POLITECNICO DI TORINO
Repository ISTITUZIONALE

How is Open Source Software Development Different in Popular IoT Projects?

Original
How is Open Source Software Development Different in Popular IoT Projects? / Corno, F., DE RUSSIS, L., SAENZ MORENO, J.P.. - In: IEEE ACCESS. - ISSN 2169-3536. - ELETTRONICO. - 8:(2020), pp. 28337-28348. [10.1109/ACCESS.2020.2972364]

Availability:
This version is available at: 11583/2790199 since: 2020-03-03T15:30:39Z

Publisher:
IEEE

Published
DOI:10.1109/ACCESS.2020.2972364

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

29 November 2026

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 13, 2020, accepted February 3, 2020, date of publication February 7, 2020, date of current version February 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2972364

How Is Open Source Software Development Different in Popular IoT Projects?

FULVIO CORNO, (Member, IEEE), LUIGI DE RUSSIS, (Member, IEEE), AND JUAN PABLO SÁENZ, (Student Member, IEEE)
Department of Control and Computer Engineering, Politecnico di Torino, 10129 Torino, Italy
Corresponding author: Luigi De Russis (luigi.derussis@polito.it)

ABSTRACT From the software point of view, the development of IoT applications differs from other kinds of applications due to the specific features that the former exhibit. In this paper, we investigate how developers contribute to IoT applications in the Open Source Software (OSS) context, to gain a deeper understanding of how their work differs from that of non-IoT applications. To that end, we conducted a quantitative analysis of a broad set of the 60 most popular publicly available IoT and non-IoT projects on GitHub. By comparing how developers contribute to these projects, our analysis provides insight into the purpose and characteristics of the code, the behavior of the contributors, and the maturity of the IoT software development ecosystem. Results reveal significant differences between IoT and non-IoT application development, in terms of how applications are realized, in the diversity of developers' specializations, and in how code is reused. This work provides evidence about some Open Source IoT software development peculiarities to be considered by future research efforts aimed at better satisfying software engineering needs in the IoT scenario.

INDEX TERMS Internet of Things, open source software, software mining, developers.

I. INTRODUCTION

Nowadays, the Internet of Things (IoT) is a well-established paradigm that has gained prominence in several aspects of our everyday lives [1]. Roughly speaking, it is based on embedding computing and communication capabilities into objects of common use [2]. This concept has given rise to the development of various kinds of solutions in several domains such as smart buildings, smart cities, environmental monitoring, healthcare, smart business, smart agriculture, and security and surveillance [3]–[5].

From a technical point of view, several definitions have been proposed for the Internet of Things [6] and various enabling technologies are considered to characterize IoT applications. According to Atzori *et al.* [7], these technologies may be categorized into identification, sensing and communication technologies; middleware components; end-user software applications; services composition; service management; and object abstraction. While identification, sensing and communication technologies mainly concern hardware components, the other enabling technologies rely on *software* to address diverse features that IoT applications expose [2].

From the software point of view, in addition, the implementation of IoT applications is particularly complex and differs from the development of mobile and web applications. According to Taivalsaari *et al.* [8], for instance, IoT development differs from mainstream mobile app and web application development in several ways, summarized by the authors into a set of *dimensions* that are unfamiliar to most software developers. Multi-device programming, the reactive nature of the application, the distributed nature of the software, and the need to write fault-tolerant software, are among these dimensions, which IoT developers must consider.

Against this backdrop, the present work relies upon software mining to gain understanding, from a practical point of view, about how developing IoT applications is different from developing non-IoT applications in the Open Source Software (OSS) context. To this end, this paper reports the comparison and quantitative analysis between the behavior of developers in the most popular IoT and non-IoT OSS projects hosted on a world leading software development platform as is GitHub.

The associate editor coordinating the review of this manuscript and approving it for publication was Pietro Savazzi.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

In particular, we conducted an empirical study mining 60 OSS repositories publicly available on GitHub. We mined 30 IoT OSS and 30 non-IoT OSS projects to analyze *a)* the way developers contribute to their projects, *b)* the files that they tend to modify the most, and *c)* the specialization and the evolution of these modifications. Finally, we assessed the maturity of the IoT software development ecosystem based on a dependency analysis in the selected projects. Besides leveraging a characterization of IoT OSS projects currently available for IoT developers, this work aims at providing evidence from a practical point of view about the IoT software development peculiarities that should guide future research efforts to better understand and satisfy software engineering needs in the IoT context.

The remainder of the paper is structured as follows. Section II describes the research goal and questions and outlines the selection process. Section III characterizes the selected OSS projects and describes the quantitative analysis conducted over them as well as the outcome of the analysis. Section IV discusses the results and presents further implications, while threats to validity are outlined in Section V. Section VI presents the related work, while Section VII concludes the article.

II. RESEARCH GOAL AND QUESTIONS

The overall goal of this research is to explore the potential differences between the development practices for IoT and non-IoT projects in the OSS context. In particular, we are interested in identifying (a) the behavior of developers and the diversity of resources they manage, and (b) the reuse of features through the adopted dependencies. These two criteria lead us to the research questions set out below.

A. RESEARCH QUESTIONS

We want to investigate whether and how developers adopt different programming languages and cover various specializations in IoT vs. non-IoT OSS projects. In particular, we are interested in:

- how different programming languages are used in the two domains;
- whether IoT developers are more specialized in any programming languages or certain types of files in their project;
- how the usage of such programming languages evolve over time.

Therefore, our first research question is:

RQ1: How developers of IoT vs. non-IoT OSS applications contribute to their projects regarding the programming languages that they adopt?

Our quantitative investigation, furthermore, exploits OSS repositories by focusing on the maturity of the IoT ecosystem for a software development point of view. We investigate this aspect in the repositories we selected by analyzing project dependencies, how many they are, and which are the most popular ones. Additionally, focusing on the IoT OSS projects, we wanted to identify which aspects of IoT application development these dependencies address and how often they are used by IoT developers. This leads to our second research question:

RQ2: How developers exploit dependencies to reuse features in IoT vs. non-IoT OSS projects?

B. SELECTION OF THE ANALYZED REPOSITORIES

To select a prominent widely-known and widely-used set of IoT OSS repositories from GitHub, we first filtered them by topic, choosing the ones that belong to the `iot` or `internet-of-things` topics on GitHub. Topics are labels to classify a repository based on its intended purpose, subject area, community, or language. They appear on the main page of a repository and repository administrators can add as many topics as they want to a repository.

Once the repositories belonging to the IoT topic were filtered, 4,696 repositories were retrieved. Therefore, to prioritize the most popular and well-evaluated ones, we sorted them according to the decreasing number of stars. Stars enable GitHub users to keep track of repositories they find interesting and to discover similar repositories [9], as well as to show appreciation to the repository maintainers for their work.[1] Lastly, we took the 30 top-starred repositories, provided they were open source code repositories. In fact, since a large portion of repositories on GitHub are not for software development [10], we inspected them manually to exclude the ones that were not software related (i.e., tutorials, documentation pages, icon-packs, fonts) or without an open source license.

The same procedure was followed to select the non-IoT repositories. The only difference was that the filter was modified to include repositories belonging to any topic *except* `iot` and `internet-of-things`.

