. Bunce, and L. Johnson. How many interviews are enough? an experiment with data saturation and variability. *Field methods*, 18(1):59–82, 2006. → page 16

[28] L. Gutiérrez-Madroñal, I. Medina-Bulo, and J. J. Domínguez-Jiménez. IoT–teg: Test event generator system. *Journal of Systems and Software*, 137: 784–803, 2018. → pages 54, 56

[29] L. Gutiérrez-Madroñal, A. García-Domínguez, and I. Medina-Bulo. Evolutionary mutation testing for IoT with recorded and generated events. *Software: Practice and Experience*, 49(4):640–672, 2019. → pages 54, 56

[30] N. Harrand, F. Fleurey, B. Morin, and K. E. Husa. Thingml: a language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 125–135, 2016. → page 54

[31] J. Henrich, S. J. Heine, and A. Norenzayan. The weirdest people in the world? *Behavioral and brain sciences*, 33(2-3):61–83, 2010. → page 16

[32] G. Hernandez, O. Arias, D. Buentello, and Y. Jin. Smart nest thermostat: A smart spy in your home. *Black Hat USA*, (2015), 2014. → page 53

[33] T. W. Hnat, V. Srinivasan, J. Lu, T. I. Sookoor, R. Dawson, J. Stankovic, and K. Whitehouse. The hitchhiker's guide to successful residential sensing deployments. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 232–245, 2011. → pages 1, 9, 52, 53, 54

[34] M. Hung. Leading the IoT, Gartner insights on how to lead in a connected world. *Gartner Research*, pages 1–29, 2017. → page 1

[35] IOTIFITY. Advanced IoT system simulation engine and test automation for enterprise IoT apps. URL https://iotify.io/. → page 49

[36] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan. A comprehensive study on deep learning bug characteristics. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 510–520, 2019. → page 54

[37] G. Jahangirova, N. Humbatova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella. Taxonomy of real faults in deep learning systems. *arXiv preprint arXiv:1910.11015*, 2019. → pages 1, 54

[38] F. Javed, M. K. Afzal, M. Sharif, and B.-S. Kim. Internet of things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review. *IEEE Communications Surveys & Tutorials*, 20(3):2062–2100, 2018. → page 4

[39] D. Jeffrey, N. Gupta, and R. Gupta. Identifying the root causes of memory bugs using corrupted memory location suppression. In *2008 IEEE International Conference on Software Maintenance*, pages 356–365. IEEE, 2008. → page 31

[40] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile app development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24. IEEE, 2013. → pages 1, 55

[41] M. Kamber, J. Pei, et al. *Data mining: Concepts and techniques*, volume 2. Morgan Kaufmann Publishers San Francisco, 2001. → page 32

[42] I. Kravets. Platformio: An open source ecosystem for IoT development. *PlatformIO.[En ligne]. Disponible sur: https://platformio. org.[Consulté le: 25-sept-2019]*, 2018. → page 49

[43] A. Krishna, M. Le Pallec, R. Mateescu, L. Noirie, and G. Salaün. IoT composer: Composition and deployment of IoT applications. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 19–22. IEEE, 2019. → page 54

[44] B. H. Kwasnik. The role of classification in knowledge representation and discovery. *Graduate School of Library and Information Science. University of Illinois . . .*, 1999. → page 20

[45] H. Liang, Q. Zhao, Y. Wang, and H. Liu. Understanding and detecting performance and security bugs in IoT oses. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 413–418. IEEE, 2016. → pages 1, 9, 52

[46] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet of Things Journal*, 4(6):1899–1909, 2017. → page 53

[47] V. Looga, Z. Ou, Y. Deng, and A. Ylä-Jääski. Mammoth: A massive-scale emulation platform for internet of things. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, volume 3, pages 1235–1239. IEEE, 2012. → page 49

