COSMOS

DevOps for Complex Cyber-physical Systems

Project Number 957254

D3.2 Catalogue of good and bad practices of DevOps for CPS

Version 1.0
22 December 2021
Final

Public Distribution

University of Sannio

Project Partners: **Aicas, Delft University of Technology, GMV Skysoft, Intelligentia, Q-media, Siemens, Siemens Healthcare, The Open Group, University of Luxembourg, University of Sannio, Unparallel Innovation, Zurich University of Applied Sciences**

Project Partner Contact Information

Aicas James Hunt Emmy-Noether-Strasse 9 76131 Karlsruhe Germany Tel: +49 721 663 968 0 E-mail: jjh@aicas.com	**Delft University of Technology** Annibale Panichella Van Mourik Broekmanweg 6 2628 XE Delft Netherlands Tel: +31 15 27 89306 E-mail: a.panichella@tudelft.nl
Intelligentia Davide De Pasquale Via Del Pomerio 7 82100 Benevento Italy Tel: +39 0824 1774728 E-mail: davide.depasquale@intelligentia.it	**GMV Skysoft** José Neves Alameda dos Oceanos Nº 115 1990-392 Lisbon Portugal Tel. +351 21 382 93 66 E-mail: jose.neves@gmv.com
Q-media Petr Novobilsky Pocernicka 272/96 108 00 Prague Czech Republic Tel: +420 296 411 480 E-mail: pno@qma.cz	**Siemens** Birthe Boehm Guenther-Scharowsky-Strasse 1 91058 Erlangen Germany Tel: +49 9131 70 E-mail: birthe.boehm@siemens.com
Siemens Healthineers David Malgiaritta Siemensstrasse 3 91301 Forchheim Germany Tel: +49 9191 180 E-mail: david.malgiaritta@siemens-healthineers.com	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of Sannio Massimiliano Di Penta Palazzo ex Poste, Via Traiano I-82100 Benevento Italy Tel: +39 0824 305536 E-mail: dipenta@unisannio.it	**University of Luxembourg** Domenico Bianculli 29 Avenue J. F. Kennedy L-1855 Luxembourg Luxembourg Tel: +352 46 66 44 5328 E-mail: domenico.bianculli@uni.lu
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	**Zurich University of Applied Sciences** Sebastiano Panichella Gertrudstrasse 15 8401 Winterthur Switzerland Tel: +41 58 934 41 56 E-mail: panc@zhaw.ch

Document Control

Version	Status	Date
0.1	Document outline	15 October 2021
0.2	Introduction and Design draft	1 November 2021
0.3	Results draft	18 November 2021
0.4	First full draft	12 December 2021
0.5	Further editing draft	14 December 2021
0.8	Updates from reviewers	21 December 2021
1.0	Final QA Version	22 December 2021

Table of Contents

Executive Summary

This deliverable shed light on the way developers set and maintain a CI/CD pipeline for CPS development, focusing more on the challenges and barriers they encounter, together with the mitigation strategies mainly adopted to overcome (i.e., to deal with) them. Specifically, starting from the transcripts of ten semi-structured interviews conducted with 10 organizations covering eight different domains, we applied a card sorting procedure to identify (i) the state of the practice in terms of CPS development process and usage of the CI/CD pipeline for it, (ii) a set of challenges and barriers faced by the interviewees when setting and maintaining a pipeline specific to the CPS context, and (iii) a set of mitigation strategies used to overcome them.

The relations between challenges/barriers and mitigation have been validated by an internal survey shared with the same companies participating in the semi-structured interviews. Furthermore, to verify the set of challenges and barriers, we used an external survey asking developers working in the CPS domain, whether or not they encountered each challenge/barrier, as well as we let them provide other challenges that did not come from the previous phase.

As the last step, we manually labeled a sample of 364 pull requests belonging to ten CPS open-source projects hosted on GitHub to check what are the challenges/barriers discussed in the open-source domain, as well as the way open-source developers act to deal with them. By doing this, it is possible to identify similarities and differences in terms of challenges/barriers together with their mitigation strategies in industry and open-source.

The main outcomes of the deliverable are: (i) a list of challenges and barriers for CI/CD pipeline setting and evolution in the CPS domain, together with (ii) a list reporting the mitigation mainly used to overcome them.

1 Introduction

Cyber-Physical Systems (CPSs) comprise heterogeneous software and hardware components interacting with each other. They aim at automating operations in different domains, such as automotive, aerospace, healthcare, or railways. As it happens for any software system, CPSs continuously evolve to cope with new customer requirements and technology changes. However, CPSs require a tailored development and operation (DevOps) process and are more challenging to evolve than conventional software [31, 43, 61, 62].

In such a context, adopting effective Continuous Integration and Delivery (CI/CD) practices off the DevOps menu is extremely relevant and appropriate for setting the execution environment, e.g., Hardware-in-the-Loop (HiL) or simulators. CI/CD has been found effective in introducing several advantages in software development, e.g., the reduction of release cycles and the early discovery of defects [64], although its application implies overcoming barriers and challenges [10, 32].

While for conventional software systems good and bad practices for applying CI/CD have been defined [17, 71], for what concerns CPSs, the practice is still immature to be able to do so. At the same time, it is possible, by interviewing professionals or by observing existing systems, to understand what are the challenges, barriers, and related mitigation when using CI/CD pipelines for CPSs. For this reason, in this deliverable we will mainly focus on the analysis of challenges, barriers and related mitigation strategies, although, wherever possible, we also identify and discuss bad practices.

In general, existing CI/CD technology cannot be applied to CPSs as is [35]. For example, CPSs demand suitable Verification & Validation (V&V) techniques, and the interaction with HiL, or the need to replace them with suitable mock-ups or simulators, would make the application of CI/CD challenging at best.

This deliverable starts by analyzing the development practices of 10 companies developing CPSs in 8 different domains. The aim is to understand how and why they apply CI/CD, how they plan to improve it, and the barriers and challenges encountered, together with the mitigation strategies applied to overcome them, focusing on three aspects of CI/CD for CPSs, namely (i) the pipeline setting, (ii) the phases involved (e.g., static analysis, testing, delivery, etc.), and (iii) the usage and configuration of simulators and/or HiL. After that, the deliverable continues by looking at challenges, barriers and their mitigation in a sample of 364 pull requests coming from 10 CPS open-source projects adopting a CI/CD pipeline.

The elicited set of challenges, barriers and mitigation strategies are impactful by providing insights to project leaders and developers, guiding them to configure CI/CD pipelines for CPSs, but also to staff projects and acquire equipment. Furthermore, they identify areas where further research is required. These include domain-aware decision making, the integration of simulators and HiL in the pipeline, but also evergreen software engineering topics like test flakiness and automation.

2 Background

This section discusses the literature related to (i) CPS development leveraged for the inception of our study, (ii) CI/CD process, and (iii) CI/CD good and bad practices.

2.1 Development of Cyber-Physical Systems

CPSs are more complex and difficult to design, develop, test, and integrate than conventional software systems [31, 43, 61, 62]. For instance, Giraldo et al. [28] highlighted challenges for CPS development and evolution such as the lack of CPS development tools, and the limited availability of hardware versions.

Törngren et al. [62] investigated how CPSs' engineering deals with the complexity of CPS design, and of the environment in which CPSs operate. In this context, it is of paramount importance to perform run-time verifi-

cation of safety requirements [27], as well as testing encapsulating model-in-the-loop (MiL) [55], software-in-the-loop (SiL), and hardware-in-the-loop (HiL) [2].

Considering the costs, risks, and complexity of conducting system testing in a real environment [13, 36], simulation is becoming one of the cornerstones in developing and validating CPSs. CPS developers mainly rely on basic simulation models [29, 56], as well as rigid-body [42, 72] and soft-body simulation environments [25, 51]. The usage of CPS simulation environments enables automated test generation and execution. However, the limited budget allocated for testing activities and the virtually infinite testing space pose challenges for adequately exercising the CPS behavior [4, 21, 69].

This deliverable complements previous studies by looking at the challenges, barriers, and benefits of applying CI/CD to support the development, V&V, and evolution of CPSs.

Related to DevOps applications in a CPS context, Ugarte Querejeta et al. [63] and Park et al. [47] analyzed the use and challenges of the digital twin to enable DevOps approaches for cyber-physical production systems to continuously improve them. Specifically, Park et al. identified challenges related to (i) discrepancies between models and their physical counterparts, (ii) integration between heterogeneous models due to the complexity of CPSs, and (iii) security issues due to the tight coupling between the digital twins and the physical environment. Instead of only looking at automating the production process, this deliverable focuses more on the CI/CD process for CPS development and evolution.

Mårtensson et al. [44] identified factors to consider for applying CI to software-intensive embedded systems such as complexity of user scenarios, compliance to standards, long build times, security, and test environments. These factors represent real impediments for companies who want to adopt CI for embedded systems. Our study is wider than Mårtensson et al. (10 vs. 2 companies), and while the above factors are still present in our results, our study deepens the analysis on different CI/CD aspects (e.g., setting, phases, and execution environment) for CPSs.

Table 1 summarizes the main challenges emerging during CPS development as stated in previous literature that are used to drive our study, although we do not focus on specific implementation details of simulators. However, we leverage the challenges identified by the aforementioned studies to devise the interview guide, in particular those related to (i) the complexity of the underlying environment, (ii) certification and compliance to standards, (iii) test automation, (iv) testing of safety requirements, and (v) MiL, HiL, and simulators.

2.2 CI/CD process

Hilton et al. [33] found that CI is becoming very popular in open source projects. The latter is also true in industry, even if Ståhl and Bosh [58, 59] found that there is not a uniform adoption of CI in industry. Furthermore, Vasilescu et al. [64] showed that CI practices improve developers' productivity without negatively impacting the overall code quality. Finally, Ståhl et al. [60], in a study involving 3 companies, found that the lack of traceability may prevent the application of CI in conventional software systems.

From a different perspective, Elazhary et al. [20] looked at the extent to which companies follow the CI practices by Fowler and Foemmel [22] through interviews. Their results emphasized differences among companies in terms of repository structure, testing automation, long build, and deployment challenges. While we share some goals with Elazhary et al., our study, and the dimensions being investigated, relates to CI/CD application for CPS development. Furthermore, Elazhary et al. [19] used grounded theory to investigate human factors in CI. Even if our study considers human factors, it is not focused on that.

Vassallo et al. [67] investigated, by surveying developers of a large financial organization, the adoption of the CI/CD pipeline during development activities, confirming what is known from existing literature (e.g., the execution of automated tests to improve the quality of their product), or confuting them (e.g., the usage of refactoring activities during normal development).

Deepening the continuous delivery practice, Chen [11] analyzed four years' CD adoption in a multi-billion-euro company, and identified a list of challenges related to CD adoption. Savor et al. [53], instead, by analyzing

Table 1: Challenges in CPS development from previous literature.

Ref.	CPS-related development challenges
[28]	Limited development tools, Error-prone hardware configuration
[62]	Environment complexity, Co-designing hardware and software
[27]	Test generation/automation, verification of safety requirements
[2]	Integration of MiL, SiL, and HiL
[36]	Where is testing performed (HiL vs. simulators)
[13, 25, 29, 42, 51, 56, 72]	Implementation of simulators
[4, 21, 69]	Simulator challenges/adequacy
[44]	Standards, long build, security, architecture, test environments of embedded systems
[47, 63]	Digital twin adoption in manufacturing and related design challenges

the CD adoption in two industrial companies, found that it does not negatively impact developer productivity even when the project increases in terms of size and complexity.

2.3 CI/CD barriers and bad practices

Different authors studied barriers and/or challenges in adopting CI/CD. These were initially identified by Duvall et al. [15], and are related to the need for maintaining a fully automated build process, handling dependencies, having different levels of builds, and coping with different target environments.

Hilton et al. [32] studied barriers developers encounter when moving toward CI: quality assurance, security, and flexibility. Olsson et al. [46], instead, looked at the challenges faced while migrating towards CD: (i) complexity of the deployment environment, (ii) need to achieve timely delivery, and (iii) lack of a complete overview of all the development projects.

Previous research also found that CI/CD may be wrongly applied, leading to bad practices. Specifically, CI/CD antipatterns have been defined by Duvall [17], and empirically elicited by Zampetti et al. [71] from interviews and Stack Overflow posts. Our study is complementary to that although, where appropriate, we compare the practices observed in our context (CPS-specific) with bad practices recommendations from previous studies.

Researchers have developed different kinds of tools with the goal of detecting and removing antipatterns from CI configuration files [24, 66], analyzing the pipeline aging by observing its execution [65], skipping builds [3], or coping with security-related issues in infrastructure-as-code [49].

Specific CPS characteristics, and related development and evolution processes imply that, as pointed out in existing books [35], the aforementioned processes and practices might not directly apply to CPS. To the best of our knowledge, there is no such broad investigation on the application of CI/CD in CPS development and evolution, as well as the challenges, barriers and benefits in applying them.