The data used in the analyses reported in this article was mined from GitHub in August 2018. Tables 1 and 2 list the selected IoT and non-IoT repositories along with their salient characteristics. Most of the information about the repositories was gathered through the GitHub GraphQL API v4.[2]

III. OSS PROJECTS ANALYSIS

A. PROJECTS CHARACTERIZATION

Before diving into the research questions, we report a characterization of the selected projects, to provide a brief but complete overview and to set the stage for the subsequent analysis. Each project was examined individually to understand its purpose and to assign it a genre. The genres aimed at describing the nature of the projects. Then, through the GitHub API, several characteristics were gathered, namely: the topics, their size (kB and lines of code), their primary language, and their total number of programming languages. Additionally,

[1] https://help.github.com/articles/about-stars/, last visited on June 6, 2019
[2] https://developer.github.com/v4/, last visited on June 6, 2019

TABLE 1. IoT popular Open Source GitHub repositories.

Repository name	Genre	Size (kB)	LOC	Prim. Lang.	# Langs.	# Depend.	Commits	Contributors
netdata	Monitoring agent	24,473	259k	C	14	18	7,321	223
kong	API gateway	9,802	151k	Lua	4	24	4,118	141
home-assistant	Home automation	85,734	562k	Python	4	456	14,773	1,211
johnny-five	Robotics programming framework	92,868	136k	JavaScript	3	338	3,215	152
gun	Graph database engine	29,683	124k	JavaScript	5	139	1,532	66
timescaledb	Time-series database	2,975	159k	C	9	0	833	33
gobot	Programming framework	9,668	179k	Go	4	16	2,507	109
node-serialport	Package to access serial ports	2,688	19k	JavaScript	5	18	1,208	145
emqx	MQTT broker	11,682	29k	Erlang	2	4	3,060	53
cylon	Programming framework	20,046	7k	JavaScript	2	5	1,323	26
urh	Wireless protocols monitoring	43,538	197k	Python	4	4	2,579	13
ArduinoJson	Arduino library	3,242	34k	C++	6	0	984	10
platformio-core	Cross-platform IDE	34,704	10k	Python	5	6	3,626	27
crate	Distributed SQL database	86,570	667k	Java	6	15	8,775	62
RIOT	Operating system	56,065	2.04M	C	10	0	19,368	287
thingsboard	IoT platform	8,785	257k	Java	9	109	1,510	59
rt-thread	Operating system	254,378	13.38M	C	24	0	6,549	226
blynk-library	IoT platform	9,318	33k	C++	7	0	1,691	19
openthread	Thread networking protocol	50,835	1.5M	C++	10	4	2,443	85
mongoose-os	Firmware development framework	44,630	106k	C	10	0	4,212	32
vernemq	MQTT broker	11,426	67k	Erlang	8	21	1,713	22
BerryNet	Deep learning gateway	181	14k	Python	4	14	159	6
PJON	Network protocol	4,529	29k	C++	2	0	1,999	34
zephyr	Operating system	123,632	10.05M	C	12	0	23,231	420
blynk-server	IoT platform	29,763	81k	Java	6	21	4,545	14
paho.mqtt.android	MQTT client library	2,014	15k	Java	3	0	194	20
tock	Operating system	136,163	79k	Rust	8	0	4,085	82
kaa	IoT platform	180,031	775k	Java	16	3	6,691	109
Sming	Programming framework	52,066	195k	C++	16	11	1,236	97
homie-esp8266	MQTT convention	1,305	10k	HTML	5	142	1,714	155

to put into perspective the comparison of the projects' size, we illustrate through the heatmap graphs in Figures 1 and 2 the growth of the source code along the projects' lifetime.

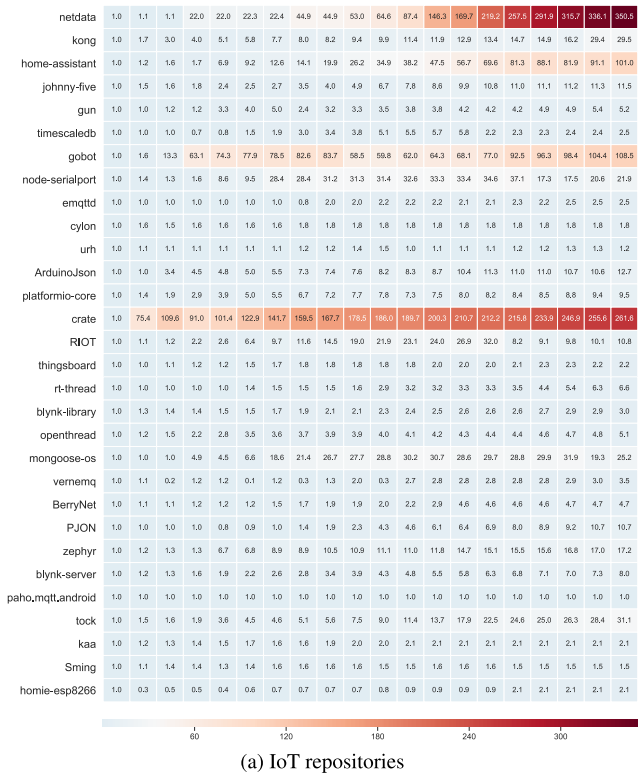
As observed in Table 1, the **genre** of the IoT OSS projects is heterogeneous, as they are scattered across operating systems, programming frameworks, libraries, network protocols, databases, IoT platforms, and IDEs. At first glance, no clear trend emerged concerning their purpose or application domain. On the contrary, when analyzing non-IoT projects (Table 2), we can notice that most of them are related to the web development area, with just 12 exceptions, such as a machine learning framework, a Zsh framework, an operating system kernel, an IDE, a text editor, and a couple of open source programming languages.

The fifteen most commonly used **topics** across the IoT projects (*mqtt*, *raspberry-pi*, *arduino*, *hardware*, *esp8266*, *esp32*, *embedded*, *robotics*, *javascript*, *java*, *iot-platform*, *i2c*, *home-automation*, *gpio*, *docker*) did not reveal a prevailing technology or application domain. Instead, the 15 topics across the non-IoT projects (*javascript*, *nodejs*, *html*, *framework*, *electron*, *css*, *windows*, *web*, *ui*, *react*, *python*, *macos*, *linux*, *go*, *frontend*) are mostly about web development. This fact leads us to think that neither in our classification nor in the labels assigned by the owners to their IoT projects, there is a strong focus towards a particular domain or technology, thus further motivating our investigation and research questions. Furthermore, our initial observations regarding the genre and the topics of the projects seem to be in line with various authors [6], [8], [11], who point out that the development of IoT applications is more complex and requires programmers with skills and expertise in several domains as might be, for instance, mobile and cloud computing, embedded devices, database design, and web development.