[48] S. Lu, Z. Li, F. Qin, L. Tan, P. Zhou, and Y. Zhou. Bugbench: Benchmarks for evaluating bug detection tools. In *Workshop on the evaluation of software defect detection tools*, volume 5, 2005. → page 31

[49] A. Makhshari and A. Mesbah. *IoT Bugs and Development Challenges Artifact Package*, August 2020.

https://github.com/IoTSEstudy/IoTbugschallenges. → pages 2, 10, 18, 20, 51

[50] A. Makhshari and A. Mesbah. Iot bugs and development challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 460–472. IEEE, 2021. → page iv

[51] R. Minerva, A. Biru, and D. Rotondi. Towards a definition of the internet of things (IoT). *IEEE Internet Initiative*, 1(1):1–86, 2015. → pages 1, 4

[52] B. Morin, N. Harrand, and F. Fleurey. Model-based software engineering to tame the IoT jungle. *IEEE Software*, 34(1):30–36, 2017. → page 54

[53] J. M. Morse. Data were saturated..., 2015. → page 16

[54] M. Murdoch. Arduinounit, 2013. URL https://github.com/mmurdoch/arduinounit. → page 49

[55] A. Nguyen-Duc, R. Jabangwe, P. Paul, and P. Abrahamsson. Security challenges in IoT development: a software engineering perspective. In *XP Workshops*, pages 11–1, 2017. → page 53

[56] M. Norris, B. Celik, P. Venkatesh, S. Zhao, P. McDaniel, A. Sivasubramaniam, and G. 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. → page 57

[57] S. Notra, M. Siddiqi, H. H. Gharakheili, V. Sivaraman, and R. Boreli. An experimental study of security and privacy risks with emerging household appliances. In *2014 IEEE conference on communications and network security*, pages 79–84. IEEE, 2014. → page 53

[58] F. S. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah. A study of causes and consequences of client-side javascript bugs. *IEEE Transactions on Software Engineering*, 43(2):128–144, 2016. → page 55

[59] P. Patel and D. Cassou. Enabling high-level application development for the internet of things. *Journal of Systems and Software*, 103:62–84, 2015. → page 54

[60] P. M. Pontes, B. Lima, and J. P. Faria. Test patterns for IoT. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, pages 63–66, 2018. → page 49

[61] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn. IoT goes nuclear: Creating a zigbee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 195–212. IEEE, 2017. → page 53

[62] P. Rosenkranz, M. Wählisch, E. Baccelli, and L. Ortmann. A distributed test system architecture for open-source IoT software. In *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*, pages 43–48, 2015. → page 53

[63] F. Schwandt. Internet of things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). *Statista*, 2016. → page 1

[64] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999. → page 14

[65] O. Serrat. The five whys technique. In *Knowledge solutions*, pages 307–310. Springer, 2017. → page 13

[66] L. Singer, F. Figueira Filho, and M.-A. Storey. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, pages 211–221, 2014. → page 16

[67] V. Song. The never ending death of smart home gadgets, Mar 2020. URL https://gizmodo.com/the-never-ending-death-of-smart-home-gadgets-1842456125. → page 50

[68] B. L. R. Stojkoska and K. V. Trivodaliev. A review of internet of things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140: 1454–1464, 2017. → pages 1, 4, 5, 9, 54

[69] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li. An empirical study on real bugs for machine learning programs. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 348–357. IEEE, 2017. → page 54

[70] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai. Bug characteristics in open source software. *Empirical software engineering*, 19(6):1665–1705, 2014. → page 13

[71] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai. Bug characteristics in open source software. *Empirical software engineering*, 19(6):1665–1705, 2014. → page 31

[72] F. Thung, S. Wang, D. Lo, and L. Jiang. An empirical study of bugs in machine learning systems. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, pages 271–280. IEEE, 2012. → page 54

[73] M. D. C. Tongco. Purposive sampling as a tool for informant selection. *Ethnobotany Research and applications*, 5:147–158, 2007. → page 16