3 Study Definition and Planning

The *goal* of this deliverable is to study the CI/CD practices in the development of CPSs, to understand the obstacles in terms of challenges and barriers, together with mitigation strategies adopted to overcome them. The *context* consists of (i) 10 companies developing CPSs for eight different domains, i.e., acoustic sensors, aerospace, automotive, healthcare, identification technology, railways and robotics; and (ii) a sample of 364 pull requests out of 7,046 ones coming from 10 GitHub CPS open-source projects using a CI/CD pipeline.

The deliverable addresses the following four research questions:

- **RQ**$_1$: *What is the CI/CD process the respondents currently apply for CPS? If they do not have a fully-integrated pipeline, to what extent single phases are automated?* We investigate the CI/CD practices adopted by the interviewed companies by looking at the conditions that determine (i) the setting of the

CI/CD pipeline, e.g., whether an incremental build is used, when the build is triggered, or whether build matrices are used; (ii) the phases instantiated in the pipeline, e.g., static analysis, various testing levels, or deployment; and (iii) the use of simulators or HiL in the context of the CI/CD pipeline.

- **RQ**$_2$: *What are the challenges and barriers respondents encounter?* We investigate the challenges (e.g., the need to cope with a slow build or flakiness, or with phases not easy to automate) and barriers encountered by the interviewed companies when dealing with the setting or evolution of the CI/CD pipeline for CPS development.
- **RQ**$_3$: *What are the mitigation strategies adopted for overcoming challenges and barriers?* We look at the actions/strategies adopted within the interviewed organizations to overcome challenges and barriers. Furthermore, we validated the identified mitigation through an internal survey shared among the same companies participating in the semi-structured interviews to verify our understanding of their strategies, as well as to give them the possibility to provide mitigation not mentioned during the interviews.
- **RQ**$_4$: *What are the challenges and bad practices for CPS development in open-source projects? How are they mitigated?* In the end, by manually validating a sample of 364 pull requests from 10 CPS open-source projects, we investigate challenges and barriers together with their mitigation occurring in the open-source for what concerns the evolution of a CI/CD pipeline. The main goal is to understand whether or not industrial and open-source projects share differences and commonalities.

It is important to highlight that this deliverable takes as input the outcome of a previously provided deliverable (i.e., D3.1), aimed at interviewing companies working in a CPS context. To improve the overall readability and comprehensibility of this deliverable, in Section 3.1 we will briefly summarize the procedure used in the previous deliverable to conduct the semi-structured interviews, before going deeper on the subsequent sections on the methodology adopted to come up with a set of challenges/barriers together with their mitigation strategies.

3.1 Semi-structured interviews

Interview structure. We defined the interview structure through an iterative process, which started from the existing knowledge on the topic (see Section 2). From such knowledge, all theoretical pending points were distilled and matched with (a) interview structure areas and (b) questions for each interview structure area. As summarized in Table 2, we start with demographics about the company and interviewee, and get a first glance at development and lifecycle management practices [5] adopted in the context of interest. Then, we gather data about the pipeline structure and technology, paying particular attention to V&V and deployment. We then explore the usage of simulators and HiL. We also investigate the presence of any machine learning or (ML)-intensive components to be automated (e.g., trained/tuned) or executed by the pipeline over any CPSs software artifact, or, conversely, the use of ML and Artificial Intelligence (AI) for pipeline automation (e.g., as part of the testing oracle), i.e., AIOps [12]. The interview concludes with questions about the main benefits achieved, barriers encountered, and challenges to tackle when configuring the pipeline.

Interview participant selection. The interview participants have been selected based on personal knowledge, with the goal of identifying experienced practitioners with prime experience over the theoretical constructs (CPS pipelines) under investigation. Upon accepted invitation, we provided participants with an overview of the questions to expect in the interview, to allow them gathering any additional information.

Table 3 summarizes the companies and interviewees involved in the study. Five out of ten companies are large (i.e., over 1,000 employees), one is medium (i.e., between 50 and 1,000 employees), two are small (between 10 and 20 employees), and two are micro (less than ten employees). Furthermore, the sample covers eight different domains: aerospace, automotive, energy, healthcare, railways, robotics, identification technology (i.e., Radio Frequency IDentification – RFID), and acoustic sensors. Finally, the participants' professional experience in the CPS field varies from 3 to 25 years, with varying job titles, and all of them are currently involved in the configuration of the CI/CD pipeline.

Table 2: Interview structure

Section	Content
Overview	Company description, domain, programming languages Respondent background and role
CI/CD Pipeline Structure	Phases and steps Tools (Versioning, Build automation, CI/CD, Use of containers) Verification and Validation approaches Deployment
Simulators and HiL	Simulator development/acquisition Simulator/HiL integration in the pipeline Simulators vs HiL tradeoffs
ML-based components	In the developed software In the pipeline
CI/CD pipeline configuration	Pipeline stability Build strategies Triggering
Conclusion	Challenges Barriers Expected benefits

Conducting interviews. Interviews were conducted, using a videoconferencing system, by one researcher (with the support of one-two other researchers), following an ordering based on interviewees' availability. Before starting the interview, the interviewer recalled study goals and gathered consent for recording. The interview structure was followed rigorously, varying only the level of detail over different areas of the interview based on the provided answers. For example, if a participant mentioned the use of simulators, we asked deeper questions on the topic, while we skipped questions not applicable to a given participant. It is important to remark that, interviews are treated as independent from each other, meaning that questions were not adjusted over different interviews. This is because, as shown in Table 3, the involved organizations cover a broad range of domains, and the main goal was to achieve a similar understanding of those domains.

Creating transcripts. After interviews have been completed, a researcher transcribed the audio, creating a document organized into sections as Table 2. The transcripts contain a total of 15,329 words and 787 sentences.

3.2 Data Analysis from interviews' transcripts

Open Coding. Two researchers, expert of the domain, (hereinafter referred to as "coders"), independently used online spreadsheets to assign codes (i.e., open coding) to sentences in the transcripts. The coding was carried out following the approach illustrated by [34], i.e., annotating a code near sentences of the transcript (a process known as micro-analysis [34]). A code is defined as a mnemonic label identifying a concept defined in a text fragment, e.g., by applying the label 'TEST' to any part of text reflecting a software testing activity. Wherever appropriate, the coder added a memo that could be leveraged to better explain the observed phenomenon, as well as to identify possible relationships between codes dealing with different aspects of the CI/CD pipeline setting and maintaining.

Open coding has been performed over four subsequent sessions including two, three, four, and, finally one interview. After each coding session, the coders held a discussion meeting, in which similar codes created by multiple coders were merged, and conflicts were resolved. After each round, we computed the Krippendorff α [38] to determine the achieved level of agreement. The obtained α values for the four iterations were 0.65

Table 3: Participant Demographics

Org_{ID}	Company		Role	CPS
	Domain	Size		Exp. (Y)
Org. A	Aerospace	Small	R&D Manager	8
Org. B	Healthcare	Large	DevOps Architect	18
Org. C	Acoustic Sensors	Small	SW and HW Integrator	6
Org. D	Robotics	Medium	Team Leader	7
Org. E	Automotive	Large	R&D Manager	20
Org. F	Aerospace	Large	R&D Manager	20
Org. G	Railways	Large	SW and HW Integrator	10
Org. H	Railways	Micro	Team Leader	25
Org. I	Identification Technology	Micro	Software Engineer	3
Org. L	Energy	Large	Project Leader	5

(close to the minimum acceptability of α = 0.66 [38]), 0.71, 0.69, and 0.86 (substantial agreement). Starting from the second iteration, the coders could reuse, through a drop-down cell, codes created during previous iterations, or create new ones. To further limit agreement by chance, each code annotation was reviewed during the meetings, not just the disagreements.

During the discussion meetings, broad groups of codes were also defined. For instance, we distinguished codes belonging to the CI/CD pipeline, and those more related to the development process. Also, we started grouping codes belonging to different phases of the pipeline, and codes related to challenges, barriers, benefits, and wishes. Such a categorization started during the first discussion meeting and then was refined over the next ones. After the first two phases of the open coding (after the first phase, the set of codes was too immature for this purpose), three researchers iteratively produced the first version of a mind map grouping codes into categories. Such a mind map has been used as a support to ease the subsequent open-coding phases and to evaluate the extent to which non-leaf nodes were saturated. Note that in this study, we do not expect a full saturation [52] due to the high diversity of the considered application domains. The mind map was then refined after each subsequent coding phase.

Overall, we identified a set of 180 codes, which led to the construction of a taxonomy explaining the phenomenon, organized across 21 high-level categories.

Finally, the two coders performed three iterations over the transcripts, codes, and memos to derive relations between different codes. For instance, it is possible that process constraints (e.g., the need to use a specific type of simulator or tool imposed by the domain, or to adopt certain coding standards), introduce challenges while setting the pipeline (e.g., the need to cope with phases not easy to automate, or slow build and flakiness) that may be addressed by relying on a particular mitigation strategy (e.g., push small changes when using incremental builds). The outcome of this step results in 90 relations from 128 sentences. We will present how different codes relate to each other and are spread among different organizations through storytelling.

3.3 Evaluation of the challenges and barriers through a survey

To verify the challenges and barriers affecting the setting and evolution of the CI/CD pipelines for CPS development, we conducted a survey involving practitioners operating in the CPS domain, having a CI/CD pipeline in place in the organization they belong to. To encourage participation, we adopted the snowball [30] sampling to reach as many practitioners as possible. Specifically, we shared the survey link to some contact points, and encouraged them to indicate us, or directly share the survey link, to further participants or people better suited to participate in the survey. We followed this strategy because, while we had personal knowledge with a limited set of contacts, snowballing may help reach relevant people (i.e., those involved in CI/CD setting for CPS

Table 4: External Survey Respondents Demographics

Participant$_{ID}$	**Domain**	**Role**	**CPS Exp. (Y)**	Pipeline Setting
P_1	Robotics	R&D Manager	>10	Intermediate
P_2	IoT building automation	Project Manager	Between 1 and 5	Mature
P_3	Automotive & UAVs	R&D Manager	Between 1 and 5	Mature
P_4	Robotics	CTO	<1	Recent
P_5	HealthCare	SW and HW Integrator	Between 1 and 5	Intermediate
P_6	Oil Refinement	DevOps Architect	Between 1 and 5	Recent
P_7	Finance	DevOps Architect	Between 1 and 5	Intermediate
P_8	Automotive	SW and HW Integrator	<1	Intermediate

development), and favor participation. Furthermore, we also posted the survey on REDDIT[1] in the *Continuous Delivery*, *Continuous Integration*, *DevOps*, and *Embedded real Time* communities. These communities have been selected as they (i) allow users to post surveys, and (ii) have a large number of active subscribers, thus increasing the potential audience.

The online survey presented to the participants had:

- An introduction explaining the goal of the survey questionnaire;
- A set of six sections, each one aimed at verifying the challenges/barriers belonging to a specific category;
- A section with an open-ended question giving the possibility to describe challenges/barriers a respondent encountered that are not presented in the previous sections of the survey;
- A demographic section in which we asked the participant: the application domain, the role within the company, years of experience in CPS development, as well as information about the CI/CD pipeline (i.e., (i) whether or not the company has a CI/CD pipeline in place, (ii) years from its introduction, and (iii) how the participant interacts with it).

For each challenge or barrier belonging to a specific category (i.e., section in the survey), we asked participants whether or not they have encountered it within their team/organization. Specifically, the respondent could choose between three different options: (i) yes, (ii) no, and (iii) not applicable to our context. The latter option is aimed at accounting for cases where the challenge/barrier cannot be encountered due to the development process adopted within the organization (e.g., if an organization does not use simulators in its pipeline, it will never face problems like simulators limited in their functionality).

The questionnaire was administrated through Survey Hero[2], and we kept the questionnaire open for six weeks. Furthermore, nobody reported having particular issues (e.g., privacy issues) with the used survey administration tool.

After closing the survey, we obtained eight complete responses. As reported in Table 4, all the participants filled the demographic section. Specifically, in terms of their role within the company, 2 are R&D Manager, 2 are DevOps Architect, 2 are software and hardware integrator, 1 is Project Manager, and 1 is CTO. In terms of years of experience with CPS development, five respondents have between 1 and 5 years of experience, 2 have less than one year, and the remaining one has more than 10 years. As regards the CPS-domain they belong to, we have: robotics (3), automotive (2), IoT building automation (1), healthcare (1), oil refinement (1), and finance (1). All of them declare that their company already has in place a CI/CD pipeline used while developing CPSs (2 introduced it less than one year ago — i.e., recent in Table 4, 4 between one and five years ago — i.e., intermediate, and 2 more than five years ago — i.e., mature), and in terms of the way they interact with the pipeline, three of them only use the CI/CD pipeline, two are involved in its setting and maintaining, and three other than setting and maintaining it also use it for their development tasks.

[1] `https://www.reddit.com/`
[2] `https://www.surveyhero.com`

3.4 Evaluation of the challenges together with their mitigation through a survey involving the interviewed organizations

To verify our understanding of how the interviewed organizations act to address the challenges/barriers encountered while setting and maintaining the CI/CD pipeline for CPS development, we conducted a survey involving them.