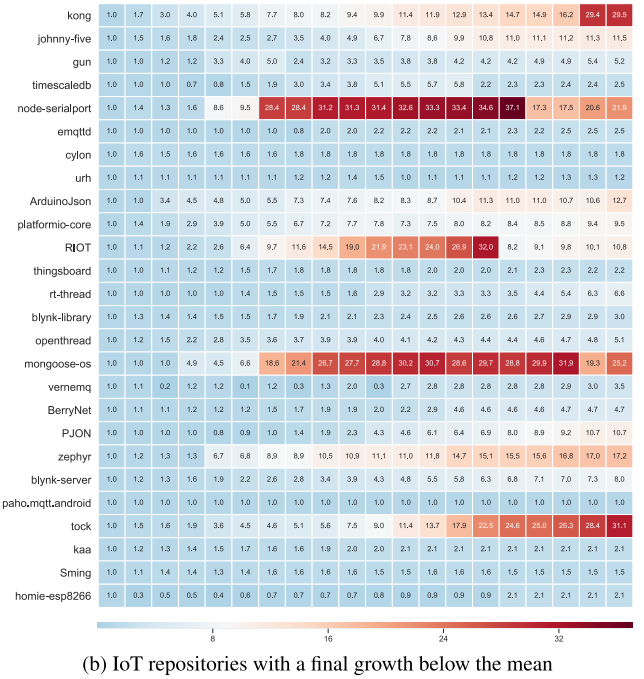
Concerning the **size** of the projects (in kB), the average non-IoT project is almost three times larger (4.56×) than a typical IoT project. However, if we look at LOC (Lines Of Code), this difference decreases significantly: on average, non-IoT projects contains 1.9M LOC, while IoT projects 1.0M (1.9×). The largest IoT project, for both kB and LOC, corresponds to `rt-thread`, a real-time IoT operating system for embedded devices. Similarly, the largest non-IoT project is the `Linux` kernel followed far behind by `kubernetes`. The smallest IoT project, in kB, is `BerryNet`, a project to turn edge devices such as Raspberry Pi 3 into intelligent gateways with deep learning capabilities running locally, on the edge device itself, without the need of an Internet connection. For what concerns LOCs, instead, the smallest IoT project is `cylon`, a JavaScript framework for robots, drones, and the IoT, developed for Arduino and similar boards. As may be observed in these last two projects, achieving a small size is fundamental given the fact that in most cases IoT software components are deployed on constrained devices with low computational

TABLE 2. Non-IoT popular Open Source GitHub repositories.

Repository name	Genre	Size (kB)	LOC	Prim. Lang.	# Langs.	# Depend.	Commits	Contributors
bootstrap	Web UI framework	124,291	102k	CSS	6	52	17,950	1,192
vue	Web UI framework	23,784	164k	JavaScript	6	89	2,620	209
react	Web UI framework	137,522	277k	JavaScript	10	78	10,326	1,379
tensorflow	Machine learning framework	189,005	4.5M	C++	21	31	41,273	1,923
d3	Data visualization library	35,963	55k	JavaScript	1	39	4,153	133
oh-my-zsh	Zsh framework	4,730	60k	Shell	6	0	4,785	1,472
react-native	Native apps framework	256,123	602k	JavaScript	17	73	14,743	2,157
electron	Desktop applications framework	40,449	195k	C++	11	829	20,369	911
linux	Linux kernel	2,192,884	26.5M	C	19	3	782,537	19,120
angular.js	MVC web framework	98,788	554k	JavaScript	5	76	8,883	1809
vscode	IDE	149,758	1.12M	TypeScript	33	102	40,831	834
create-react-app	React app setup command	5,718	129k	JavaScript	5	18	1,782	551
animate.css	CSS animations library	713	6k	CSS	2	572	423	101
node	JavaScript runtime engine	399,664	9M	JavaScript	13	12	23,947	2,480
moby	Programming framework	137,525	1.43M	Go	8	4	35,841	2,120
jquery	JavaScript library	27,910	93k	JavaScript	4	39	6,343	349
axios	Promise based HTTP client	2,762	10k	JavaScript	3	37	833	171
atom	Text editor	301,069	223k	JavaScript	7	514	35,684	516
go	Go programming language	182,273	2.44M	Go	16	11	38,051	1,404
laravel	Web framework	9,222	5k	PHP	3	8	5,804	573
swift	Swift programming language	324,175	2.47M	C++	17	0	78,463	779
three.js	JavaScript 3D library	662,910	1.39M	JavaScript	5	399	25,250	1,238
redux	Programming framework	6,562	184k	JavaScript	3	29	2,684	666
socket.io	Real-time application framework	12,264	182k	JavaScript	1	11	1,706	171
webpack	Module bundler	16,478	124k	JavaScript	5	74	7,259	552
Semantic-UI	Web UI framework	110,010	283k	JavaScript	3	508	6,659	232
reveal.js	HTML presentations framework	8,271	33k	JavaScript	3	15	2,200	264
rails	Web framework	165,151	524k	Ruby	7	62	70,368	4,490
meteor	Web framework	76,020	529k	JavaScript	9	12	21,688	474
kubernetes	Container-orchestration system	797,674	4.72M	Go	10	10	73,512	1,951

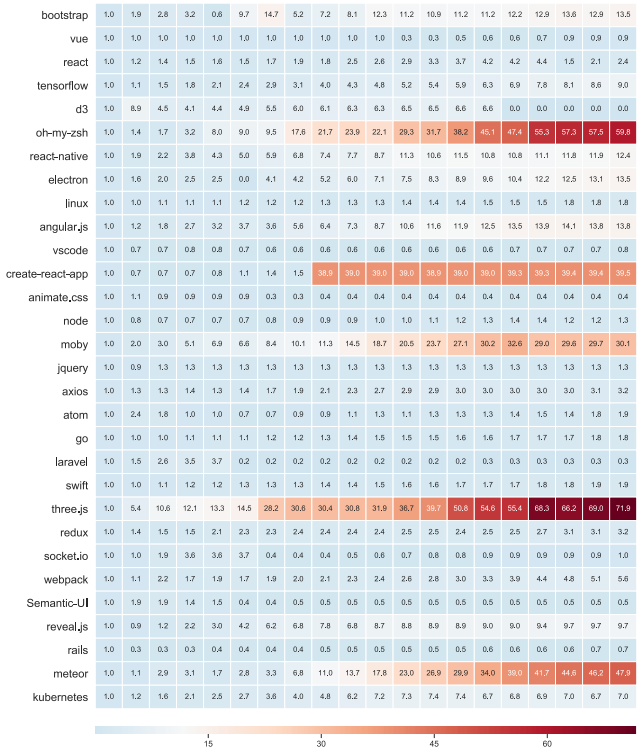


(a) IoT repositories

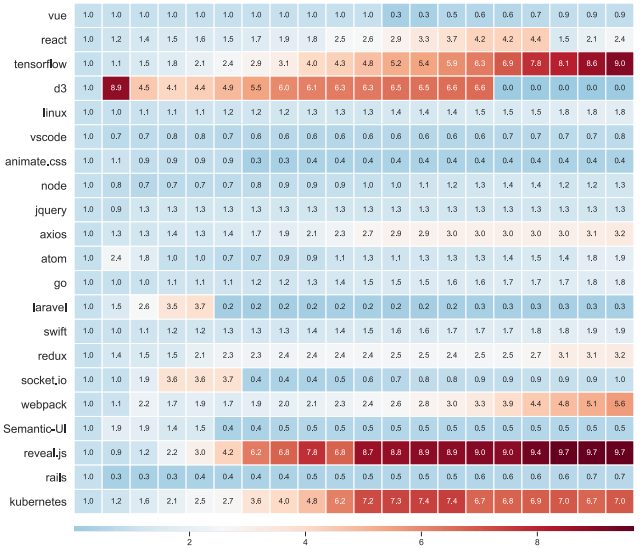


(b) IoT repositories with a final growth below the mean

FIGURE 1. Growth speed of the IoT repositories.



