When Code Meets Things: The FLOSS Side of IoT Systems

Igor Muzetti Pereira
igormuzetti@ufop.edu.br
Federal University of Ouro Preto
João Monlevade, Minas Gerais, Brazil

Tiago Garcia de Senna Carneiro
tiago@ufop.edu.br
Federal University of Ouro Preto
Ouro Preto, Minas Gerais, Brazil

Eduardo Figueiredo
figueiredo@dcc.ufmg.br
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil

Abstract

The growing adoption of Internet of Things (IoT) systems in domains such as smart cities, industry 4.0, and embedded devices poses new challenges to software engineering. This study aims to characterize Free/Libre and Open Source Software (FLOSS) projects related to IoT, focusing on Real-Time Operating Systems (RTOS) and Development Tools (DT). We analyzed 25 GitHub repositories using quantitative metrics grouped into popularity, repository activity, and community attributes. Through K-Means clustering and correlation analysis, we identified four main project clusters and their distinctive traits. Results indicate that most FLOSS IoT systems projects rely on C/C++, suitable for resource-constrained environments. Strong positive correlations between Forks–Issues and Watchers–Stars suggest that more popular projects foster greater community engagement. RTOS projects show deeper collaboration, while DT projects exhibit broader popularity. These findings provide insights into the collaborative dynamics and sustainability of FLOSS IoT ecosystems.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

Internet of Things, FLOSS, GitHub Analysis.

ACM Reference Format:
Igor Muzetti Pereira, Tiago Garcia de Senna Carneiro, and Eduardo Figueiredo. 2026. When Code Meets Things: The FLOSS Side of IoT Systems. In *Proceedings of 8th International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT '26)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The Internet of Things (IoT) is increasingly integrated into daily life through applications such as smart cities, smart homes, industry 4.0, and wearable devices [20]. IoT systems consist of connected objects that collect and process data to generate value for users [2], operating through the interaction of sensor networks and embedded system architectures [19].

The design and quality of IoT systems depend on both hardware and software development processes. Software engineering aims to improve these processes to deliver reliable, high-quality systems at sustainable costs [21]. Given that IoT-related projects impose specific constraints and differ from traditional software domains, it is essential to understand how these projects are structured and evolved as a foundation for future investigations [15].

IoT development poses distinct challenges compared to web and mobile applications, which are often implemented in high-level languages such as JavaScript, Python, Java, and Ruby [8, 13]. These projects emphasize usability and graphical interfaces, supported by environments with abundant computational resources. Conversely, IoT projects predominantly use C, C++, and Assembly—languages suited for resource-constrained devices such as microcontrollers. Consequently, IoT software development prioritizes robustness, reliability, and resilience, ensuring consistent operation under strict hardware and environmental limitations [18, 25].

To characterize the development and community dynamics of Free/Libre and Open Source Software (FLOSS) projects related to the IoT, we mined software metrics from 25 GitHub repositories. Repositories were retrieved using the GitHub API and manually validated to ensure relevance to IoT embedded software or supporting tools. Metrics were organized into 3 categories: popularity, repository activity, and community and analyzed through K-Means clustering and Spearman's correlation to identify common patterns and relationships [24]. Data collection followed the Mining Software Repositories (MSR) study.

The popularity of FLOSS projects on GitHub is influenced by organizational and technical factors, with organizational ownership, frequent releases, regular updates, and tagging practices contributing to higher visibility and engagement [5]. Additionally, the adoption of GitHub Actions has led to more streamlined contribution processes, as accepted pull requests tend to include fewer commits [16]. Overall, the pull-request–based development model typical of social coding platforms enhances community engagement by reducing entry barriers and increasing transparency, thereby motivating sustained participation and contributions [9].

The study provides an empirical characterization of FLOSS IoT-related projects on GitHub, focusing on RTOS and DT. An analysis of repositories reveals a predominance of C/C++ and identifies distinct project patterns through clustering. Strong correlations are observed between popularity and engagement metrics, indicating that more popular projects foster higher levels of community participation. The results highlight differences between RTOS projects, which exhibit deeper collaboration, and DT projects, which achieve broader popularity, offering insights for future research on sustainability and collaborative dynamics in IoT FLOSS ecosystems.

The Section 2 contains the research setup we followed that led to the results and discussion we present in Section 3. Section 4 shows the threats to validity and the mitigating actions we took. Section 5 summarizes the related work. Final considerations and future work are discussed in Section 6.