The online survey presented to the participants had:

- An introduction explaining the goal of the survey questionnaire;
- A set of ten sections in which we validate the relations between the 10 challenges/barriers for which we found at least one mitigation strategy from the transcripts;
- A demographic section in which we asked the participant: the application domain, the role within the company, years of experience in CPS development, as well as information about the CI/CD pipeline (i.e., (i) whether or not the company has a CI/CD pipeline in place, (ii) years from its introduction, and (iii) how the participant interacts with it).

It is important to highlight that, since the main goal of the survey is to validate our correct understanding of the challenges/barriers and related mitigation strategies, we clearly asked the participants to provide their personal contacts (among them the name of the company) mainly for traceability purposes.

For each section in the survey, we start asking the respondent whether or not the challenge/barrier has been faced at least once from the team she is working with. Specifically, instead of using a yes/no question, we added a third option aimed at highlighting those cases where the challenge/barrier cannot be encountered due to the development process adopted by the organization. For instance, if an organization does not use HiL in their development process, it will never have problems due to the high cost or lack of scalability of the hardware devices. If the challenge/barrier has been encountered at least once within the organization, we list a set of questions each one aimed at investigating the adoption of the identified mitigation strategy to overcome the previously presented challenge/barrier. Specifically, the respondent could choose between three different options: (i) yes, and we used it, (ii) yes, but we never used it, and (iii) no. Two out of 17 questions dealing with mitigation strategies, provide only two options to the respondents: (i) yes, it happened, and (ii) no, it never happened. At the end of each section, we add an optional free comment field where the respondent could provide additional mitigation strategies adopted for overcoming the related challenge/barrier.

The questionnaire was administrated through Survey Hero, and we kept the questionnaire open for six weeks. Furthermore, nobody reported having particular issues (e.g., privacy issues) with the used survey administration tool.

After closing the survey, we obtained eight responses from seven organizations involved in the semi-structured interviews. Specifically, organizations C, H, and L did not fill in the survey, while in organization E we obtained two different responses, even if one of them did not provide demographic information. Moreover, among the seven respondents providing their personal contacts, only one has also participated in the semi-structured interviews. Among the seven participants who filled the demographic section, 3 are R&D Manager, 2 are DevOps Architect, 1 is a software and hardware integrator, and 1 is a DevOps QA Engineer. In terms of years of experience with CPS development, three respondents have between 1 and 5 years of experience, 2 between 5 and 10, and the remaining 2 have more than 10 years. All of them declare that their company already has in place a CI/CD pipeline used while developing CPSs (1 introduced it less than one year ago, while 5 between one and five years ago), and in terms of the way they interact with the pipeline, among the 5 participants who answered this question, 1 only uses the CI/CD pipeline, 2 are involved in its setting and maintaining, and 2 set, maintain, and use it for their development tasks.

Table 5: CPS-related open source projects used for the qualitative study

Project	# Forks	Size	# Contr.	Topic(s)	# PRs
PX4/PX4-Autopilot	10,830	130,184	514	drone, drones, ros	11,598
arduPilot/ardupilot	10,615	206,090	602	drone, robotics, ros	12,693
bulletphysics/bullet3	2,050	275,904	207	robotics	1,685
cartographer-project/cartographer	1,833	5,915	49	robotics, self-driving	1,277
carla-simulator/carla	1,696	197,768	115	autonomous-vehicles, autonomous-driving, slef-driving-car, ros	1,066
cleanflight/cleanflight	1,383	362,328	345	embedded	1,385
paparazzi/paparazzi	985	80,009	113	drones	2,035
nasa/fprime	762	265,168	103	embedded, embedded-systems	521
mavlink/mavros	732	4,339	102	ros	409
cyberbotics/webots	712	3,895,048	75	autonomous-vehicles, robot, robots, robotics, ros	2578

3.5 Pull requests mining and analysis for challenges, barriers and mitigation in open-source

Context Selection. The study context for answering RQ_4 is made up of 364 closed pull requests (PRs), sampled from an initial set of 7,046 candidate PRs, which have been extracted from 10 CPS-related open-source projects hosted on GitHub and written in two different programming languages, i.e., C and C++, the ones most popular for CPS development as also highlighted from the ten semi-structured interviews.

To identify CPS-related projects, we used the GitHub query search feature. Since the goal is to identify projects belonging to different CPS domains, we used the following query

```
https://api.github.com/search/repositories?q=topic:TOPIC+language:
LANGUAGE
```

to download the whole set of C and C++ projects having a topic in the following set: *automotive, autonomous-driving, autonomous-vehicles, cyber-physical-systems, drone, drones, embedded, embedded-systems, robot, robotics, ros, self-driving-car, self-driving cars*.

Since the search has been conducted by topic and considering that a project may have more than one topic, we have removed from the total number of projects duplicated instances. Furthermore, we sorted the projects using the number of forks as a proxy for their popularity, whereas we excluded projects:

- being forked from others;
- not adopting a CI/CD infrastructure;
- having less than 100 commits;
- having less than 50 pull requests;
- having less than 50 contributors.

Finally, we considered the top-ten ranked projects for the analysis. Table 5 reports for each selected project, the number of forks, the size, the total number of contributors, the set of topics belonging to the project among the ones used for the search, and in the last column, we also report the total number of closed pull requests. After that, we cloned the projects' repository locally and used Perceval [14] to retrieve their closed PRs. We filtered out the pull requests with an empty description, since the goal was to identify, starting from the discussion, possible challenges dealing with the CI/CD pipeline setting and maintenance. After that, for each closed pull request, we used the GitHub API to retrieve the set of commits belonging to it together with the path of the files they impact. Through regular expressions applied to the file names, we filtered out all pull requests that do not change the pipeline configuration files, such as.yml files and .xml files dealing with simulators' configuration. As a result, we ended up with 7,046 candidate PRs to inspect.

Qualitative Analysis of CI/CD challenges and mitigation strategies. To identify what are the challenges, barriers and bad practices encountered by open-source developers when using a CI/CD pipeline for CPS development, together with their mitigation, we manually analyzed a statistically significant sample of 364 randomly

Table 6: # of PRs in the sample scattered across the 10 CPS open-source projects

Project	# PRs after the filtering (%)	# sampled PRs
PX4/PX4-Autopilot	2,881 (40.89)	149
paparazzi/paparazzi	1,827 (25.93)	94
arduPilot/ardupilot	759 (10.77)	39
cyberbotics/webots	695 (9.86)	36
carla-simulator/carla	282 (4.0)	15
cleanflight/cleanflight	213 (3.02)	11
cartographer-project/cartographer	141 (2.0)	7
nasa/fprime	138 (1.96)	7
mavlink/mavros	74 (1.05)	4
bulletphysics/bullet3	36 (0.52)	2
TOTAL	7,046 (100)	364

selected pull requests (confidence interval $\pm$ 5%, confidence level 95%) from the 7,046 candidate ones. Table 6 reports, for each project, the total number of closed PRs after the filtering, together with the number of PRs being manually analyzed. It is important to highlight that, the PRs have been selected proportionally to the overall number of PRs belonging to each project. For instance, after the filtering, MAVLINK/MAVROS ended up with a total of 74 candidate PRs (1.05% of the total candidate PRs), so we randomly sampled only 4 PRs from this project (1.05% of 364).

The manual analysis was conducted to classify each pull request as (i) *no*: the pull request does not discuss issues for setting or maintaining the pipeline; and (ii) *yes*: the pull request points out the presence of a challenge developers have encountered during the CI/CD pipeline evolution. All the cases labeled as *yes* have additionally been categorized to discriminate between cases where (i) the challenge is CPS-specific, e.g., inclusion of HiL or setting of the simulators for the hardware devices, or (ii) the challenge may also occur when maintaining a pipeline for traditional software development, e.g., the build succeeds when a failure/error is thrown. Furthermore, independently of whether or not the challenge is specific to CPS development, each pull request has been further categorized with a label describing the challenge/bad practice, as well as the mitigation strategy adopted to overcome it. The labels were created by following a card-sorting procedure, and specifically a cooperative (multiple annotators) hybrid card-sorting (partial set of predefined categories) [57].

We started from a predefined list of categories related to (i) bad practices in setting, using, and maintaining CI pipeline [71], (ii) challenges and barriers obtained as a result of the coding of the interviews' transcripts together with their mitigation, and (ii) a set of restructuring actions applied during CI/CD pipelines evolution [70]. The pull requests have been validated independently by two coders who could assign one of the previously defined categories, or add a new one when needed. The pull requests to inspect were organized into a Google sheet.

At the end of the labeling process, an open discussion was performed by adding a third coder. In the open discussion, we checked all the pull requests for which there were disagreements among the two coders (231 out of 364). Note that, based on the labeling procedure, we could have different types of disagreements: (i) cases where one annotator labeled the PR as discussing an issue, while the other did not find it; (ii) cases where the two coders disagree regarding whether the discussed problem is CPS-specific or not; (iii) cases where the coders assigned different challenges; and (iv) cases where the coders disagreed about the mitigation strategy used to overcome the problem.

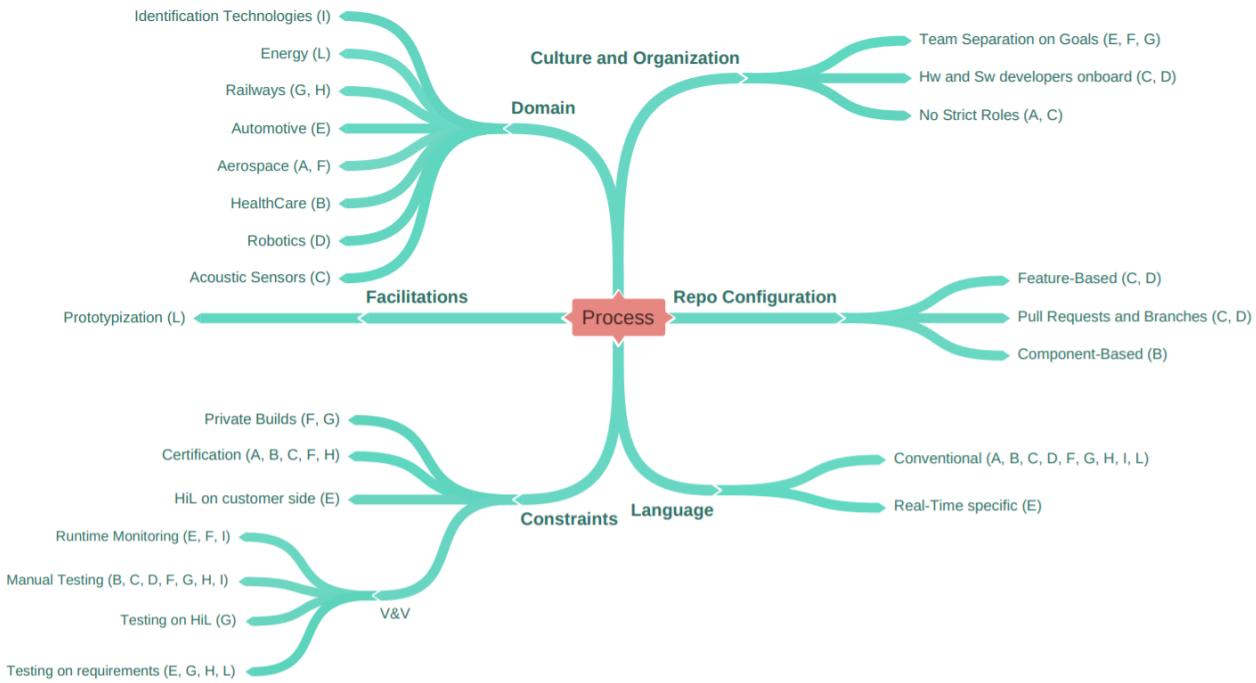

Figure 1: Overview of the CPS process adopted by the 10 interviewed organizations

4 Study Results

4.1 RQ_1: *What is the CI/CD process the respondents currently apply for CPS? If they do not have a fully-integrated pipeline, to what extent single phases are automated?*

The transcripts of our interviews together with the labeling procedure helped us forming organization profiles, focusing more on how they set and maintain a pipeline for CPS development. Figure 1 shows the characterization of the CPS development process of the ten interviewed organizations, while Figure 2 reports the characterization of the CI/CD pipeline adopted within them for CPS development. Each leaf corresponds to a specific "code" obtained by coding the interviews' transcripts, together with the set of companies belonging to it (in parenthesis). Based on the codes, the first research question aims at summarizing the development process of the interviewed organizations, focusing more on the status of their CI/CD pipeline in terms of (i) build triggering strategies (e.g., continuous or periodic), (ii) co-existence of multiple pipeline configurations (e.g., for different devices) and the frequency of changes occurring to them, (iii) the phases being automated/executed within the pipeline, and (iv) the usage and setting of HiL and simulators within the pipeline (e.g., a pipeline can be a mix of different environments used in different circumstances). In the following sections, for each organization participating in the semi-structured interviews, we summarize, by using the codes in Figure 1 and Figure 2, the overall CPS development process, together with the adopted CI/CD pipeline.