(a) Non-IoT repositories



(b) Non-IoT repositories with a final growth below the mean

FIGURE 2. **Growth speed of the non-IoT repositories.**

and/or storage resources. This same restriction holds for most of the other IoT projects, especially those to be deployed on the gateway architectural element.

Finally, Figure 1 (for IoT projects) and Figure 2 (for non-IoT projects) aim at visualizing the growth of the projects' source code, expressed as the proportion between the initial size of the programming files and their size along the lifetime of the projects. We divided the period between the first commit in the project and the last commit before August 2018 (the date when the repositories were mined for this analysis), into 21 equally spaced date intervals for each project. Then, on each of these dates, we checked out from GitHub the corresponding version of the project and calculated the size of the programming files. To this end, we relied on Linguist; the open-source library that GitHub uses to determine file languages for syntax highlighting, and project statistics.[3] Specifically, we used the Ruby API provided by this library that, given a directory, returns a dictionary with the detected programming languages along with their size.

The growth of the project was calculated by dividing the size of each checked out version of the project by the size of the second checked out version. By taking the second version instead of the first one (initial commit) we could avoid empty projects (without source code) that would have made our calculation impossible or meaningless. In this manner, the first measure is always one, and the following values represent the variation regarding the initial size of the projects' programming files. Hence, the last measure represents how many times the source code grew in comparison with respect to its initial size.

As can be observed in Figure 1a, a subset of four IoT projects grew up hugely. Namely `netdata` (350 times, 24.4 MB, and 7.3k commits), `home-assistant` (101 times, 85.7 MB, and 14.7k commits), `gobot` (108 times, 9.6 MB, and 2.5k commits), and `crate` (261 times, 86.5 MB, and 8.7k commits). Indeed, while the average growth is 35.15 times, the standard deviation is 78.83 times. To improve the readability of the graph for project with less dramatic growth, we generated a second heatmap visualization, restricted to the projects whose final growth is below the mean, only (Figure 1b).

Concerning non-IoT projects (Figure 2b), five of them grew up significantly, although not as dramatically as the subset of IoT projects that grew above the mean. These repositories were: `oh-my-zsh` (60 times, 4.7 MB, and 4.7k commits), `create-react-app` (39 times, 5.7 MB, and 1.7k commits), `moby` (30 times, 137.5 MB, and 35.8k commits), `three.js` (72 times, 662.9 MB, and 25.2k commits), and `meteor` (47.9 times, 76.0 MB, and 21.6k commits). The average growth in non-IoT projects is 11.88 times, and the standard deviation 18.81 times. As with the IoT projects, Figure 2b reports a second heatmap visualization with the IoT projects whose final growth is below the mean.

[3]https://github.com/GitHub/linguist, last visited on November 26, 2019

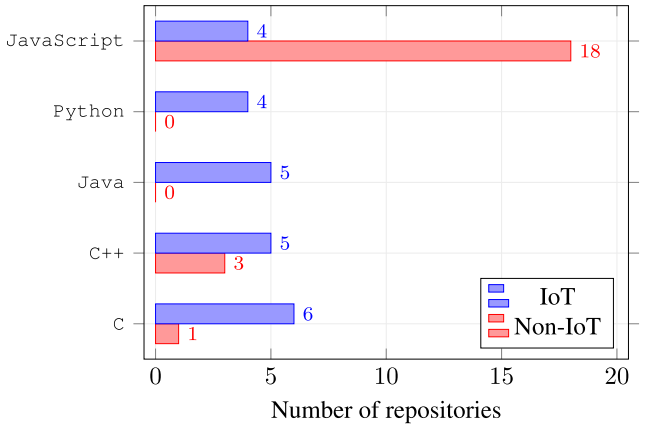


FIGURE 3. Top primary programming languages in IoT and non-IoT repositories.

Among the IoT projects, `paho.mqtt.android` is the one that has remained more stable over time (1.0 times, 2.0 MB, and 194 commits), it consists of an MQTT client library written in Java for developing applications on Android. Nevertheless, the last of its 195 commits was 4ht October 2017, and it has just two releases. After it, the project that remained more stable was `urh` (1.2 times, 43.5 MB, and 2.5k commits), it consists of a tool for analyzing unknown wireless protocols by taking samples from Software Defined Radios and transforming them into binary information. For its part, the non-IoT project whose code growth remained more stable over time is `socket.io` (1.0 times, 12.2 MB, and 1.7k commits), a library that enables real-time, bidirectional and event-based communication between the browser and the server.

B. RQ1: DEVELOPMENT ACTIVITIES

To answer RQ1, we performed an analysis of the commit history for all the OSS projects. In particular, each repository was cloned locally so that its git history could be saved into an external text file, and processed later by a custom-developed text mining tool. This tool extracted from each commit the set of files that were modified, the modification date, and the author name. Several classifications and cross-checking analyses over this information allowed us to determine the most widely-modified file formats, and especially the commit history over time of such resources. In addition, we gathered complementary information from the GitHub API, when appropriate.

1) DISTRIBUTION OF PROGRAMMING LANGUAGES

Among the information that Linguist provides there is the primary language, which is the most used programming language within a project (Figure 3). The most popular primary programming language among non-IoT projects is JavaScript, which is the also the *lead* language since 18 non-IoT projects use it (60%). It is followed far behind by C++ and C (3 and 1 project, respectively). IoT projects, instead, exhibit a more balanced distribution of primary languages, with the most popular languages being C, C++, Java, Python, and JavaScript. All of them are the primary language on almost the same number of projects (from 4 to 6 projects, each).

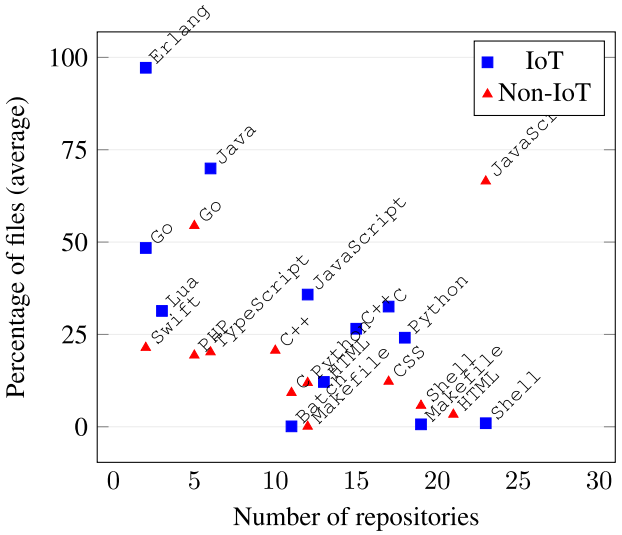
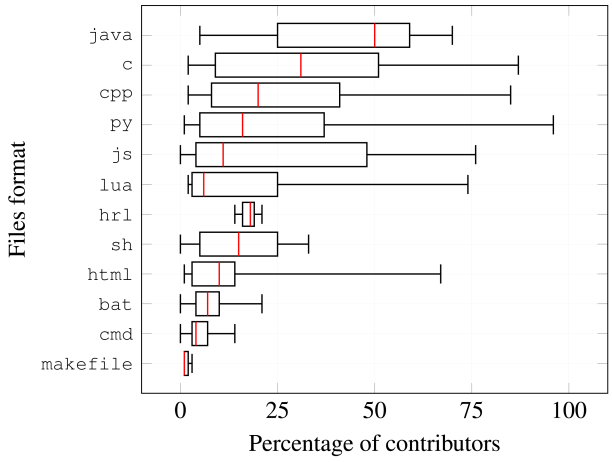