2 Research Setup

We mined software metrics from 25 FLOSS IoT systems projects. The study aimed to characterize these projects in terms of their development processes and community dynamics within distributed software ecosystems. Our main research question was: What are the characteristics of FLOSS projects for IoT-related systems?

We selected repositories using the GitHub API query[1]. Then, we manually inspected each repository to confirm its relevance, excluded non-software items (e.g., books or tutorials), and kept only projects related to IoT embedded software or tools that support their development. We considered only repositories written in English.

Repositories with more than 1,000 stars were selected because stars are widely recognized as an indicator of popularity and community interest on GitHub [5]. Most developers consider this metric before using or contributing to a project. However, this selection may introduce biases, favoring projects with active marketing, larger contributor bases, or popular languages, such as JavaScript.

The complete dataset is publicly available in our replication package[2]. Our study follows MSR best practices and aligns with the ACM Empirical Standards, incorporating methodological guidance from Barros (2021) [3].

We organized the metrics into 3 categories: Popularity (stars, forks, issues, watchers), Repository Activity (releases, commits, branches, pull requests), and Community (contributors, Truck-Factor (TF), and one-time contributors). To identify characteristic patterns across projects, we applied an unsupervised clustering approach with the K-Means algorithm and determined the optimal number of clusters using the Elbow Method. We also calculated Spearman's rank correlation coefficient to examine the relationships among metrics [24].

3 Results and Discussion

Tables 1 and 2 summarize the selected RTOS and DT projects, listing names, stars, licenses, and the top 5 programming languages. Both tables are sorted by stars.

Among the 25 projects analyzed, 17 are primarily developed in C and two in C++, confirming their focus on resource-constrained environments. The remaining 6 use JavaScript, Go, Ruby, or Java. Assembly, Python, and Shell frequently appear as secondary languages. Compared to GitHub's most popular languages (JavaScript, Python, Java, TypeScript, and C#)[3], our dataset shows a distinct profile dominated by C/C++ programming languages, consistent with IoT system demands.

Among the selected projects, both restrictive ("copyleft") and permissive licenses were identified. Copyleft licenses require that redistributed or modified versions preserve the same freedom to modify and share, while permissive ones allow derivative works under different terms, even proprietary. According to a 2022 WhiteSource study[4], permissive licenses remain dominant across open-source projects, which aligns with our sample's preference for the Apache 2.0 license.

We grouped the repository metrics into *popularity*, *repository activity*, and *community* categories. By applying K-Means and the Elbow Method [7], we identified 4 optimal clusters. The dataset contains 5 software types: RTOS (9), frameworks (9), libraries (3), programming languages (2), and IDEs (2). The *popularity* metrics exhibited the strongest relationships (Figure 1), and detailed charts for all categories appear in our replication package.

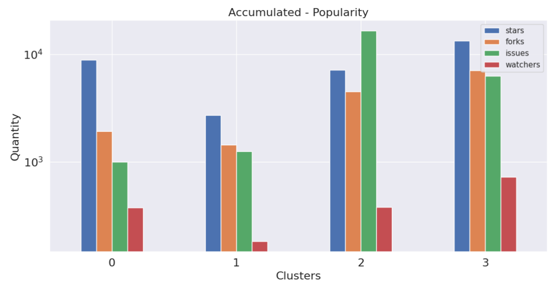

Figure 1: Average metrics by cluster.

Table 3 lists project clusters by type and category. The K-Means algorithm grouped repositories with similar popularity, repository activity, and community profiles. Cluster patterns reveal how architectural evolution, governance, and community participation differentiate projects.

The zephyrproject-rtos/zephyr project stood apart in all clusters, reflecting its exceptional popularity and adoption within the IoT ecosystem. Maintained by The Linux Foundation and supported by dozens of global organizations, Zephyr combines extensive hardware compatibility with a highly modular codebase spanning multiple architectures, boards, and subsystems. This breadth, along with strong institutional governance and an active community, distributes critical knowledge across many specialized maintainers, which explains its unusually high TF ()31).

The repositories contiki-os/contiki and contiki-ng/contiki-ng diverged only in the community category. The latter, a fork of the former, underwent significant architectural updates to integrate newer technologies. While Contiki received its last update in 2021, Contiki-NG remained active until mid-2024, illustrating how modernization and design evolution shape community engagement and participation within open IoT software ecosystems.

The FreeRTOS/FreeRTOS and aws/amazon-freertos projects shared similar clusters in most categories. Although both derive from the same RTOS base, Amazon's fork pursued integration with proprietary cloud services. The company later discontinued public development of Amazon FreeRTOS, maintaining it in read-only mode, while the community-driven FreeRTOS continued under the MIT license. This contrast highlights the impact of corporate control on openness and repository activity within FLOSS IoT platforms.