4.1.1 Organization A

Organization A is involved in verification and validation tasks for the aerospace domain (i.e., on-board software for satellites) where the scope of the pipeline is only for V&V. The latter is because, due to the safety integrity level of the CPS, the development and the V&V teams and pipelines must be kept distinct. For what concerns the programming language being adopted, A relies on conventional language dictated by standards in the aerospace domain (*"We mainly use ANSI C-99 following the MISRA rules"*). The latter, also intro-

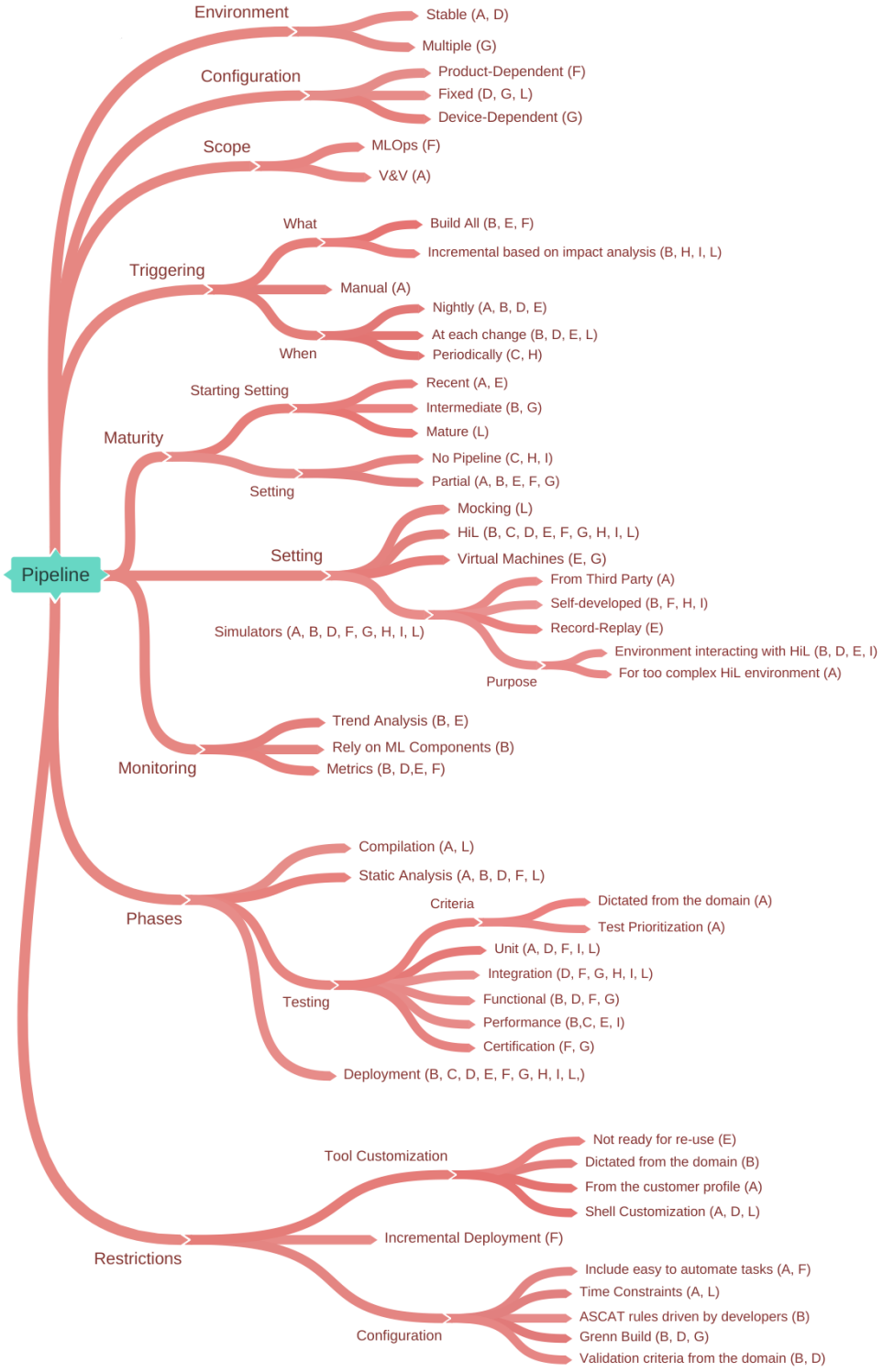

Figure 2: Overview of the CPS pipeline adopted by the 10 interviewed organizations

duces a process constraint related to the need for certifying software (i.e., follow the Motor Industry Software Reliability Association (MISRA) standards [1, 7]), as also found by Mårtensson et al. [44].

Organization A has started adopting CI/CD practices less than one year ago, mainly due to a limited culture within the team about CI/CD principles. Moreover, it does not have a strict separation of roles for what concerns the type of interaction with the pipeline (*"a developer who needs to customize a CI/CD pipeline by simply using yaml files can customize it directly"*). An interesting point to note is that, A does not rely on build matrices with jobs related to different environment variants since that *"the pipeline does not have to change/evolve based on the changes in the technologies being used (version for compilers and or programming languages) since that the aerospace domain follows the waterfall process... So everything is [frozen]: no changes may occur later on in the process."*

Moving the attention on the type of tasks automated within the pipeline, due to the application domain and the related standards and certification constraints, the pipeline compiles the software provided and developed by the customer, relies on SonarQube for (i) checking the fulfillment of the MISRA rules for certification, (ii) identifying maintainability problems (i.e., *"among the requirements we receive we also have non-functional requirements expressed in terms of rules available in SonarQube"*) mainly related to the presence of duplicated code, and (iii) spotting bugs as soon as they are introduced, and executing the unit tests and tests for robustness to *"check how the system behaves/reacts in the presence of unexpected inputs ([e.g.,] inputs having values out of the admissible range)"*. Furthermore, considering the overall scope of the pipeline, its triggering is manual, even if there are also nightly builds used in the presence of test suites requiring a long time to complete. Furthermore, A has to consider time constraints for the pipeline setting to deal with possible issues that may arise when launching the simulator (e.g., memory leaks or impossibility to access the simulator). Finally, the testing criteria to derive the test cases to include within the pipeline are expressed from the customer as non-functional requirements (*"i.e., use MC/DC for deriving the test suite"*).

Organization A cannot involve HiL in the pipeline (i.e., requires a clean room not accessible from the outside), so it only relies on third-party simulators. Moreover, A cannot choose the simulators since the customer follows *"a framework for simulation aimed at hosting different simulators for different satellite models for the digital twin of the satellite"*. Relying on third-party simulators helps in reducing the costs/efforts needed to develop the simulators from scratch, as well as, helps in guaranteeing the trustworthiness of the outcome being produced and provided to the customer. Of course, the level of trustworthiness increases for those cases where the simulator is provided by the same vendor of the hardware device that must be simulated.

4.1.2 Organization B

B is a large organization involved in the healthcare domain: it provides Computed Tomography (CT) scanners for clinical use. Nowadays, CT scanners are growing in complexity in terms of the need for interconnecting them with other systems, both product- and organization-related. In terms of development process, B has a team for each component being developed (i.e., around 17 different teams working on 70 branches), together with an integration branch where all the other branches are integrated into a *"single joined point"*. Furthermore, each team adopts conventional programming languages for CPS development, i.e., mainly C# and C++. Organization B already has a CI/CD pipeline in place for CPS development that has been introduced 4 years ago, that is in a continuous improvement state. Finally, based on its application domain, B is constrained to *"follow medical application frameworks providing a base set of rules in terms of how to build applications and how to integrate them"*. The latter requires the adoption of processes aimed at verifying whether or not the overall development process adheres to the regulatory standards for developing medical applications.

For what concerns the build triggering strategies, B adopts both incremental (i.e., rolling) and nightly builds. Of course, the tasks involved in the two different types of builds, as well as, the execution environment involved in them vary. Specifically, nightly builds make use of HiL indeed the pipeline has automated deployment on a "real" CT Scanner *"without reusing existing artifacts while building all of them from scratch in a clean environment"*, for executing the whole test suite in a real production environment. Note that, when talking

about "real" CT Scanner, organization B refers to *"physical systems that are equivalent to the real hardware in the CT Scanner but not connected to anything around it which has a simulator running on it"*. Furthermore, differently from incremental builds, nightly builds run three different types of testing, namely unit/component, sub-system and system testing. To provide developers fast feedback about the impact of their changes, B relies on incremental builds executing only a subset of the whole set of functional tests — by doing *"impact based testing to figure out the impact of the changes and select the tests to be executed based on the impact."* To control the overall build execution time, B encourages developers to push small changes leading to *"small sets of tests to be executed."* Finally, both incremental and nightly builds run static code analysis tools mainly aimed at identifying maintainability and security flows in the code.

Going deeper on the way organization B checks the fulfillment of non-functional requirements for the system under development, it is possible to state that, starting from 2 years, there is a specific type of build aimed at checking performance requirements like *"test whether each component (some components) stays within the resource limits they are assigned to"*. The outcome of the execution of the performance builds is compared over time to identify and monitor possible performance degradations within the whole system. Moreover, B has a specific DevOps team for checking the fulfillment of security requirements, even if this is not done continuously while only *"near the finalization of the product"*, and is not automatized. It is clear that, both the triggering strategies adopted by organization B and the tasks being automatized within each type of build influence the choice between using simulators and/or HiL. For what concerns the simulators, B relies on self-developed simulators — there are a suitable knowledge and skills to properly develop simulators, i.e., B develops both the software and the hardware. However, at the moment B does not use simulators (*"mainly used for functional testing only"*) for checking non-functional (i.e., performance) requirements.

Finally, one interesting point to highlight for organization B is that the way tools are used in the pipeline, as well as, the way tools should be customized is highly dependent on the specific technology imposed by the domain (i.e., *"the Windows situation does not help us with dockerization and our application strategies are not yet towards isolated aspects to be dockerized or containerized."*)

4.1.3 Organization C

Organization C is involved in CPS innovation activities for industry, among others the development of the SPL Noise Meter Board, i.e., a low-cost, high quality, electronic sensing board capable of measuring noise of the environment. It does not have any separation of roles between the members of the team (*"The team is the company."*), however, the team is composed of both software and hardware experts who work together, simplifying the overall development process, in particular for those activities requiring the integration and communication between software and hardware components (*"Useful for sensors' integration... [it] help[s] having knowledge about the hardware components, how they work and how it is possible to communicate with them."*)

C adopts a pull-request (PR) development process with one branch per feature (i.e., *"Several branches for maintaining and developing different features."*) Furthermore, even if it does not have strict guidelines in terms of coding standards, C attempts to adopt similar coding styles within each branch. Finally, it relies on a conventional programming language for CPS development, i.e., Python for testing and C or C++ for micro-controllers development.

At the moment, C does not have a CI/CD pipeline for CPS development. However, the deployment is fully-automated, while the testing is still manual mainly due to the impossibility of automating the oracle specification, in particular for testing acoustic signals. Furthermore, even if C does not have certification constraints for the developed code, there exist certification constraints that need to be satisfied *"for the acoustic signals"*.

Finally, C only uses real hardware devices, even if within the organization there is the wish of including simulators in the process to test the acoustic signal (i.e., the main outcome of their product) in a controlled environment, i.e., *"removing noise from the surrounded environment"*.

4.1.4 Organization D

Organization D is involved in the development of autonomous robots, and is made up of several teams of developers where each team accounts for both hardware and software developers. Furthermore, it adopts a pull-request (PR) development process with one branch per feature (i.e., *"We have a branch for each feature that needs to be implemented and/or improved and we use PRs to merge the work in the stable release branch."*) Based on the application domain, i.e., robotics, it mainly adopts C++, together with Python for users' interfaces and for interacting with the hardware devices.

D has a fully dockerized pipeline for CPS development. It relies on continuous and nightly builds, even if they are not used for running time-intensive tasks — *"(that is not so expensive in terms of execution time)"* — while for running regression testing activities on already packaged components and for deployment to the customers. Focusing the attention on the tasks automatized while building PR changes, our interviewee mentions the execution of static code analysis tools to advertise the original developers about possible code quality degradation, and unit tests relying on simulators, i.e., ROS[3]. Only when the PR is peer-reviewed and there are no failures in the entailed CI/CD process, it is possible to merge the change on the stable repository and enact a release process for shipping the product to the customers. D stores all the information coming from the execution of the build process into a database to monitor the overall quality of the development process in terms of static analysis metrics and code coverage from the unit test execution. Furthermore, the application domain does not introduce certification constraints, i.e., *"If you want to sell a robot you do not need to have a certified robot"*, while it hinders the automation of non-functional testing within the pipeline. Specifically, our interviewee mentions the manual execution of reliability and safety test such as *"running the robot a long time with a guy supervising the test execution to understand when and why the robot starts to not work anymore"* or *"guaranteeing that once pressing the stop button the robot actually shuts down"*.

As regards the execution environment being used within the pipeline, D relies on simulators and HiL. Furthermore, the CI/CD configuration is pretty stable meaning that, even if each branch may rely on a customized CI/CD process, the configuration does not have to change over time.