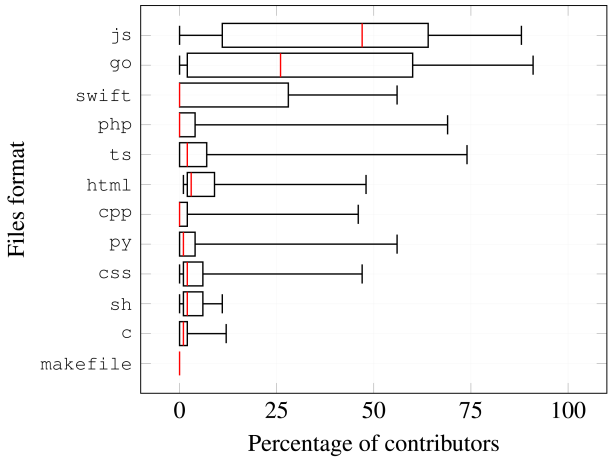
FIGURE 4. Presence of programming languages in IoT and non-IoT projects.

Besides the primary language, there are several other languages on each project: on average, 7.4 different languages for non-IoT projects vs. 8.3 for IoT projects. To gather additional insight on this comparison, since the averages' difference is not statistically significant due to the small size of the sample, we compared the percentage of files written in a given programming language with the number of projects in which that language is present and reported it in Figure 4. It illustrates, given a programming language, the number of projects in which it is present, and the average percentage of files on those projects. Regarding this graph, it can be observed that no languages were present on a high number of IoT projects with a significant percentage of files (right-upper quadrant). In most of the IoT projects, the chart identifies programming languages that are present in many projects with a marginal percentage (right-lower quadrant), as well as programming languages that have a significant percentage of files but just on a few projects (left-upper quadrant). In the first category, Java and Erlang have a significant percentage of files on a few projects. In the second category, C++, C, and Python are present in around half of the projects, with percentages of files ranging from 26% to 32%. Furthermore, several IoT projects have a small portion of Shell scripts (on average, 0.96% of the files in 23 projects).

For non-IoT projects, JavaScript is still the only programming language with a significant percentage of files on most projects (66.47% on 23 projects). This results gives an initial indication that the programming languages IoT developers deal with are observably different and more varied from those worked on by non-IoT developers, and supports the idea that the development of IoT applications requires programmers with skills and expertise in several domains.

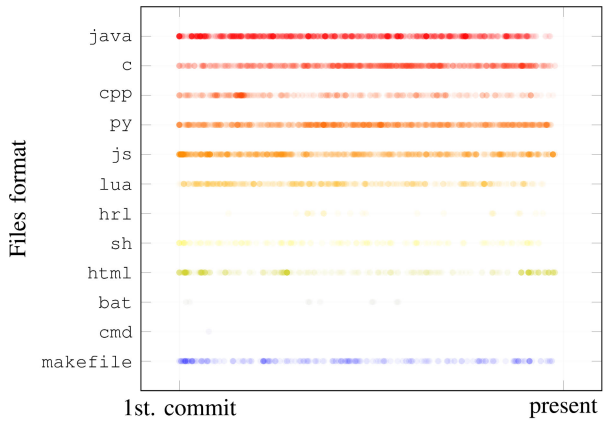


(a) Percentage of contributors that modified certain files formats in IoT repositories

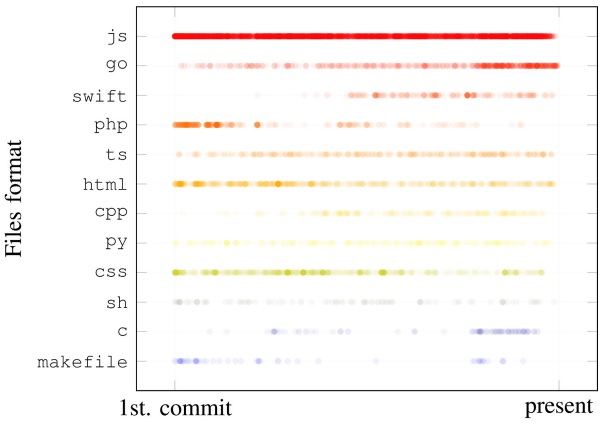


(b) Percentage of contributors that modified certain files formats in non-IoT repositories

FIGURE 5. **Percentage of contributors by file format.**



(a) IoT repositories



(b) Non-IoT repositories

FIGURE 6. **Commit history over time by file format.**

2) SPECIALIZATION OF CONTRIBUTORS BY PROGRAMMING LANGUAGE

Figures 5a and 5b illustrate the average percentage of contributors that modify the files developed in a given programming language, among the projects where it is present. Inside the IoT projects, the files modified by a higher proportion of the contributors are Java, C, C++, Python, and JavaScript. As before, this result indicates that programming languages used by IoT developers are more variegate and diverse than other contexts, with a lower specialization towards a few lead languages. On the contrary, shell executable files, batch files, and command files are manipulated by a percentage that reaches, on average, 15% of the contributors. This percentage suggests a higher level of specialization for shell-oriented languages.

For what concerns non-IoT projects, the files modified by a higher proportion of the contributors are by far JavaScript and Go. The rest of the files are modified by a dramatically lower proportion of contributors. Moreover, shell-oriented files (e.g., sh files) in non-IoT projects are modified by a significantly lower proportion of contributors, in comparison with IoT projects. However, we must clarify that Figure 5 does not represent an overall ranking of the most used programming languages among IoT and non-IoT projects. Instead, it corresponds to the programming language whose files are modified by a higher percentage of contributors, among the repositories that we analyzed. For instance, although Go is the second programming language modified by a high percentage of contributors, it is present in just three IoT projects and five non-IoT projects, in both cases with around half of the files.

3) EVOLUTION OF FILES BY PROGRAMMING LANGUAGES

Figures 6a and 6b aim at visualizing the files modified in the commits, grouped by their format. To facilitate the interpretation, the dates of the commits, from all the analyzed projects,

were normalized and placed on a common timeline since the first commit to the data extraction date. Moreover, as the modifications to the files from the analyzed projects sum up to approximately 0.6 million diffs in IoT projects, and 3 million in non-IoT projects, and larger projects have a significantly higher number of commits, we decided to randomly sample 500 modifications, at most, per each project. In this manner, we guaranteed that the graph could be readable and balanced concerning the represented number of modifications from each project. Otherwise, there would be so many points that it would not be possible to identify the trends, and most of them would belong to the larger projects.

This visualization of the modifications in the commits by files format allows observable trends concerning the frequency of the changes to be identified. This chart indicates that compiled and interpreted programming languages are continually modified along the IoT projects lifetime, while shell-oriented languages are rarely modified. Thus, the commits over time are consistent with the specialization trends by language (Figure 5), the presence of the programming languages and the primary programming languages (Figures 4 and 3). This shows that developers focus more on source code concerning the business logic of the application rather than the execution scripts.

Regarding non-IoT projects, JavaScript files are evidently the most modified over time, no matter in which project they were used (e.g., user interface frameworks, general purpose libraries, MVC frameworks, runtime engines, programming frameworks). Other types of files evolved equally, with no evident differences, across the various development phases.