[1]https://api.github.com/search/repositories?q=topic:internetofthings,iot+stars:>1000
[2]https://zenodo.org/records/17418800
[3]https://octoverse.github.com/2022/top-programming-languages
[4]https://www.mend.io/resources/white-papers/the-complete-guide-for-open-source-licenses/

Table 1: RTOS Projects Selected for Analysis

Project name	Stars	License	Programming languages
RT-Thread/rt-thread	7895	Apache-2.0	C; Assembly; Python; C++; Shell
zephyrproject-rtos/zephyr	7165	Apache-2.0	C; Python; Assembly; Perl; Shell
RIOT-OS/RIOT	4351	GNU LGPL-2.1	C; C++; Python; Shell; Assembly
ARMmbed/mbed-OS	4274	Apache-2.0	C; Assembly; C++; Python
contiki-os/contiki	3520	3-clause BSD	C; Java; C++; Python; Assembly
FreeRTOS/FreeRTOS	3253	MIT	C; Assembly; Python
aws/amazon-freertos	2526	MIT	C; Python; Shell; Go
apache/nuttx	1226	Apache-2.0	C; Assembly; Shell; Python; C++
contiki-ng/contiki-ng	1018	3-clause BSD	C; Python; Shell

Table 2: DT Projects Selected for Analysis

Project name	Stars	License	Programming languages
Tencent/ncnn	16328	3-clause BSD	C++; C; Python; Shell
micropython/micropython	15839	MIT	C; Python; C++; Shell
esp8266/Arduino	14692	GNU LGPL-2.1	C++; C; Python; Shell; Assembly
arduino/Arduino	13360	GNU GPL-2.0	Java; Python
rwaldron/johnny-five	12908	MIT	JavaScript
espressif/arduino-esp32	10025	GNU LGPL-2.1	C; C++
espressif/esp-idf	9813	Apache-2.0	C; Python; C++; Assembly; Shell
hybridgroup/gobot	8084	Apache-2.0	Go; C
jerryscript-project/jerryscript	6470	Apache-2.0	C; JavaScript; Python; Shell; C++
hybridgroup/cylon	3956	Apache-2.0	JavaScript
adafruit/circuitpython	3417	MIT	C; Python; Shell; C++
espressif/ESP8266_RTOS_SDK	2935	Apache-2.0	C; Python; C++; Shell
jerryscript-project/iotjs	2527	Apache-2.0	C; JavaScript; Python; Shell
cesanta/mongoose-os	2356	Apache-2.0	C; Assembly; Python; C++; JavaScript
hybridgroup/artoo	1532	Apache-2.0	Ruby
arduino/arduino-ide	1287	GNU AGPL-3.0	TypeScript; JavaScript

Table 3: K-Means check

Project	Kind	General	Popularity	Delivery	Community
zephyrproject-rtos/zephyr	RTOS	2	2	1	2
ARMmbed/mbed-OS	RTOS	3	1	2	1
RIOT-OS/RIOT	RTOS	3	1	2	3
contiki-os/contiki	RTOS	0	1	3	0
contiki-ng/contiki-ng	RTOS	0	1	3	3
RT-Thread/rt-thread	RTOS	3	1	0	1
aws/amazon-freertos	RTOS	0	1	0	0
apache/nuttx	RTOS	3	1	2	3
FreeRTOS/FreeRTOS	RTOS	0	1	3	0
esp8266/Arduino	Library (DT)	1	0	0	1
arduino/Arduino	IDE (DT)	1	0	3	3
espressif/arduino-esp32	Library (DT)	1	0	3	1
arduino/arduino-ide	IDE (DT)	0	1	3	0
espressif/ESP8266_RTOS_SDK	Framework (DT)	0	1	3	0
espressif/esp-idf	Framework (DT)	1	0	3	1
jerryscript-project/iotjs	Framework (DT)	0	1	3	0
jerryscript-project/jerryscript	Library (DT)	0	1	0	0
micropython/micropython	Language (DT)	1	3	0	1
hybridgroup/gobot	Framework (DT)	0	1	3	0
hybridgroup/cylon	Framework (DT)	0	1	3	0
hybridgroup/artoo	Framework (DT)	0	1	3	0
Tencent/ncnn	Framework (DT)	1	3	3	3
rwaldron/johnny-five	Framework (DT)	1	3	3	0
cesanta/mongoose-os	Framework (DT)	0	1	3	0
adafruit/circuitpython	Language (DT)	3	1	0	0

The projects apache/nuttx and RIOT-OS/RIOT emphasize the importance of governance and licensing in sustaining open-source longevity. NuttX, once an incubated project, became an official Apache Foundation project in 2022[5], incorporating CI/CD practices and stable release processes. Meanwhile, RIOT-OS adopted the LGPL-2.1 license after community debate[6], seeking to ensure long-term openness and protection against code closure, an essential aspect for the sustainability of free IoT operating systems.