[74] C. Towers-Clark. Uk to introduce new law for IoT device security, May 2019. URL https://www.forbes.com/sites/charlestowersclark/2019/05/02/uk-to-introduce-new-law-for-iot-device-security/. → page 50

[75] H. Tschofenig, J. Arkko, and D. McPherson. Architectural considerations in smart object networking, internet engineering task force, rfc-7452. *Internet Engineering Task Force, Fremont, CA, USA*, 2014. → page 5

[76] M. Usman, R. Britto, J. Börstler, and E. Mendes. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology*, 85:43–59, 2017. → page 20

[77] J. Voas, R. Kuhn, and P. Laplante. Testing IoT systems. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 48–52. IEEE, 2018. → pages 49, 53

[78] vyshwanara. Lack of hardware clock in raspberry pi - possible time lag issues, 2018. URL https://blog.pisignage.com/lack-of-hardware-clock-in-raspberry-pi-scheduling-issues/. → page 21

[79] Z. Wan, D. Lo, X. Xia, and L. Cai. Bug characteristics in blockchain systems: a large-scale empirical study. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 413–424. IEEE, 2017. → pages 1, 54, 55

[80] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv. Edge computing security: State of the art and challenges. *Proceedings of the IEEE*, 107(8): 1608–1631, 2019. → page 53

[81] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang. An empirical study on program failures of deep learning jobs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1159–1170. IEEE, 2020. → page 54

[82] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang. An empirical study on tensorflow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 129–140, 2018. → page 54

[83] H. Zhou, J.-G. Lou, H. Zhang, H. Lin, H. Lin, and T. Qin. An empirical study on quality issues of production big data platform. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 17–26. IEEE, 2015. → page 55

[84] W. Zhou, C. Cao, D. Huo, K. Cheng, L. Zhang, L. Guan, T. Liu, Y. Zheng, Y. Zhang, L. Sun, et al. Logic bugs in iot platforms and systems: A review. *arXiv preprint arXiv:1912.13410*, 2019. → page 52

[85] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang. Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms. In *28th USENIX Security Symposium (USENIX Security)*, pages 1133–1150, 2019. → pages 5, 7, 53

[86] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 2019. → page 55

Appendix A

Supporting Materials

Table A.1: Subject Repositories (part 1)

Repo Name	Bug Count	Stars	Forks	Language
kubeedge/kubeedge	188	2219	559	Go
lelylan/lelylan	41	1465	91	JavaScript
ct-Open-Source/tuya-convert	3	1328	192	Python
GladysAssistant/Gladys	52	1326	191	JavaScript
homieiot/homie-esp8266	94	1109	246	HTML
mysensors/MySensors	85	1046	824	C++
timdorr/tesla-api	2	994	376	Ruby
particle-iot/device-os	327	928	501	C
home-assistant/operating-system	3	730	224	Makefile
pliablepixels/zmNinja	200	635	196	JavaScript
Nekmo/amazon-dash	13	600	51	Python
codetheweb/tuyapi	13	563	106	JavaScript
256dpi/arduino-mqtt	3	553	146	C
Azure/iot-edge-v1	56	511	267	C
thingsboard/thingsboard-gateway	3	474	253	Python
macchina-io/macchina.io	3	412	121	C++
smartHomeHub/SmartIR	3	342	209	Python
Azure/azure-iot-sdk-c	126	340	464	C
espressif/esp-mqtt	2	318	160	C
danielwelch/hassio-zigbee2mqtt	2	318	103	Shell
freedomotic/freedomotic	21	313	475	Java
ikalchev/HAP-python	5	279	73	Python
balloob/pychromecast	2	1888	267	Python
blynkkk/blynk-server	300	1717	616	Java
eclipse/smarthome	589	857	847	Java
ggravlingen/pytradfri	4	663	112	Python
sitewhere/sitewhere	144	619	275	Java
eclipse/upm	30	612	394	C++
esphome/esphome-core	7	552	113	C++
eclipse/kura	465	320	226	Java
eclipse/hono	217	275	90	Java
microsoft/devkit-sdk	91	46	42	C
NEEOInc/neeo-sdk	26	46	14	TypeScript
futurice/vor	12	44	9	Java