RQ1: How developers of IoT vs. non-IoT OSS applications contribute to their projects regarding the programming languages that they adopt? IoT projects present contributions in diverse programming languages, without a unique widely used language. In IoT projects, in addition, the files modified by a higher proportion of contributors are Java, C, C++, Python, and JavaScript. Additionally, Shell executable files, Batch files, and Command files are manipulated by a percentage that reaches, on average, 15% of the contributors. The above indicates a more variegate usage of programming languages and a higher level of specialization in shell-oriented languages than in non-IoT projects. Concerning files' evolution over time, compiled and interpreted programming languages are continually modified along the IoT projects lifetime, while shell-oriented languages are rarely modified. This is less visible for non-IoT projects.

C. RQ2: MATURITY OF THE IoT SOFTWARE ECOSYSTEM

To investigate the maturity of the IoT software ecosystem for answering RQ2, we explored the dependencies of each project and identified how many they are and which ones are present in the various projects. Initially, we relied on the GitHub API to extract the data about dependencies. However, in this case, the data provided by the API is not completely accurate because GitHub is not able to identify the dependencies of a project if they are not defined in one of the supported manifest file types.[4] Moreover, these manifests are limited to a reduced set of supported languages, namely Java, JavaScript,.NET, Python, and Ruby. For this reason, we had to manually explore each project looking for the files where dependencies are specified along with their versions.

When manually looking for the dependencies, we first tried to find the equivalent to the manifest file in the project root directory. If such a manifest did not exist, we proceeded to examine the content of the files, through the GitHub search engine, looking for keywords that could help us to identify the files in which dependencies could have been declared. Concretely, the query keywords were: `dependencies`, `deps`, `dev-deps`, `import`, `include`, `require`. Furthermore, to identify the dependency's corresponding repository on GitHub, we also used as a query keyword the substring "`github.com/`". In that case, the search could highlight the URL within GitHub of the declared dependencies. Unfortunately, this strategy was not always effective, particularly in the largest projects where the query retrieved thousands of source code files, most of which contained the keywords inside documentation blocks. When we were able to find one or more dependencies, we added them to the data gathered with the GitHub API; otherwise, we assumed that the project under analysis did not have any explicit dependency.

Afterwards, the API data and the data gathered manually were consolidated, and the analysis was performed taking into account two conditions: (i) dependencies had to correspond to open source software projects so that we could explore and analyze them, (ii) the dependencies declared directly in the analyzed project, only, were included: *dependencies of the dependencies* were excluded from the analysis. Consequently, the number reported in the *# Dependencies* column in Tables 1 and 2, corresponds to the number of dependencies that could be correctly identified either via the API or manually, and that satisfy the just described conditions. For this reason, we must clarify that zero dependencies reported in the table does not necessarily imply that, in practice, the concerned project does not have any dependencies at all.

Regarding the number of dependencies, we observe that developers of non-IoT projects adopt more dependencies than those working on IoT projects. Specifically, IoT projects exhibited 1,084 dependencies, compared to 1,868 dependencies for non-IoT projects (1.7×). In addition, the number of dependencies shared among different repositories is significantly higher in non-IoT projects. Accordingly, Figure 7 shows the percentage of dependencies present in a given number of projects. In both cases, the majority of the dependencies are not shared, but while in the non-IoT projects the percentage of dependencies shared by 2 or more projects is approximately 35%, in IoT projects is around 5%.

[4]https://help.github.com/articles/listing-the-packages-that-a-repository-depends-on/, last visited on June 6, 2019

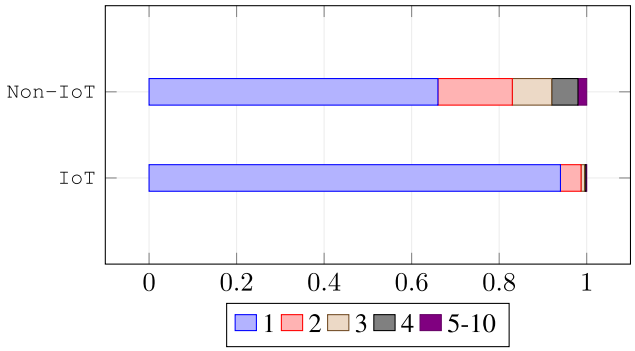


FIGURE 7. Distribution of dependencies present in one or more projects.

TABLE 3. Most popular dependencies of IoT projects.

Dependency	# Repos.	Description
mochajs/mocha	5	Test framework for nodejs
petkaantonov/bluebird	4	Promise library for nodejs
eslint/eslint	4	Linting utility for JavaScript
gruntjs/grunt	3	JavaScript task runner
substack/minimist	3	JavaScript argument parser
substack/node-mkdirp	3	Recursively `mkdir` for nodejs
kelektiv/node-uuid	3	RFC UUIDS generator
request/request	3	HTTP request client
sinonjs/sinon	3	Test framework for JavaScript
shama/gaze	2	File system watcher wrapper
jashkenas/underscore	2	Utility library for JavaScript
eclipse/paho.mqtt.python	2	MQTT Python client library
requests/requests	2	HTTP library for Python
pyserial/pyserial	2	Serial port access library
numpy/numpy	2	Scientific computing package

TABLE 4. Most popular dependencies of non-IoT projects.

Dependency	# Repos.	Description
isaacs/node-glob	11	glob implementation in JavaScript
eslint/eslint	10	Linting utility for JavaScript
isaacs/rimraf	10	`rm -rf` utility for nodejs
chalk/chalk	9	Terminal string styling utility
lodash/lodash	9	JavaScript utility library
Microsoft/TypeScript	8	Superset of JavaScript
substack/minimist	8	JavaScript argument parser
substack/node-mkdirp	8	Recursively `mkdir` for nodejs
npm/node-semver	8	Semantic versioner for npm
acornjs/acorn	7	JavaScript parser
rollup/rollup	6	Module bundler for JavaScript
sinonjs/sinon	6	Test framework for JavaScript
mishoo/UglifyJS2	6	JavaScript beautifier toolkit
mathiasbynens/he	6	HTML entity encoder/decoder
browserify/resolve	6	`require.resolve()` implementation

Finally, Tables 3 and 4 present the list of the top-15 most popular dependencies among IoT and non-IoT projects, respectively. By analyzing the type of the dependencies, it can be highlighted that most of the dependencies of non-IoT projects correspond to *utilities* aimed at easing code development, such as parsers, test frameworks, beautifiers, and algorithm implementations. In the IoT projects, instead, some of the most popular dependencies concern network protocols client libraries, HTTP requests libraries, a serial port access library, and a test framework. A few dependencies were common across IoT and non-IoT projects, and they are utilities mainly concerning code source code formatting, linting, and testing.

RQ2: How developers exploit dependencies to reuse features in IoT vs. non-IoT OSS projects? Non-IoT projects have more dependencies than IoT projects (1.7×). Moreover, the number of shared dependencies is significantly higher for non-IoT projects. Although in both of them, IoT and non-IoT projects, most of the dependencies were not shared among different projects, in non-IoT projects the percentage of dependencies shared by 2 or more projects is approximately 35%, while in IoT projects is around 5%. Finally, the most popular dependencies in the analyzed IoT projects were shared at most by 5 projects, and among these popular dependencies, there were network protocols client libraries, HTTP requests libraries, a serial port access library, and a test framework. Among the most popular non-IoT projects, instead, dependencies mainly concerned utilities aimed at easing code development.