[5]https://cwiki.apache.org/confluence/collector/pages.action? key = NUTTX

[6]https://github.com/RIOT-OS/RIOT/issues/2128

The arduino/arduino-ide project is maintained by the Italian company Arduino. This project is an integrated development environment (IDE), a platform for prototyping. This repository contains the Arduino IDE 2.x source code. There is also an Arduino/Arduino repository with Arduino IDE 1.x source code. According to the company, the latest version has been rewritten, and no code from the pioneer repository has been reused. The pioneering project has more contributors overall, but none made only a single contribution. In contrast, the project under development includes a subgroup of five contributors who each made just one contribution. On the TF, the pioneer has a value of 2, and the current has a value of 1. These values caused the algorithm to classify them into 3 different clusters. May be because they do not share codes according to the README.md[7] file of the arduino/arduino-ide project. Seven other projects in the sample also have TF 1.

The espressif/ESP8266_RTOS_SDK project is also owned by the Chinese company Espressif. It is a development framework for the ESP8266 SOC. The jerryscript-project/jerryscript library is used to develop JavaScript IoT projects. It was used to develop the jerryscript-project/iotjs framework.

There are 3 projects from the same owner called hybridgroup are frameworks for IoT. What distinguishes them is their support for different programming languages. Gobot is in Golang, Cylon is for JavaScript and Artoo is for Ruby. The last project is also a framework called cesanta/mongoose-os for IoT firmware development. These last 4 projects are part of the same clusters.

Table 4 reports Spearman correlations [24] among metrics. Strong positive correlations ($\rho > 0.8$) were found for Forks x Issues (0.825) and Watchers x Stars (0.909). The Forks x Issues correlation suggests that projects with more forks tend to have more issue reports, reflecting both user engagement and community size. The Watchers x Stars correlation indicates that users who star repositories often follow them closely. Although correlations don't imply causality, they highlight interactive dynamics of active FLOSS ecosystems.

Table 4: Spearman's correlation coefficient for each category

Combination	r_s	p-value
Forks vs. Issues	0.825	0.000
Watchers vs. Stars	0.909	0.000

Figures 2 and 3 present these relationships. RTOS projects are marked with stars and DT with dots. Cluster centroids (yellow) represent typical metric patterns. Silhouette scores of 0.57 and 0.59 indicate well-defined clusters, with some overlap near boundaries.

DT projects have the highest values for Stars, Forks, and Watchers. We suspect that DT projects have many users, but these users don't necessarily contribute to the source code. In RTOS projects, their users interact more with the community and the source code repository. The number of Forks drastically influences the number of Issues. Projects that originated from a Fork of another project may have received more attention from a new group of contributors due to changes in business models or evolution strategies. Issues serve as an information exchange between parties, and the number of Issues may increase as the number of Forks of that project also increases [11].

[7]https://github.com/arduino/arduino-ide/blob/main/README.md

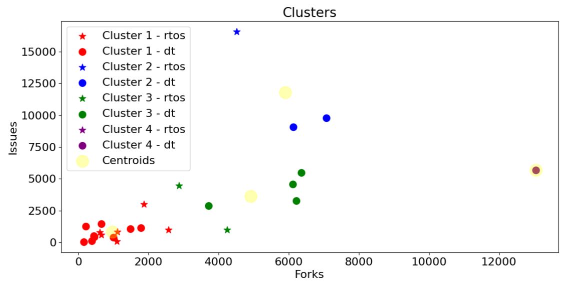

Figure 2: Scatter plot between Forks and Issues.

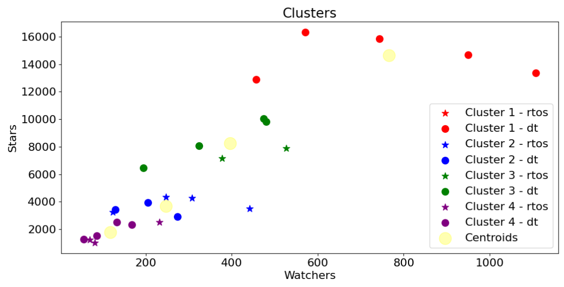

Figure 3: Scatter plot between Watchers and Stars.