Finally, one interesting point raised by our interviewee is related to the partial usage of Docker on the hardware so that it is possible to run the robot in a privileged mode and switch between software versions quite easily: *"each one may choose the version of the software that has to be run over the robot."*

4.1.5 Organization E

E is a large company operating in the automotive domain where *"the transformation from the "Engine"-focused car to the "Electric" and "Software"-focused user experience drive platform is just on the verge."* It is organized into three different teams each one with a specific goal: *"one working on virtual machines, one working on web services because we provide DevOps solutions for embedded systems, and finally, we have a small team working on customer delivery with the goal of adapting our tools to the customers' needs."* Furthermore, when talking about programming languages, it is the only organization relying on real-time languages, i.e., Real-time Java allowing to cope with scheduling requirements of embedded systems. However, it already has a CI/CD pipeline in place mainly for deployment purposes (*"we are able to support updating software on devices on the fly"*), even if it is working on improving the status of the pipeline — *"it is still kind of an infancy we are still working on improving"*.

Moving on to the build process, E uses the pipeline mainly for deployment. However, differently from other organizations, E relies on virtual machines instead of using containers for several reasons: (i) better control over the resources (i.e., *"the ability to enforce our resources usage inside the virtual machine while you do not have quite the same extent with a container"*), (ii) versioning capability, i.e., *"when a new service comes in, you register it so it is easy to start a new version of this service, run down the old one and switch over*

[3] https://www.ros.org/

the new one during run-time", and (iii) memory safety guarantee, i.e., *"By looking at a recent post by both Google and Microsoft we found that around 70% of the security violations are due to failures of memory safety. So by using a garbage-collected environment, we can prevent those issues from occurring."* Going deeper on how the deployment process works, organization E first creates the virtual machine, (i.e., emulating the virtual environment), then the OSGi that goes on the platform, and finally tests individual modules. The latter means that E does not test all the developed modules together since *"we deploy individual bundles to a platform."*

For what concerns the verification of non-functional requirements, E executes security and performance testing, even if they are not included in the pipeline. Specifically, for real-time systems it is important to monitor the impact of each change on performance properties to be able to identify, as soon as possible, the change introducing performance degradation —- i.e., *"we have various performance tests that we run regularly to track our performance as the system evolves."* Since E develops software for embedded entertainment in the automotive domain, the HiL is only available for a final validation on the customer's side: *"then employ our customer for the last mile"*, so most of the work is done relying on virtual environments.

Finally, E stresses the traceability problem that is highly important for CPS development: *"only [a] few people do requirement-based testing which is required for every certification regime"*. The latter raises several challenges that need to be addressed such as the complexity in integrating automated traceability tools within the pipeline.

4.1.6 Organization F

Similarly to organization A, F operates in the aerospace domain, and is mainly involved in the development and refining of the routing algorithm for the Free Route Airspace (FRA). For what concerns the programming language being adopted, F relies on conventional language — *"Java is used for front-end development together with Javascript. C and C++ [are] used for the back-end."*

F already has a CI/CD pipeline mainly for deployment purposes, that is under continuous improvements. Moreover, our interviewee mentions that the pipeline is more an MLOps than a simple DevOps pipeline. Among the phases being automatized, there are (i) execution of static code analysis tools for identifying maintainability flows and spotting bugs as soon as they are introduced, (ii) unit testing, (iii) integration testing and (iv) deployment. Furthermore, the execution of non-functional testing activities are mainly done manually and outside the pipeline due to the high complexity of the real-time operating system under development. Similarly to organization A, it is required that the developed code satisfies strict certification requirements that are mainly checked by relying on code coverage tools. Differently from other organizations, F does not rely on nightly builds, meaning that also time-intensive tasks are executed at each change, i.e., *"even the slow builds are continuously built."* Finally, F recommends developers to use private builds before pushing their changes on the stable release branch at least for what concerns the execution of unit testing.

Moving the attention on the execution environment, F relies on both simulators and HiL, however, based on the safety integrity level of the application domain, *"we do not have simulators and HiL in the same pipeline mostly for certification issues."* Specifically, it is possible to rely on real devices only when there is enough trustability about the software in terms of correct behaviour, as well as the absence of crashes gained by relying on self-developed simulators. Finally, the pipeline provides a monitoring mechanism for what concerns aspects of the real-time operating system such as scheduling and memory that *"gives us the possibility to collect feedback/evidence that may help us in obtaining the certifications."*

4.1.7 Organization G

Organization G is involved in delivering software for railways, i.e., Train Control Management System (TCMS), and similarly to what reported for the aerospace domain, due to the safety integrity level of the software under development, developers and testers must be different (i.e., *"Testers and Developers are in separate teams in presence of new functionality to be implemented both start together to implement and write test*

cases.") G already has a CI/CD pipeline in place for CPS development — *"introduced two years ago"* — that, at the moment, is in a continuous improvement state since that it does not automatize the whole development process (i.e., *"The deployment on the real train or on the hardware test track is not automated at the moment even if we are working on making it automatic."*) In terms of programming language for CPS development, the interviewee mentions the need of adapting the programming language to the device on which the software has to be executed, however they mainly rely on conventional languages.

Based on the application domain, G adopts staged builds following the "green-build rule". In the first stage, the build process is executed on a virtual machine — *" virtual train, software running on a PC that should behave like it does on a real train"*. Specifically, each device/component has a proper CI/CD configuration. Once a change occurs on a specific component the related build process is activated and in presence of a green status, all the components are deployed together so that it is possible to enable the execution using the virtual train (*"the devices are run in some kind of containers and we have frameworks building and connecting the whole set of devices and components."*). If the build process ends with a successful state, it is possible to move to the next stage that relies on the hardware test track — i.e., *"where we have the whole set of devices and even some more that we do not have in the virtual train."* Finally, if the build process for the second stage ends with a green status, it is possible to run the last stage relying on a real train. Based on what was reported before, it is clear that G adopts btoh simulators and HiL even if they are used in different stages of the build process.

For what concerns the type of tasks automatized within the pipeline, G uses the pipeline to automatically test basic functionality (i.e., *"We test specific train functionality such as whether we should activate the train in [a specific] mode"*), as well as, the interaction between different components/devices (i.e., *"we have a long sequence of events for each test that involves different devices and components so we are mainly doing integration testing."*). Note that the test suites used in different stages of the build process may be different since *"for some test cases, we are not allowed to rely on the virtual environment while we must consider hardware track or real train."* At the moment, G has automated deployment within the pipeline only for the first stage, i.e., relying on the virtual train (for which the overall build execution is *"around one hour and [a] half"*), while it is done manually for what concerns the other two stages: hardware test track and real train. Another task that is not automatized within the pipeline is the one aimed at checking the fulfillment of non-functional requirements. The latter is mainly due to the high variability and complexity of the environment.

Going deeper on how developers interact with the CI/CD pipeline, G enforces developers to run private builds before pushing their changes on the main stable repository. The private builds are aimed at executing the whole test suite running within the pipeline — *"for the moment we cannot configure the number and type of tests to be executed locally"*. Furthermore, the "green-build rule" is used for determining the development tasks: *"in presence of a failure all developers are stopped until the build becomes green again."*

4.1.8 Organization H

The application domain of H is railway, specifically the development of a specific component used for transmitting data between on-board and ground applications. At the moment, it counts less than 10 employees in total and, due to the limited availability of human resources together with the complexity and the safety integrity level of the application domain, it does not have a CI/CD pipeline in place for CPS development. Furthermore, H uses C and C++ (i.e., conventional programming languages) for CPS development, and it has strict constraints for what concerns the production code that has to satisfies strict certification requirements, as well as the compliance with the railway standards and specifications. The latter is mainly checked by relying on *"a specific complex tool that can be configured ... based on ... specifications and standards."*

Differently from the other organizations, H has a little level of automation included in the development process. Specifically, only the adherence to standards and specifications is automatized within the process, while functional test cases are written and executed manually by developers — *"tests are written manually starting from requirements and system specification[s] but also their execution requires a manual effort"*. The latter occurs also for what concerns non-functional and integration testing activities (i.e., *"we have a set of testers*

in front of a screen who monitor and check for the presence of any discrepancies about what is expected and what is instead observed while running the system.") However, due to the effort and time needed to manually verify the reliability of the software under test, while functional tests are executed for each change, integration tests are only executed when the change impacts the *"interfaces with other modules/components."*

Due to the high cost of the hardware devices involved in this particular application domain, H mainly relies on simulators that are self-developed (*"we do not rely on third party very expensive simulators"*). However, once per week, H does a testing session with a real *"train running in a real environment with real traffic [and] possibly without people."*

4.1.9 Organization I

Organization I is involved in *"develop[ing] software relying on identification technologies such as RFID [(Radio Frequency IDentification),] Bluetooth low energy or bar codes*, other than mobile app development for which there is a CI/CD pipeline used for testing and automated deployment on the play stores. It is a micro company counting less than 10 employees, and the limited availability of human resources, together with a lack of culture for setting a pipeline dealing with sensors and actuators, results in not having a CI/CD pipeline being in place for CPS development.

For what concerns the CPS development, I relies on conventional programming languages such as C# and Java, with the testing phases being almost fully-automated. Specifically, there are *"RFID-readers connected to a network"* on which it is possible to execute unit and integration testing activities automatically. For what concerns the integration testing activities, it is important to remark that there are cases requiring the manual intervention of the tester (i.e., *"For instance, when we need to test a transfer of tags between different antennas we cannot use automation"*), as well as cases where it is required to interact with the hardware devices that cannot be simulated. Of course, in this specific setting, it is not possible to guarantee the overall reproducibility of the results of the test, however, *"the reproducibility of the test in this context is not required."* Furthermore, I does not run the whole test suite at each change while manually selecting some meaningful test cases based on impact analysis: *"select what are the test cases that are impacted by the change that, consequently, need to be executed"*. Moreover, other than having unit and integration testing activities, I also executes, from time to time, performance testing. Finally, for what concerns the deployment of CPS-related software, I relies on Docker for creating images that are manually deployed onto the servers.

The development process adopted by I relies on both simulators (self-developed) and HiL. The simulators are developed based on specific needs of the organization resulting in simulators with limited functionality. One scenario in which the organization relies on simulators is the following: *"When creating a tunnel, i.e., a sequence of antennas on which it is possible to transfer tags, for verifying the content of the packets being transferred without opening them, since starting from data observed in a real environment it is possible to verify the movement of the packets across different antennas as well as verify whether the algorithm is behaving as expected"*.

Finally, the development process also contains a monitoring component aimed at controlling both the internal development platform and the devices of the customers to be notified about anomalous behaviour and errors, as soon as they are experienced.

4.1.10 Organization L

L is involved in the development of prototypes and proof of concepts for the energy domain. The development of prototypes rather than real products represents a concrete facilitation since that there may be less stringent constraints in terms of pipeline setting and evolution. The latter justifies the presence of a mature (i.e., introduced in 2016) pipeline adopted within the organization for CPS development that uses conventional programming languages, mostly Java and Python.

Going deeper on the tasks automatized within the pipeline, other than having a compilation phase, it is mainly aimed at executing unit test cases, as well as integration test cases (*"Our pipeline is mostly for unit testing (80%) but there is also some integration testing."*), followed by a deployment phase where the packaged version of the software is usually stored into an artifact repository as a docker image. Furthermore, safety requirements, such as checking that a battery is not charged more than a certain rate, are specified and checked through unit test cases that do not involve the real devices. Thanks to the need of developing prototyping solutions, the pipeline accounts for static code analysis tools and linters that are mainly used for checking maintainability issues only (i.e., *"They are not used for checking out bugs, but mostly for making sure that the code is easy to read for other colleagues and for maintainability purposes."*) Moving the attention on the triggering strategies, L does not rely on nightly builds. It only uses incremental builds so that each build execution time does not overcome the 10-minutes rule highlighted by Fowler [22].

Looking at the execution environment, L does not need to run the software on embedded devices meaning that it *"tr[ies] to find devices having interfaces to communicate with. So basically we run our software on a traditional machine and it just communicates with the hardware.* So, differently from other organizations, organization L, other than simulating the hardware when needed (i.e., *"we simulate the battery for testing the charging protocol"*), mainly replaces it with mock-ups (i.e., *"it is very easy to mock a client just to see if our software sends the right commands or does not use any register twice"*). Only when the real devices are available and it is safe to use them for testing, L uses Docker images for checking the correct behaviour over the real devices, as well.

Finally, while asking the interviewee about the frequency of changes occurring to the pipeline configuration, we realized that the configuration is pretty stable probably due to the development of prototyping solutions that do not need to be shipped to real environments.

4.2 RQ_2: *What are the challenges and barriers respondents encounter?*

This section starts by describing the challenges faced by the interviewed organization related to the CPS development process they adopt. After that, there is a description of the barriers and challenges encountered while setting and maintaining the CI/CD pipeline for CPS development. Finally, we report the results of the external survey used to verify the previously discussed pipeline-related challenges and barriers.