IV. DISCUSSION AND IMPLICATIONS

After presenting the results of our analysis, in this section we focus on (i) a discussion of the results and on (ii) an analysis of the implication that our work has both for researchers and practitioners.

A. DISCUSSION

Our results showed a number of points to be further highlighted and discussed, in particular:

The development of IoT applications is different. While the knowledge about an inherent complexity in developing IoT applications was already hinted in the literature (e.g., [6], [8], [11]), we evaluated this complexity in a more quantitative way. We observed that developers, involved in the creation of IoT vs. non-IoT software applications, are less oriented towards the adoption of a lead programming language, but they work with different programming languages, according to the task at hand or to the specific capability of the infrastructure (e.g., a micro-controller or a cloud service) where the IoT application should be deployed. Furthermore, this heterogeneity of languages is also reflected in the IoT projects' topics, thus unveiling one of the main sources of complexity in IoT applications development, i.e., the co-existence of various kinds of devices, protocols, and architectures within the same application. The tools and methodologies to support IoT developers can not, therefore, be constrained to a given technological stack but they should be language and platform agnostic.

Specialization of a few contributors towards command-line scripting. The percentage of contributors that modified

specific files and the tracking of the commits over the lifetime of IoT projects showed that a strong majority of the developers are frequently modifying the files written in compiled and interpreted programming languages, where the business logic of the application reside, while a few contributors specialize in shell-oriented languages (e.g., bash), generally related to the configuration and deployment of the software components in a particular execution environment. Indeed, differently from non-IoT projects, shell-oriented languages are present in most of the IoT projects. This result reveals that, in IoT projects, the execution environment is particularly relevant yet problematic for what concerns the different (and often incompatible) target devices.

The way files evolve is different. We observed the files evolution during the history of software projects. IoT developers focus more on compiled and interpreted programming languages (i.e., Java, C, C++, Python, and JavaScript) able to fulfill the core business logic of the IoT application. All these files evolved equally across the various development phases, while shell-oriented files are scarcely modified. IoT developers seems not to focus on configuration and deployment scripts, probably immutable once the target platform(s) is chosen. Conversely, non-IoT developers constantly and significantly evolve the JavaScript files of their applications, only, being they user interface frameworks, general purpose libraries, MVC frameworks, runtime engines, or programming frameworks. Other types of files evolved equally, with no evident stops, across the various development phases.

Dependencies are considered differently. Non-IoT projects have more dependencies than IoT projects, and 35% of those dependencies are shared among 2 or more non-IoT projects. IoT developers do not only use less dependencies, but such dependencies are also shared among fewer projects, with only 5% of them shared by two or more repositories. However, dependencies in non-IoT projects mainly represent utilities, while dependencies in IoT projects are more varied and oriented towards software integration tasks. The relatively high number of dependencies used by IoT projects may entail a relatively good maturity of the IoT ecosystem, but the analysis also highlight some issues in sharing the knowledge about the existence of a given dependency.

B. IMPLICATIONS

The aforementioned findings have a number of implications for researchers and practitioners. Researchers should acknowledge the specificity of this domain, and explicitly consider *IoT-oriented* software engineering as a study branch. More specifically:

1) **IoT-oriented tools and methodologies.** Given the wide heterogeneity of IoT applications and adopted programming languages, stemming from both the results and the literature, tools like Integrated Development Environments (IDEs) and software methodologies to support IoT developers should be language and platform agnostic, and not constrained to any given technological stack. In addition, research could focus on ways to abstract this heterogeneity, to allow developers to more easily share their IoT-related efforts, code, and documentation.
2) **Supporting automation for multiple and diverse deployment targets.** The specialization towards shell-oriented languages and their relative immutability, generally related to the configuration and deployment of the software components in a particular execution environment or embedded device, may indicate that execution environments are particularly relevant for IoT development. Research efforts should consider approaches to deal with this devices heterogeneity and to automate the generation and execution of deployment commands across several, often incompatible, devices.
3) **IoT-specific dependencies sharing mechanisms.** Our results report that developers exploit some existing dependencies in their projects, but the same projects do not present common dependencies. Likely, this is due both to the heterogeneity of the IoT projects and to the relatively new and not yet consolidated software community behind those projects. This represents an opportunity for researchers for the definition of novel mechanisms that IoT developers can adopt to make their code more extensible, modular, and reusable, given the peculiarities of the deployment platforms.

Practitioners need to find appropriate ways to handle and share dependencies, as well as to create a more focused software community around these topics. Finally, confirming previous insights in the literature, our results suggest that IoT software development requires skills and expertise in several and disparate domains, differently from those required by the development of traditional software. Developers are indeed called to be more creative and able to adapt to different contexts and programming environments. Thus, it would be beneficial for students to have dedicated courses (e.g., similar to the courses reported in [12]) where they could gather these skills to approach the development of IoT applications.

V. THREATS TO VALIDITY

A. SAMPLE VALIDITY

The selection criteria of the analyzed projects aimed to be as neutral as possible from our appreciations. For this reason, we only relied on their number of stars, prioritized them accordingly, and took the 60 top starred ones. Additionally, their IoT and non-IoT nature were determined by the topics that the project owners assigned them. Since tags are freely added by project owners, this might have excluded some potentially interesting IoT projects from our analysis. The only two interventions of our criteria consisted of excluding projects that were not software related or without an open source license. Nevertheless this selection procedure, unintentionally, resulted in a strong shift in the non-IoT projects towards web-related frameworks. However, we opted to keep

this selection criteria because, on the one hand, it was replicable and transparent, and on the other hand, it reveals GitHub users trends about their interests.

On the other hand, the inclusion of the most starred projects spontaneously resulted in a significant number of files, commits, and an active contributors community. According to Kalliamvakou *et al.* [10], these variables help to avoid *perils* while performing software engineering research on GitHub. Moreover, we took inspiration from the methodology adopted by Pascarella *et al.* [13]. Authors included the same number of projects in their comparative analysis of video games and non-video games OSS projects.

B. FILE CLASSIFICATION VALIDITY

We relied on the statistics provided by the GitHub API concerning the percentage of programming language on each project. As already mentioned, this measure is calculated by GitHub using the open source Linguist library, which we assume, provides accurate statistics. However, we could asses the accuracy of such statistics later when computing the percentage of contributors working on a given programming language. We locally cloned each project and, with a text mining tool developed by us, we processed the commits to extract the files modified by each contributor and observed that the results delivered by our tool were consistent with the percentages retrieved through the API.

C. DEPENDENCIES IDENTIFICATION

The GitHub API retrieves the number and list of dependencies if they are defined in one of the supported manifest file types, only. These types are only attached to Java, JavaScript,.NET, Python, and Ruby projects. Therefore, to avoid inconsistencies in the analysis of ecosystem maturity, we had to manually explore each project looking for the files where software dependencies and their versions are specified. This manual process, given its complexity, could have lead to omissions or mistakes in the identification of the dependencies.

Finally, the higher number of dependencies in the non-IoT projects could depend from the nature of these projects: they are homogeneous in web development and a large number of them have the same primary language (i.e., JavaScript). Given these conditions, it is logical that non-IoT projects share more dependencies among them than IoT projects, which are more heterogeneous.