A repository receives a Star from users when they want to favorite it. Usually, this occurs because the user is a contributor or uses the software at a particular time. When a user wants to understand more about the development dynamics of that software, they can become a Watcher of the repository. From then on, any movement of Issues, PR, among others, is notified. With this, he begins to have a detailed view of the project and better understands the details. It is common for this user to contribute to the project after becoming a Watcher [12]. So, a repository with many Stars can imply many Watchers.

4 Threats to Validity

Threats to external validity relate to the generalizability of our results beyond the study's scope. Since we focused on FLOSS IoT-related project repositories, our findings may not apply to other types of systems due to differences in programming languages, collaborator profiles, and domain complexity. The selection of repositories with more than 1,000 stars aimed to ensure the analysis of widely adopted and active projects, which is a common practice in studies on social and technical metrics in FLOSS ecosystems. However, we acknowledge that the number of stars is not an absolute indicator of technical quality or deployment success, as it may also reflect factors such as project visibility, marketing, or community affinity. To mitigate this bias, we rely on evidence from prior work (e.g., [5]), which shows that GitHub stars correlate with project popularity and influence, yet should be interpreted with caution.

Threats to conclusion validity concern the reasonableness of our analysis conclusions. These threats are less likely to affect the quantitative results for RQ, as they derive primarily from direct metrics and counts of repository data, where the analysis is mainly descriptive. To mitigate these threats, we employed an iterative process of analysis and discussion among multiple authors, seeking consensus on the definitions of categories emerging from the data.

5 Related Work

IoT-related projects development is different from application development for other domains. Rigid real-time requirements, the experimental work of testing software in emulators, the documentation that often needs to be formally built for critical systems, integration testing due to co-design of software and hardware, testing and production environments are often different from the development environment, are challenges in this domain and cannot be fully covered by traditional software development processes [22].

However, some studies have identified a transformation from traditional methods to an agile approach, where both types are found: organizations with 2 kinds of processes operating simultaneously and even having models to be used as a reference for a trans- formation [4, 17]. Some studies investigated the use of agile methods in the embedded context, the most common evidence being the use of practices and techniques from Scrum and Extreme Program- ming [1, 23]. We noticed in these studies that practices more focused on project management are the most used, but pair programming and test-driven development have also been widely discussed.

6 Conclusion and Future Work

This study empirically characterizes FLOSS projects in the IoT domain by analyzing 25 GitHub repositories to examine their organization, evolution, and community dynamics, with a focus on RTOS and development tools. Using data mining, clustering, and correlation analysis, the study identifies distinct patterns of popularity, activity, and engagement, highlighting the prevalence of C/C++ and strong positive relationships between key metrics (e.g., Forks x Issues and Watchers x Stars) as indicators of collaborative ecosystems. The findings show that differences in software type are associated with varying levels of collaboration intensity and suggest that organizational models, governance, and licensing play a significant role in sustaining community participation and project longevity, offering insights for both practitioners and researchers interested in scalable and sustainable IoT FLOSS ecosystems.

Future work includes expanding the dataset to incorporate IoT platforms and repositories from alternative hosting services, such as GitLab and Bitbucket. Another promising direction is integrating qualitative analysis, through interviews or surveys, with the quantitative insights obtained here, allowing for a deeper understanding of the motivations and challenges faced by FLOSS IoT communities.

References

[1] Tom Holvoet Karel De Vlaminck Andrew Wils, Stefan Van Baelen. 2006. Agility in the Avionics Software World. In *Extreme Programming and Agile Processes in Software Engineering*. 123–132.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.

[3] Daniel Barros, Flavio Horita, Igor Wiese, and Kanan Silva. 2021. A Mining Software Repository Extended Cookbook: Lessons learned from a literature review. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering (SBES)*. 1–10. doi:10.1145/3474624.3474627

[4] Vebjorn Berg, Jorgen Birkeland, Anh Nguyen-Duc, Ilias O. Pappas, and Letizia Jaccheri. 2020. Achieving agility and quality in product development - an empirical study of hardware startups. *Journal of Systems and Software* 167 (2020). doi:10.1016/j.jss.2020.110599

[5] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software* 146 (2018), 112–129. doi:10.1016/j.jss.2018.09.016

[6] Martin Host Magnus C. Ohlsson Bjorn Regnell Claes Wohlin, Per Runeson and Anders Wesslen. 2000. *Experimentation in Software Engineering: An Introduction.* Kluwer Academic.