4.2.1 Process-related challenges

Table 7 reports the set of process-related challenges mentioned by the 10 organizations participating in the semi-structured interviews, together with the traceability among which challenge has been encountered by which organization. Specifically, the challenges have been grouped into six different categories, each one related to a specific aspect of the CPS development process — general, culture, environment, testing, deployment, and simulators. In the following, for each category, there is a brief description of the challenges belonging to it, together with some examples.

General. This category accounts for two challenges, each one mentioned by only one out of ten interviewees. One of the main benefits of adopting a CI/CD pipeline is related to the overall cycle time reduction. However, even if organization B has already invested, in the last years, effort and money in reducing their release time, it already sees space for reducing it (*"The biggest problem ... is cycle time. Three years ago, the cycle time was six weeks, while now we could do it every day. It is still not enough from a developer perspective because the feedback is not fast enough."*)

Industrial companies, as well as, open-source communities, are very often open to on-board new developers. To this extent, organization G is facing problems when trying to on-board new developers mainly due to the complexity of the railways' domain. Specifically, the interviewee stressed the fact that in the railways' domain for developing a new functionality it is important to follow specific standards that need to be known

Table 7: Set of process-related challenges

Category	ID	Challenge	Organizations
General	PRC_1	Cycle-time reduction	B
	PRC_2	On-board developers	G
Culture	PRC_3	Limited CI/CD culture	A
	PRC_4	Limited CI/CD culture for CPS development	I
Environment	PRC_5	Complexity of the environment	B, D, E, F, H
	PRC_6	Variability of the environment	G
	PRC_7	Lack of redundancy in the environment	G
	PRC_8	Complexity for integrating tools for automated traceability	E
	PRC_9	Complexity for integrating tools for requirements' management	E
Testing	PRC_{10}	Test cases manually derived	E, H
	PRC_{11}	Test cases manually executed	C, I
	PRC_{12}	Different interpretations for the same requirements	G
	PRC_{13}	Need a controlled environment for test automation	C, D
	PRC_{14}	Complexity in oracle specification for test automation	G, E, F, H, I
	PRC_{15}	Complexity for deriving integration tests	L
	PRC_{16}	Complexity for deriving safety tests	D, E, H
Deployment	PRC_{17}	Late deployment	B
	PRC_{18}	Expensive deployment	G
Simulators	PRC_{19}	Lack of trustworthiness for simulators	C
	PRC_{20}	Complexity for oracle automation with simulators	H

and properly understood by developers and testers. The latter, of course, makes more difficult the process of on-boarding new developers (i.e., *"It's quite time-consuming to on[-]board new developers."*)

Culture. This category groups two challenges dealing with the presence of a limited CI/CD culture/knowledge within the development teams, hindering the possibility to properly set and use a CI/CD pipeline for supporting the overall development process. Specifically, while organization A reports the adoption of a pipeline that only includes tasks that are easy to automate mainly due to *"lack of knowledge"*, organization I has already in place a pipeline for developing and deploying mobile apps to the app-store (i.e., *"The setting of a CI/CD pipeline in the mobile context has been very easy"*), however, it does not have a pipeline for CPS development due to *"a lack of a deeper knowledge in the CI/CD context for CPS"* in particular for what concerns the level of communication and interaction between software and hardware components.

Environment. This category contains five different challenges dealing with (i) the characteristics of the physical environment in which the developed code has to be deployed (PRC_5, PRC_6 and PRC_7), and (ii) the complexity when integrating tools that are dictated from the application domain (PRC_8 and PRC_9). As regards the former, only PRC_5, i.e., environment complexity, is mentioned by multiple organizations (five out of ten), while the remaining two come from the same organization (G). Specifically, the complexity of the environments impacts the execution environment used within the pipeline (simulators or HiL), since if companies rely on self-developed simulators, the presence of a complex environment to simulate results in simulators that are limited in their functionality or else hinders the level of trustworthiness in terms of their functional correctness. For instance, organization D mentioned: *"Walking is not so easy to simulate so we need a real walking robot for spotting bugs*, while organization H stated: *"It could be difficult, demanding and expensive to have a one-to-one relationship between simulators and real systems.* Organization G, instead, faces a problem, i.e., the high variability of the environment, strictly related to the application domain where there is a need to configure the architecture of the train based on the trains' characteristics, — *We can rarely copy-paste software that has to run on different train architectures.*". Furthermore, it also faces a challenge due to the structure of its development process that is not cloud-based and has no redundancy, implying that *"in the presence of network issues or server issues we are totally black and this is affecting everyone."*

The challenges dealing with tool integration (PRC_8 and PRC_9) come from a single organization (E) stressing more the importance of requirement engineering when developing CPSs. Specifically, the interviewee stated:

"every time you make a change impacting the traceability you must have a requirement change that drives that change in the code ... Good formal testing tools are lacking together with requirement engineering and guaranteeing traceability between requirement, code changes and tests."

Testing. This category groups seven different challenges dealing with the testing strategies adopted by the interviewed organizations. It is important to note that, as shown in Table 7, this is the only category where the challenges are encountered by more than one company (the only exceptions are PRC_{12} and PRC_{15}). Two out of 10 organizations (E and H) mention as a challenge the manual specification of test cases, while organizations C and I felt like a challenge the manual execution of testing activities (e.g., *"Another big barrier is related to the test case execution that, at the moment, we are doing manually since both the environment setting and the oracle definition require manual intervention."*) Related to the first point, organization G found it difficult to automate the test case specification mainly because the standards might be interpreted differently by different developers, and both the interpretations might be correct at the same time — *"how do you read the standard? The standard is interpreted so the same requirement can be differently interpreted by different people (a challenge for automation)."* The need for a controlled test environment influences the choice of simulators and HiL used in the pipeline, indeed the interviewee from organization C mentioned: *"Since the output of the system is sound and the test should check the sound quality it is better to have it in a controlled environment that makes use of simulation."*

A different challenge impacting the automation of the test execution is related to the complexity encountered for specifying the oracle (PRC_{14}), mentioned by 5 out of 10 organizations. Specifically, the impossibility of specifying an automated oracle limits the phases automatized within the pipeline. The latter may occur, for instance, when one needs to evaluate a signal received from a sensor, i.e., *"The main challenges for automatizing the test execution: a good way to model the test itself and have an oracle that can compare with the actual behaviour."*

The remaining two challenges deal with the difficulty encountered while one needs to specify/derive integration (PRC_{15}) and safety (PRC_{16}) test cases. As regards the former, L develops prototypes requiring the interconnection of many different sub-components making it difficult to determine the expected behaviour of the system, on the whole, i.e., *"It is quite hard to derive integration test cases due to the complex combination of all different parts."* As regards the latter, instead, the problem faced by all the three organizations when talking about the specification of test cases aimed at checking safety requirements is related to the fact that they must highlight situations *"that could never happen."* or *"that you do not expect to happen."*

Deployment. This category is made up of two challenges dealing with the process adopted within the organization for the deployment of new versions of the produced software to the customers. The latter influences the set of phases included within the pipeline, and, at the same time, may raise challenges for setting and maintaining the pipeline itself. For instance, having deployment too late in the process (PRC_{17}) may result in installation issues (PC_9 in Table 9), as experienced by organization B: *" we will not be able to run the software on the system because the installation even does not work on the system, because the update/upgrade does not work, or because the system behaviour is not being considered in the early stages of development."*

Furthermore, there are also cases where the deployment is expensive in terms of time and effort needed to complete it, impacting both the type of execution environment adopted within the pipeline, as well as the build triggering strategy used by the organization. Specifically, as experienced within organization H in the railways' domain, the deployment over the test track requires *"one day with people involved in the testing and on a train a couple of days where many people need to be involved."*

Simulators. The last category, among the process-related challenges, deals with the usage of simulators. One of the challenges in this category (PRC_{20}) is related to the complexity of automatizing the test execution due to the impossibility to properly specify an oracle (PRC_{14}). If it is complex for a human to specify the expected behaviour for some scenarios, of course, it is not possible to rely on simulators that are able to emulate the same behaviour. Similarly, organization C pointed out the presence of scenarios where it is complex to trust the outcome provided by the simulators since there might be many external factors impacting the behaviour of the system in a real environment.

Table 8: Set of pipeline-related barriers

Category	ID	Barrier	Organizations
Resources	B_1	Limited human resources	H, I
	B_2	Limited availability of software and/or hardware resources	A, B, F, G, L
Domain	B_3	Complex non-functional requirements	F
	B_4	HiL not usable, e.g., for safety or security reasons	L
	B_5	Security configuration prevents CD	B

4.2.2 Barriers for CI/CD pipeline setting and maintaining

Table 8 summarizes the five barriers encountered by the ten interviewed organizations, grouped into two different categories.

Resources. This category groups the barriers dealing with limited availability of both human (B_1) and software and/or hardware resources (B_2), both influencing the type of execution environment adopted within the pipeline. Specifically, organizations F and G mainly rely on simulated environments due to the high cost of the hardware in the aerospace/railways' domain, i.e., *"Based on the fact that in the avionics domain the cost of the hardware is very expensive, we do most of the work in simulated environments"*, or *"Resources for the hardware devices (hardware test tracks and testbeds as real trains) represent an issue for us. We have a limited number of test tracks."* Differently, organization H does most of the work relying on HiL due to a limited availability of human resources needed to develop/configure simulators (which turned out to be a very effort prone activity) — *"given the needs and the budget of our company, it's much better for more complex scenarios to rely on the hardware in the loop and only use simulations when whatever needs to be simulated is very simple."*

Domain. This category includes three different barriers, two of them influencing the phases to include in the pipeline (B_3 and B_5), while the other one (B_4) impacting the execution environment. On the one hand, organization F faced as a barrier the presence of complex non-functional requirements in the aerospace domain (i.e., real-time operating systems) that constraints the setting of the pipeline to only include easy to automate tasks. On the other hand, organization B cannot include the automated deployment, at the moment, due to security policies posed by the healthcare domain (i.e., *"We cannot deploy at the moment because a change in the security configuration of the software prevented our standard [deployment] process."*) Finally, organization L raised as a domain-related barrier the impossibility to adopt, under specific circumstances, hardware-in-the-loop in the CI/CD pipeline (e.g., for safety reasons).

4.2.3 Pipeline-related challenges

Table 9 shows the pipeline-related challenges mentioned by the 10 organizations participating in the semi-structured interviews, together with the traceability among which challenge has been encountered by whom. The challenges have been grouped into five different categories, each one related to a specific aspect of the CI/CD pipeline setting and evolution — pipeline properties, thoroughness, simulators, HiL, and flaky behaviour. In the following there is a description of each identified challenge, together with some examples aimed at better understanding and clarifying them.

Pipeline Properties. This category accounts for six different challenges, two of them dealing with the build execution time (PC_1 and PC_2), while the remaining four dealing with the overall pipeline configuration. Four out of 10 organizations deal with an excessively long build execution time influencing the type of tasks automatized within the pipeline (e.g., F mentioned: *"Slow builds hinder the inclusion of running non-functional testing in the pipeline."*) While this is also considered a relevant challenge for conventional applications [16, 65, 71], for CPSs the problem can be further exacerbated upon deploying and executing software on simulators or HiL. For instance, organization B indicates how despite the huge investment, the pipeline is not as fast as expected: "*we spent a lot of money on infrastructure to get them going [however] when developers come to work, jobs*

Table 9: Set of pipeline-related challenges

Category	ID	Challenge	Organizations
Pipeline Properties	PC_1	Long build execution time	A, B, E, F
	PC_2	Build time estimation	I
	PC_3	Static code analysis tools configuration	C, G
	PC_4	Lack to access the production code from the pipeline	A
	PC_5	CI/CD configuration highly coupled with the environment	B, E
	PC_6	Re-usability of build artifacts	B
Thoroughness	PC_7	Development environment detached from the execution environment	A
	PC_8	Detecting deployment-related errors	B, F
	PC_9	Continuous installation	B, D
	PC_{10}	Closing the loop introduces performance degradation	E
	PC_{11}	Complexity in closing the loop due to uncontrollable factors	D, I
	PC_{12}	Complexity in closing the loop due to data collection from the field	E
Simulators	PC_{13}	Limited in their functionality	B, D, G, H, I, L
	PC_{14}	Functional correctness	E, F, G, L
	PC_{15}	Deal with real-time properties	F, I
	PC_{16}	Interaction with the environment	E, F, H, I
	PC_{17}	Accessibility	A
HiL	PC_{18}	Availability	L
	PC_{19}	Automated deployment on HiL	G, H, I
	PC_{20}	Test Automation on HiL	B, D, F, G, I
	PC_{21}	Costs and scalability	B, E, G
	PC_{22}	HiL standalone	C
Flaky Behaviour	PC_{23}	Dependency installation	D
	PC_{24}	Features' interaction	B
	PC_{25}	HiL availability	L
	PC_{26}	HiL inputs	E, L
	PC_{27}	Lack of control over resources	D, F, G, I, L
	PC_{28}	Network issues	D, E, L
	PC_{29}	Timing issues	D, L

are still running." A different challenge experienced by organization I deals with the variability in terms of the amount of time required for the overall build execution mainly due to the used infrastructure, i.e., *"Since our platform works in the cloud we need to know how much time it is required to acquire and elaborate a huge amount of data points"*.