Table A.2: Subject Repositories (part 2)

Repo Name	Bug Count	Stars	Forks	Language
openhab/openhab-core	54	253	191	Java
eclipse/hawkbit	26	229	93	Java
oakbrad/brad-homeassistant-config	3	207	22	HTML
OpenZWave/Zwave2Mqtt	77	204	52	JavaScript
devicehive/devicehive-java-server	30	202	92	Java
particle-iot/particle-cli	67	200	86	JavaScript
yodaos-project/yoda.js	16	183	48	JavaScript
theyosh/TerrariumPI	10	178	64	JavaScript
adafruit/Adafruit_IO_Python	9	146	57	Python
hoobs-org/HOOBS	109	131	14	Shell
danimtb/dasshio	5	120	55	Python
AlCalzone/node-tradfri-client	19	119	24	TypeScript
mysensors/NodeManager	130	104	72	C++
danobot/entity-controller	4	95	25	Python
Norien/Home-Assistant-Config	2	91	11	Python
smarthomeNG/smarthome	21	80	81	Python
Azure/iotedgedev	13	78	40	Python
esphome/issues	6	60	8	Python
bertmelis/VitoWiFi	5	60	15	C++
openairproject/sensor-esp32	7	50	15	C
ibm-watson-iot/iot-java	10	49	81	Java
alf45tar/PedalinoMini	7	49	14	C
eclipse/ditto	6	146	52	Java
eclipse/kapua	688	144	135	Java
mycontroller-org/mycontroller	86	127	82	Java
enesbcs/rpieasy	23	44	10	Python
klausahrenberg/WThermostatBeca	3	81	18	C++
quickbird-uk/QuickbirdUWPDashboard	13	19	5	C#
thingsboard/thingsboard	3	5183	1756	Java
cesanta/mongoose-os	7	1929	383	C
mainflux/mainflux	66	796	278	Go
pihome-shc/pihome	5	27	21	HTML
iobroker-community-adapters/ioBroker.hue	4	26	25	JavaScript
JakeHartnell/Grow.js	7	23	11	JavaScript

Table A.3: Subject Repositories (part 3)

Repo Name	Bug Count	Stars	Forks	Language
home-assistant/home-assistant	236	31665	9549	Python
hybridgroup/gobot	28	6253	780	Go
hybridgroup/cylon	3	3676	339	JavaScript
blynkkk/blynk-library	67	2554	732	C++
freedomotic/freedomotic	21	313	476	Java
grodansparadis/vscp	42	32	16	C++
foss-for-synopsys-dwc-arc-processors/embarc_osp	9	31	38	C
vapor-ware/synse-server	21	23	8	Python
domoticz/domoticz	4	2500	942	C
home-assistant/appdaemon	29	417	293	Python
DexterInd/GrovePi	19	397	418	Python
telefonicaid/iotagent-node-lib	98	44	52	JavaScript
tobyweston/temperature-machine	6	40	12	Scala
museumsvictoria/nodel	31	36	8	Java
phpMyDomo/phpMyDomo	6	32	20	PHP
casanet/casanet-server	11	32	8	TypeScript
MCS-Lite/mcs-lite-app	22	27	10	JavaScript
telefonicaid/iotagent-json	27	21	56	JavaScript
telefonicaid/lightweightm2m-iotagent	13	19	25	JavaScript
CommonGarden/Grow-IoT	14	17	1	JavaScript
rbccps-iisc/ideam	39	14	10	Python
telefonicaid/iotagent-ul	33	17	41	JavaScript
AstroPrint/OctoPrint-AstroPrint	14	15	4	Python