[7] Pratap Dangeti. 2017. *Statistics for Machine Learning: Techniques for exploring supervised, unsupervised, and reinforcement learning models with Python and R.*

[8] Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. 2022. On the Use of GitHub Actions in Software Development Repositories. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 235–245. doi:10.1109/ICSME55016.2022.00029

[9] Alberto Bacchelli Georgios Gousios, Margaret-Anne Storey. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*. 285–296. doi:10.1145/2884781.2884826

[10] Lise Tordrup Heeager and Peter Axel Nielsen. 2020. Meshing agile and plan-driven development in safety-critical software: a case study. *Empirical Software Engineering* 25 (2020), 1035–1062. doi:10.1007/s10664-020-09804-z

[11] Jiahuan He Xin Xia Pavneet Singh Kochhar Jing Jiang, David Lo and Li Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22 (2017), 547–578. doi:10.1007/s10664-016-9436-6

[12] Eirini Kalliamvakou Daniela Damian Jyoti Sheoran, Kelly Blincoe and Jordan Ell. 2014. Understanding "watchers" on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (WSR)*. 336–339. doi:10.1145/2597073.2597114

[13] Timothy Kinsman, Mairieli Wessel, {Marco A.} Gerosa, and Christoph Treude. 2021. How do software developers use github actions to automate their workflows?. In *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 420–431. doi:10.1109/MSR52588.2021.00054

[14] Surapon Kraijak and Panwit Tuwanut. 2015. A survey on internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends. In *2015 IEEE 16th International Conference on Communication Technology (ICCT)*. 26–31.

[15] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A Survey of DevOps Concepts and Challenges. *ACM Computing Survey* 52, 6 (2019). doi:10.1145/3359981

[16] Marco A. Gerosa Christoph Treude Mairieli Wessel, Joseph Vargovich. 2023. GitHub Actions: The Impact on the Pull Request Process. *Empirical Software Engineering* 28 (2023). Issue 6. doi:10.1007/s10664-023-10369-w

[17] P. Manhart and K. Schneider. 2004. Breaking the ice for agile development of embedded software: an industry experience report. In *International Conference on Software Engineering (ICSE)*. 378–386. doi:10.1109/ICSE.2004.1317460

[18] Iori Mizutani, Ganesh Ramanathan, and Simon Mayer. 2022. Integrating Multi-Disciplinary Offline and Online Engineering in Industrial Cyber-Physical Systems through DevOps. In *Proceedings of the 11th International Conference on the Internet of Things (IoT*. 40–47. doi:10.1145/3494322.3494328

[19] Rebeca Campos Motta, Valéria Silva, and Guilherme Horta Travassos. 2019. Towards a more in-depth understanding of the IoT Paradigm and its challenges. *Journal of Software Engineering Research and Development* 7 (2019), 3:1 – 3:16. doi:10.5753/jserd.2019.14

[20] Elisa Yumi Nakagawa, Pablo Oliveira Antonino, Frank Schnicke, Thomas Kuhn, and Peter Liggesmeyer. 2021. Continuous Systems and Software Engineering for Industry 4.0: A disruptive view. *Information and Software Technology* 135 (2021).

[21] Igor Muzetti Pereira, Tiago Garcia de Senna Carneiro, and Eduardo Figueiredo. 2021. Understanding the context of IoT software systems in DevOps. In *IEEE/ACM 3rd International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT)*. 13–20. doi:10.1109/SERP4IoT52556.2021.00009

[22] Jussi Ronkainen and Pekka Abrahamsson. 2003. *Software Development under Stringent Hardware Constraints: Do Agile Methods Have a Chance?* 73–79. doi:10.1007/3-540-44870-5_10

[23] O. Salo and P. Abrahamsson. 2008. Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *IET Software* 2 (2008), 58–64. Issue 1. doi:10.1049/iet-sen:20070038

[24] C. Spearman. 1904. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology* 15 (1904), 72–101.

[25] Fiorella Zampetti, Damian Tamburri, Sebastiano Panichella, Annibale Panichella, Gerardo Canfora, and Massimiliano Di Penta. 2023. Continuous Integration and Delivery Practices for Cyber-Physical Systems: An Interview-Based Study. *ACM Trans. Softw. Eng. Methodol.* 32, 3 (2023). doi:10.1145/3571854