Moving the attention on the challenges dealing with the configuration of the pipeline itself, in the absence of clear coding standards or guidelines (organization G), i.e., with ad-hoc coding conventions defined by developers, the adoption of code style checking tools becomes problematic, if not unfeasible. In this scenario, approaches for coding style inference may be desirable [50]. Similar considerations apply to bug-finding tools, sometimes inapplicable to CPSs for automating code review, as experienced by organization G: "*we need expertise on the developers' side for determining whether or not a train is behaving in the expected way.*" As experienced by organization A, the impossibility to have access to production code limits the ability to properly set the pipeline, e.g., for what concerns static analysis or testing tools — *"One big challenge is that we need to guarantee the protection of the source code: How to test a component without having the production code of the component?"* On the same line, there is a challenge (PC_5) related to the complex environment infrastructures being used within the company for developing CPSs that impacts the setting of the pipeline, i.e., technology for organization B where *"the Windows situation does not help us with dockerization"*, as well as *"Microsoft Technologies. We leverage WDS (Windows Deployment Service) for automating windows installation that is not very fun without this service"*, and deployment infrastructure for organization E using virtual machines for containerization instead of Docker — *"However there is a cost to pay while relying on VMs instead of Docker."*

The last challenge is related to the impossibility to re-use previously built artifacts in the integration branch (i.e., organization B mentioned: *"It's a huge pain that we do not reuse artifacts"*), mainly due to the constraint of

"follow[ing] medical application frameworks providing a base set of rules in terms of how to build applications and how to integrate them."

Thoroughness. This category groups six different challenges related to (i) ensuring the overall accuracy and completeness of the CI/CD pipeline, i.e., PC_7, PC_8, and PC_9 scattered across four different organizations, and (ii) closing the DevOps loop by gathering data from the hardware, i.e., PC_{10}, PC_{11} and PC_{12} experienced by three out of 10 organizations. As regards the former, by looking at their names in Table 9, it is possible to state that they have an impact on the phases included within the pipeline. Specifically, concerning deployment, continuous installation (PC_9) cannot be achieved due to the late deployment strategy (PRC_{17} in Table 7) adopted by organization B. This is because changes to the environment impact the pipeline configuration, which needs to be adapted every time. Another challenge (PC_8 experienced by organizations B and F) occurs in the presence of incremental deployments, which make it difficult to detect and isolate deployment errors. Furthermore, organization F reported how this even makes it necessary to reconfigure the entire pipeline — *"you deploy blocks, if there is an error in one of the blocks detecting it and reconfigure and reset the pipeline is a problem."* Finally, A poses a challenge dealing with having a development environment detached from the execution environment that hinders the accessibility of the production code to be tested.

Focusing the attention on the need of closing the DevOps loop, from the interviews it comes up that there are three different reasons (challenges) hindering the acquisition of data from the physical environment (or hardware device). Specifically, organization E stresses the introduction of performance degradation due to invasive measurement instruments: *"The challenge is that monitoring becomes invasive with respect to the system performance."*, as well as the presence of guarantees about data collection (PC_{12}) during CPS development: *"There are architectural ways to deal with that so that if some sensor does not update on time, you still can make a relatively informed decision. But even then, you have to make sure that the drift is not over a certain size because then you cannot make reasonable decisions anymore."* Organizations D and I, instead, highlighted how the presence of uncontrollable factors in a real execution environment hinder them to close the DevOps cycle, e.g., D reported: *"Differently from other software applications, there is data that we cannot control such as the presence of something on the floor that the robot is not able to perceive so it will fail. You have to analyze the video data and this is very hard."*

Simulators. This category groups five challenges related to issues and limitations of simulators. Specifically, the complex environment and the need to develop them in-house may lead to limited functionality (PC_{13}), limited capability to simulate a complex interacting environment (PC_{16}), and even correctness problems (PC_{14}). Specifically, a lack of knowledge about the device/system to simulate can lead to wrong assumptions affecting the simulator's correctness, as experienced within organization L: (*"This happens more at the beginning of a project when you are not too familiar with the device and you make assumptions on how it works."*) Furthermore, barriers in terms of limited human resources being available within the company that self-develops simulators may lead to simulators that are limited in their functionality, of course impacting the execution environment adopted within the pipeline (e.g., G stated: *"we prefer to spend time in testing on real hardware instead of spending time in developing complex simulators."*) Also in presence of a complex environment to simulate, companies prefer to do most of the work relying on HiL instead of developing complex simulators (e.g., D reported: *"Walking is not so easy to simulate, so we need a real walking robot for spotting bugs."*) Finally, the high level of interaction between different components in the real environment that need to be simulated leads companies to directly test feature interaction by using real devices (i.e., HiL), instead of simulate them in a virtual environment. As an example, within the organization F *"for the CAN data, what do you want to wish to happen here? If you are driving around something you need to know how fast the wheels are turning, as well as, what the engine revolutions are together with other sensitive data you might pick up over the canvas. There are a lot of details that are very application dependent."*

Limited/no capability to simulate real-time properties is also a challenge originating from a too complex environment (as experienced within organization F), but also generates further challenges related to flakiness, as experienced within organization D. Finally, simulators owned by third-parties are problematic to access and use from the outside, as experienced by organization A that has been constrained to use a specific type of simulator dictated from the aerospace domain in which it operates.

HiL. This category groups five challenges related to issues and limitations of adopting hardware-in-the-loop within the pipeline. As shown in Table 9, three out of five challenges in this category are experienced by multiple companies, while two are organization-dependent. Specifically, testing on HiL (PC_{20}) is considered very demanding to put in place (e.g., B reported: *"If you translate test strategies to the hardware it is very demanding."*) and this is mostly a consequence of limited human resources being available. However, there are cases where testing on HiL is constrained by the high cost and lack of scalability of the hardware devices/systems (e.g., *"This costs and does not scale"* for B, or *" it is very costly to test on trains"* for G).

Furthermore, as experienced within organizations G, H and I, also deployment on HiL may be challenging (PC_{19}): e.g., *"Remote installation cannot be used with real systems"* in organization H, or *"The other challenge is related to having a fully automated deployment over the customers' server in which it is possible to have full control on what is going on and try to identify, as soon as possible, failures/errors occurring during the deployment."* in organization I.

Finally, L has problems with checking the availability of the hardware (i.e., *"One of the biggest problems, when any particular hardware is involved, is that the hardware may either not be available, or it may be switched off"*) before using it for testing purposes within the pipeline (PC_{18}). The latter might also generates further challenges related to flakiness because without a proper check of the availability of the hardware the build outcome might fail intermittently since the pipeline was not able to properly communicate with the device (i.e., *"We experienced flakiness in terms of non-deterministic behaviour mainly due to hardware not being available."* (PC_{25})).

Flaky behaviour. This category accounts for seven different root causes that may lead to non-determinism in the build execution used for CPS development. Flakiness related to non-determinism in test execution [74] has been largely studied [18, 39, 41, 48, 75] and approaches to cope with them proposed [40, 54, 73]. However, the root causes behind flaky behavior in CPSs may be different from conventional software. Specifically, the pipeline can suffer from flakiness because simulators could not cope with timing issues (PC_{29}, e.g., *"the last problem is related to multi-threaded programming"* within organization L), because of the complex interacting environment (PC_{24}, i.e., *"Flakiness: it's a huge topic, indeed it is not only the network but the complexity of our subsystems whose features interact across many indirections that may lead to non-deterministic behaviour"*), or of external resources changing their behavior while the pipeline owner has no control over them (e.g., *"The most important root cause we experienced is related to the load on the server-side"* within organization D, or *"This is complex because most of the cases are related to external tools that may behave differently (e.g., virtual machines behaving differently)"* within organization G). Finally, when using remote HiL, flakiness can be determined by the presence of noise in the measured values as experienced by organizations E and L (i.e., *"Other times the charge level that you read out would go a little bit higher or there is noise in the measurements."* for L, and *"you need to understand what your sensors are sensing and what the acceptable range of inputs are"* for E).

4.2.4 Barriers and Challenges validation through the external survey

Table 10 summirizes the results of the external survey involving eight different practitioners (referred to with P_{ID}) belonging to several CPS domains. For each challenge/barrier, we report the name, as well as the participants who have encountered it at least once within their team/organization (third column), the ones who have not faced it even if they might occur considering their CPS development process (fourth column), and the ones reporting that, based on their development process, the challenge/barrier will not be possible to occur.

Similarly to what we already found while describing the challenges/barriers in the context of the ten organizations participating in the semi-structured interviews, by looking at Table 10, we can conclude that for CPS development it is not possible to define a "good" CI/CD configuration that may apply to different domains. In other words, CPS development usually deals with high complex scenarios varying based on the application domain, meaning that it is not possible to rely on an "ad-hoc" configuration. However, the table highlights that each challenge has been encountered by at least three out of eight participants, with three challenges faced by

Table 10: Evaluation of barriers and challenges through the external survey

Category	Challenge/Barrier	P_{ID} YES	P_{ID} NO	P_{ID} Not applicable
Resources	Limited availability of software and/or hardware resources	P_1,P_2,P_6,P_8	P_3,P_4,P_5,P_7	—
Domain	Complex non-functional requirements	P_1,P_2,P_5, P_6,P_7	P_3,P_4,P_8	—
	HiL not usable, e.g., for safety or security reasons	P_1,P_2,P_5, P_6,P_8	P_3,P_4,P_7	—
	Security configuration prevents CD	P_2,P_3, P_6,P_7	P_4,P_5,P_8	P_1
Pipeline Properties	Long build execution time	P_1,P_2,P_4, P_5, P_6,P_7, P_8	P_3	—
	Build time estimation	P_1, P_5, P_6,P_7, P_8	P_3, P_4	P_2
	Static code analysis tools configuration	P_6,P_7, P_8	P_1, P_3,P_4,P_5	P_2
	Lack to access the production code from the pipeline	P_2,P_6,P_7, P_8	P_3,P_4,P_5	P_1
Thoroughness	Development environment detached from the execution environment	P_1,P_2,P_3,P_4, P_5, P_6	P_7, P_8	—
	Continuous installation	P_1,P_2,P_4, P_5, P_6	P_7, P_8	P_3
	Closing the loop introduces performance degradation	P_1,P_2,P_3, P_5, P_6	P_7, P_8	P_4
	Complexity in closing the loop due to uncontrollable factors	P_1,P_2,P_3,P_4, P_5, P_6	P_7, P_8	—
	Complexity in closing the loop due to data collection from the field	P_2,P_4, P_5, P_6, P_7	P_1, P_3, P_8	—
Simulators	Limited in their functionality	P_1, P_5, P_6	P_2, P_3, P_4	P_7, P_8
	Functional correctness	P_1, P_5, P_6	P_2, P_3	P_4, P_7, P_8
	Deal with real-time properties	P_1, P_2, P_3, P_4, P_6, P_8	P_5	P_7
	Interaction with the environment	P_1, P_2, P_3, P_4, P_5, P_6	—	P_7, P_8
	Accessibility	P_2, P_5, P_6	P_1	P_3, P_4, P_7, P_8
HiL	Availability	P_1, P_2, P_5, P_6	P_4, P_7, P_8	P_3
	Costs and scalability	P_2, P_3, P_4, P_5, P_6,P_8	P_1, P_7	—
	HiL standalone	P_2, P_4, P_5, P_6	P_3, P_7, P_8	P_1
Flaky Behaviour	Dependency installation	P_2, P_4, P_6	P_1, P_3, P_5, P_7, P_8	—
	Features' interaction	P_2, P_3, P_4, P_5, P_6, P_7, P_8	P_1	—
	HiL availability	P_1, P_2, P_3, P_5, P_6, P_8	P_4, P_7	—
	HiL inputs	P_1, P_2, P_3, P_5, P_6, P_7, P_8	P_4	—
	Lack of control over resources	P_2, P_3, P_6, P_7, P_8	P_1, P_4, P_5	—
	Network issues	P_3, P_4, P_5	P_1, P_2, P_6, P_7, P_8	—
	Timing issues	P_1, P_2, P_3, P_4, P_5, P_8	P_6, P_7	—

almost all of them (7 out of 8). Kind of expected, the most frequently occurring challenges deal with long build execution time, and flakiness in the build process due to (i) the presence of noise in the measurements coming from the hardware devices, and (ii) the high level of interaction between the different hardware devices in the real environment. Furthermore, by looking at data reported in Table 10, it is possible to state that for the challenges dealing with the adoption of simulators in the CI/CD pipeline for CPS development, we have the majority of participants stating that they cannot apply due to their development process. The latter means that, at least for what concerns the practitioners involved in our external survey, they almost apply HiL instead of software-in-the-loop.