VI. RELATED WORK

This work lies in the software engineering domain and is intended to provide insights into the peculiarities of IoT development in the OSS context. To the best of our knowledge, no other research aimed at exploring and analyzing how developers work within several OSS IoT projects. Indeed, various authors have pointed out the need for research on software engineering for IoT systems in view of the several challenges that the development of such systems poses. In the following we approached the related work from two areas: the needs and challenges of software engineering in the IoT context, and Software Mining research in other fields different from IoT.

According to Morin *et al.* [14], IoT applications have two main characteristics from a software engineering viewpoint. The first is their distribution over a large range of processing nodes. The second is high heterogeneity of the processing nodes and the protocols used between them. To deal with these characteristics, authors introduce a modeling language aligned with UML, an advanced multiplatform code generation framework, and a methodology specifying the development processes and tools used by both IoT service developers and platform experts.

Similarly, Čolaković and Hadžialić [15] hold that IoT software architectures and frameworks are necessary to overcome the inherent complexity of IoT systems and to provide an environment for services composition. In their opinion, IoT software platforms should be created as an Open Application Platform to enable modular design as well as providing an open API (Application Programming Interface) that would easily integrate sensors and other devices.

On the basis that IoT applications have been based on fragmented software implementations for specific systems and use cases, Weyrich and Ebert [16] propose the use of reference architectures as a mean to facilitate interoperability, simplify development, and ease implementation.

According to Larrucea *et al.* [11], no consolidated set of software engineering best practices for the IoT has emerged yet. On the author's words, *"IoT landscape resembles the wild west, with programmers putting together IoT systems in ad hoc fashion"*. They consider that industry needs guidance to engineer the new generation of scalable, highly reactive, often resource-constrained software systems characteristic of the IoT. Among such guidance, authors remark the need for a new generation of development environments and the training of the new generation of IoT software developers.

Patel and Cassou [17] draws attention to the lack of a software engineering methodology to support the entire IoT application development life-cycle, which results in highly difficult to maintain, reuse, and platform-dependent design. To deal with such difficulty, authors introduce a development methodology for IoT application development, based on model-driven development and involving sensor network macroprogramming techniques.

Regarding IoT projects in OSS, Taivalsaari and Mikkonen [18] hold that nowadays nearly all the component areas of a typical IoT cloud back-end architecture can be constructed from open source technologies. On their opinion, given the availability and maturity of open source components, the role of back-end developers today could be characterized more as software composition or orchestration instead of traditional software development.

Concerning sotfware mining, as mentioned before, the methodology followed in this work took inspiration from the work of Pascarella *et al.* [13], in the video games OSS context. The authors conducted a study on 60 projects,

and their results confirmed the existence of significant differences between game and non-game development, in terms of how project resources are organized and in the diversity of developers specializations. Another source of inspiration was the work of Ray *et al.* [19]: they performed a large scale study on GitHub about the of programming languages type and use on software quality. They examined the interactions of language, domain, and defect type through a combination of regression modeling, text analytics, and visualization. Their results suggested that strong typing is modestly better than weak typing, and among functional languages, static typing is also somewhat better than dynamic typing. However, authors point out that effects arising from language design are overwhelmingly dominated by the process factors such as project size, team size, and commit size. Additionally, they determined that the defect proneness of languages, in general, is not associated with software domains.

VII. CONCLUSION

IoT software development is known to differ from the development of other kinds of applications. It poses several challenges and requires expertise in various areas due to the diverse features that IoT applications expose. In this article, we provide empirical insights into the peculiarities of IoT software development through the analysis of OSS projects. This analysis was structured around two criteria: the behavior of the contributors, and the maturity of the IoT software development ecosystem. Specifically, we conducted an exploratory study mining 30 popular IoT OSS and 30 popular non-IoT OSS projects available on GitHub. Our results are intended to provide evidence about IoT development characteristics (such as the distribution of programming languages, the specialization of contributors, the evolution of the files, and the adopted dependencies), that should be considered by future research efforts aimed at better satisfying software engineering needs in the IoT scenario.

REFERENCES

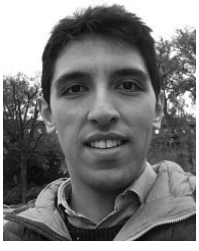
[1] J. A. Stankovic, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.
[2] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of Things: Vision, applications and research challenges," *Ad Hoc Netw.*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012.
[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.
[4] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.
[5] S. M. Riazul Islam, D. Kwak, M. Humaun Kabir, M. Hossain, and K.-S. Kwak, "The Internet of Things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015.
[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.
[7] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.
[8] A. Taivalsaari and T. Mikkonen, "A roadmap to the programmable world: Software challenges in the IoT era," *IEEE Softw.*, vol. 34, no. 1, pp. 72–80, Jan. 2017.
[9] H. Borges and M. T. Valente, "What's in a GitHub Star? Understanding repository starring practices in a social coding platform," *J. Syst. Softw.*, vol. 146, pp. 112–129, Dec. 2018.
[10] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining Github," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*. New York, NY, USA: ACM, 2014, pp. 92–101.
[11] X. Larrucea, A. Combelles, J. Favaro, and K. Taneja, "Software engineering for the Internet of Things," *IEEE Softw.*, vol. 34, no. 1, pp. 24–28, Jan. 2017.
[12] F. Corno and L. De Russis, "Training engineers for the ambient intelligence challenge," *IEEE Trans. Educ.*, vol. 60, no. 1, pp. 40–49, Feb. 2017.
[13] L. Pascarella, F. Palomba, M. Di Penta, and A. Bacchelli, "How is video game development different from software development in open source?" in *Proc. 15th Int. Conf. Mining Softw. Repositories (MSR)*. New York, NY, USA: ACM, 2018, pp. 392–402.
[14] B. Morin, N. Harrand, and F. Fleurey, "Model-based software engineering to tame the IoT jungle," *IEEE Softw.*, vol. 34, no. 1, pp. 30–36, Jan. 2017.
[15] A. Čolaković and M. Hadžialić, "Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues," *Comput. Netw.*, vol. 144, pp. 17–39, Oct. 2018.
[16] M. Weyrich and C. Ebert, "Reference architectures for the Internet of Things," *IEEE Softw.*, vol. 33, no. 1, pp. 112–116, Jan. 2016.
[17] P. Patel and D. Cassou, "Enabling high-level application development for the Internet of Things," *J. Syst. Softw.*, vol. 103, pp. 62–84, May 2015.
[18] A. Taivalsaari and T. Mikkonen, "On the development of IoT systems," in *Proc. 3rd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Apr. 2018, pp. 13–19.
[19] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in GitHub," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: ACM, 2014, pp. 155–165, doi: 10.1145/2635868.2635922.



FULVIO CORNO (Member, IEEE) has been the Leader of the e-Lite Research Group, since 2002, where he focuses on ambient intelligence systems by integrating novel interaction modalities with the IoT architectures. He is currently a Full Professor with the Department of Control and Computer Engineering, Politecnico di Torino. He is a member of IEEE Computer Society and ACM.



LUIGI DE RUSSIS (Member, IEEE) has been an Assistant Professor with the Department of Computer and Control Engineering, Politecnico di Torino, since 2018. His current research focuses on human–computer interaction, with an interest on how to overcome interaction challenges in complex settings, such as within the IoT systems. He is a member of IEEE-HKN, IEEE Computer Society, and ACM.



JUAN PABLO SÁENZ (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Computer and Control Engineering, Politecnico di Torino. His current research focuses on software engineering, with an interest on development tools and methodologies for the IoT systems. He is a Student Member of ACM.