Finally, we have looked at the comments left by the participants in the free comment field to check the presence of other challenges/barriers they encountered that are not being highlighted by the interviewed organizations. Six out of eight practitioners provided further details, some of them belonging to the development process adopted within the organization that are not specific to the setting and evolution of a CI/CD pipeline for CPS development. For instance, P_6 stresses as a problem the presence of worldwide distributed teams *"which are difficult to talk with to address joint issues or integration concerns"*. Furthermore, some of the comments are traceable to the process-related challenges we have identified in the interviewed organizations (see Table 7). For instance, P_2 highlights the problem of having requirements that are not clear from a technical perspective making it complex to properly specify the oracle for test automation (i.e., PRC_{14}), as well as the presence of strict requirements for the cyber-security which makes it difficult to derive the safety test (PRC_{16}), i.e., *"there is a really broad area of possible test/audit checks but it is very hard to estimate"*. The latter has also been confirmed by P_7: *"Security, notably supply chain security is a major concern."* From a different perspective, P_4 stressed as a problem the presence of a complex environment they have to deal with resulting in PRC_5 — *"Setting a full ecosystem together (raspberry pi, remote controller and flight control unit in our case)"*, also reported by P_1: *"we can hardly recreate the behavior of close loop motion controllers and sensors to perceive the environment"*, who alleviates the challenges by using extensive field tests, while stressing the need to include in the DevOps cycle the execution of field tests. Only P_1 and P_5 pose challenges impacting the pipeline setting and evolution. Specifically, P_5 points out the importance of data quality within the healthcare domain, i.e., *"data quality is a serious issue in our context, we cannot simulate unless specific data properties are ensured"*, impacting the triggering strategy, the build phases automated, and the execution environment adopted. Finally, P_1, similarly to organization D, stresses as root cause of discrepancies between test outcome

Table 11: Relations between challenges/barriers and mitigation strategies as seen from the semi-structured interviews and the internal survey

Challenge/Barriers$_{ID}$	**Org. (int.)**	**Org. (survey)**	**Mitigation**	**Org. (int.)**	**Org. (survey)**
B_2:Limited hw/sw resources	A, B, F, G, L	A, B, D, E, F, G, I	Test Case Prioritization	A	A, B, D(n), E, F, G, I(n)
			Use incremental builds	B	A, B, D, G, I
B_4: Domain hinders HiL	L	A, B, I	Use simulation/mocking	L	A, B, I
PC_1: Long build execution	A, B, E, F	B, D, E, F, G	Test Case Prioritization	A	B, D(n), G
			Parallelize the build execution	B	B, D, E, F, G
			Use nightly builds	A	B, D, E, F, G
			Use incremental builds	B, E, L	B D(n), E, F(n), G
PC_9: Continuous installation	B, D	B, D, E, G	Use containerization	D	D, E, G(n)
PC_{13}: Sim. with limited functionality	B, D, G, H, I, L	A, E, G, I	Rely on both sim. and HiL	D	E, I(n)
PC_{16}: Sim. highly coupled with the env.	E, F, H, I	B, D, G, I	Rely on both sim. and HiL	I	B, D
PC_{17}: Sim. accessibility	A	A, E, G, I	Use "timeout"	A	E, G(n)
PC_{21}: Costs and scalability for HiL	B, E, G	A, B, D, E, G, I	Rely on both sim. and HiL	B, G	A(n), B, D, E(n), I
			Green-build rule before HiL	G	B, D, G(n)
PC_{27}: Lack of resources' control	D, F, G, I, L	B, F, G, I	Fix the code	D	F, G, I
			Fix the pipeline configuration	G	F, I
PC_{28}: Network issues	D, E, L	A, B, D, F, G, I	Use "retry"	D, E, L	B, D, G, F(n)

in simulated and real environment the fact that *"to speed up execution of tests we simulate faster than in reality where possible"* resulting in *"subtle timing difference in the test results."*

4.3 RQ_3: *What are the mitigation strategies adopted for overcoming challenges and barriers?*

Table 11 summarizes the set of challenges and barriers for which there was an explicit mitigation strategy reported by at least one of the interviewed organizations that have been evaluated by using the internal survey (see Section 3.4). Specifically, Table 11 reports for each challenge/barrier, the organizations highlighting it during the interviews (second column), the organizations who declare to have dealt with it at least once (third column), the mitigation strategies used to overcome it (one per row), as identified during the coding of the interviews' transcripts, the set of organizations reporting their usage from the interviews (fifth column), and finally, the companies, that from the survey identify the strategy useful to overcome the related challenge/barrier (last column). As regards the latter, the presence of **(n)** near the name of the organization indicates that the respondent considers the mitigation strategy useful even if, she has never used it within the organization. Finally, when there is an organization colored in purple in the second (or fifth) column in Table 11, it means that even if the challenge (mitigation) has been highlighted during the semi-structured interviews, the survey respondent belonging to the same organization reported that she has never encountered that challenge in her team (does not think that the mitigation can be useful to overcome the related challenge). Before going deeper on the relations coming from the interviews together with their internal evaluation, it is important to remark that the survey respondents may differ from the people participating in the semi-structured interviews. Furthermore, organizations C, H and L did not validate the relations by filling in the survey.

Five out of ten organizations (A, B, F, G, L) during the interviews reported as a barrier the limited availability of software and/or hardware resources. The latter is confirmed by the survey, even if also organizations D, E, and I stress this as a concrete barrier. Two are the mitigation strategies that came from the analysis of the interviews' transcripts: (i) prioritize and select the test cases to be included within the pipeline (i.e., *"Some strategies rely on genetic algorithms to optimize the resources available for the testing execution environment."* from organization A), and (ii) adopt incremental builds mainly relying on impact analysis, as reported by organization B — *"for what concerns rolling builds we try to limit the amount of testing being executed in them to be as fast as possible."* The survey confirms the previous findings, however for the former also organizations B, E, F, and G report to rely on them, while organizations D and I consider it useful while having never relied on it. As regards the latter, instead, also organizations A, D, G and I mention its adoption. Furthermore, organization B also reports as an alternative solution *"architectural changes with improved testing concepts"*, while organization E points out the possibility to rely on *"platforms virtualization"*.

Organization L mentioned as a barrier the impossibility to rely on HiL due to constraints coming from the application domain (B_4), and adopts simulation/mocking for the hardware devices to overcome it. Organizations

A, B and I encountered it as a barrier and used the same strategy to deal with it. Furthermore, organization B mentions the presence of an alternative solution to the barrier that *"digital twin hardware that avoids the safety issues (no moving parts, no radiation) but simulates the hardware to some much better"*.

Four out of ten organizations (i.e., A, B, E, F) mentioned a wide set of actions to deal with long build execution time, (PC_1). However, A did not confirm this in the survey, while D and G declared to have encountered it. One possibility is to prioritize and select only a subset of test cases in the test suite to be executed (reported by organization A, used also by B and G, and considered a useful action by D). A different approach, highlighted by organization B, deals with the introduction of parallelization within the overall build process, i.e., *"We have 20 test machines in parallel for managing the overall test size especially for nightly builds."*. The latter is also used by organizations D, E, F, and G. A different possibility is to run the whole build process only within nightly builds, even if this may be controversial since it defeats the CI/CD purpose [15]. However, this is considered acceptable for organization A, as its pipeline is limited in scope, i.e., used only for V&V purposes. Even if the respondent belonging to organization A does not consider the build time execution as a challenge, organizations B, E, F, and G rely on nightly builds to execute time-intensive tasks, while adopting incremental builds during the working hours (B, E, L). The latter is also used by organization G, while D and F consider the mitigation useful even if they have never adopted it.

As regards the problems hindering the continuous installation as a consequence of the variability of the environment in terms of dependencies, organizations B, D, E, and F have encountered it with D pointing out that using containerization it is possible to facilitate the switching between software versions to deploy. The latter is also used by organization E, while G considers it as a useful solution.

As regards the challenges coming from the inclusion of simulators within the CI/CD pipeline (i.e., PC_{13} and PC_{16}), it is a common habit to adopt a pipeline that uses both simulators and HiL in different build stages. A clear example of this is organization G where there is a build process made up of three different build stages, each one adopting a specific execution environment (see Section 4.1). The results of the survey confirm the previous findings, even if there are some discrepancies since it is possible that the survey respondents work within a different team with respect to the participant in the semi-structured interviews.

During the interviews, only organization A declares to adopt simulators from third-party introducing as a challenge their accessibility (PC_{17}) addressed by using "timeout" within the pipeline. Quite surprisingly, other organizations confirmed the adoption of external simulators (i.e., E, G, I). However, while E uses the previously found strategy, G considers it useful to deal with the challenge. The respondent belonging to organization A, instead, indicated that within her team, instead of using "timeout", they contact the simulators' provider letting them know the problem and address it (i.e., *"request some customization at the customer side of their simulators. Sometimes it is accepted, most of the time not."*)

Even if during the interviews only organizations B, E and G reported as a challenge the high costs for the hardware systems, as well as, their poor scalability, from the survey the challenges has been confirmed by other three companies (i.e., A, D, I). There are two possible strategies to deal with this: (i) rely on a mixed pipeline (from organization B and G) where continuous builds are run on simulators and some periodic builds on HiL (confuted by G, used also by D and I, and considered useful by A and E), and (ii) adopt the green build rule when transitioning between simulators and HiL [17], as highlighted by organization G: *"Only when the tests in the virtual train are green can we move to the next step."* The latter is also used by organizations B and D, while not emerging during the interviews' coding. Finally, the survey respondent belonging to organization E highlights as an alternative solution *"working with virtual devices instead of real hardware devices"*.

About flakiness mitigation, when the problem is related to varying-behavior components (PC_{27}), the solutions adopted are (i) to change and fix the pipeline configuration, i.e., G stated: *"The misbehavior is reported back to the integration team responsible for the Jenkins configuration with the goal of finding a solution."*), as well as (ii) fix the root cause of the flaky behaviour within the code *"to not experience it anymore in the system"*. The results of the internal survey confirm the previous relation, even if both the organizations providing the mitigation strategies do not rely on them (organization G), or else has never encountered it (organization D). When the root cause of the flaky behaviour is in the networking, the companies leverage the "usual" retries:

e.g., *"of course we have some retry for network issues"* for D, or *"For what concerns flaky connections, you have to be concerned about missed messages and retries"* for E. Looking at the survey results, the respondent belonging to organization E has never encountered flakiness due to network issues, however, also organizations A, B, F, G and I faced it. Moving onto the mitigation strategy, instead, only organizations B and G use retries, while F only considers it a valuable solution. Furthermore, the respondent belonging to organization B mentioned as an alternative solution the *"introduction of quarantine builds together with an appropriate process of how to deal with these tests"*.

4.4 RQ_4: *What are the challenges and bad practices for CPS development in open-source projects? How are they mitigated?*

From the 364 pull requests manually analyzed, we found 140 (38.5%) pull requests discussing at least one bad practice or challenge related to the CI/CD pipeline setting and maintaining. Specifically, since the same pull request may belong to several bad practices and/or challenges, in the sampled pull requests there are 158 problems being discussed of which 62 are classified as bad practices and 96 are related to challenges and barriers. Furthermore, during the manual validation, the coders have also tried to check whether the bad practice/challenge is specific to CPS development, or else may occur also for traditional software development. By looking at the results, all the 62 bad practices being identified are not CPS-specific, while for what concerns the 96 challenges, 80 are CPS-specific, while 16 may also occur when dealing with the setting of a CI/CD pipeline for the development of conventional software. In the following, it is possible to find a high-level description of the bad practices identified together with some qualitative examples aimed at better understanding their meanings. After that, there is a detailed description of the challenges being identified in the 140 PRs, focusing more on the ones that are specific to CPS development, and have not been discussed as a result of the semi-structured interviews and the internal survey.

4.4.1 Bad Practices

Table 12 reports the set of bad practices identified among the 140 PRs discussing at least one problem dealing with the pipeline setting and its maintenance. Specifically, for each bad practice, the table reports the category to which it belongs, as well as its name and the total number of PRs in which the bad practice has been discussed. Furthermore, the table highlights the bad practices that have been added, while not being reported in the previous catalog [71] — used as starting point for the manual validation procedure.

By looking at the data summarized in Table 12, it is possible to state that 36 different bad practices are discussed belonging to 14 different categories each one specific to a particular aspect of the CI/CD pipeline setting. 13 out of 36 bad practices identified are not reported in the previous catalog. In the following, there is a description of the 13 bad practices discovered in the context of this study, while for the other ones you can refer to the original catalog [71].

In the **Infrastructure Choices** category, two new bad practices are dealing with the poor software choices related to the definition of the overall pipeline that may introduce (i) unnecessary build failures, (ii) miss to highlight relevant problems, and (iii) degradation of the overall build process maintainability. As regards the choice of the CI framework to use, PR #15463[4] in ARDUPILOT describes the need to change the CI/CD frameworks (i.e., move from Travis-Ci to Github Actions) by pointing out the main advantage of the proposed infrastructure such as the possibility to have more than 20 tasks/jobs running in parallel, together with *"no more timing issue on autotest"*. As regards, instead, the type of infrastructure used for caching, PR #3228[5] from ARDUPILOT describes the advantages of adopting the container-based infrastructure rather than virtual machines (i.e., *"The biggest advantage of using the container-based one is the ability to cache a) things we*

[4] https://github.com/ArduPilot/ardupilot/pull/15463
[5] https://github.com/ArduPilot/ardupilot/pull/3228

Table 12: List of bad practices mentioned in the 364 PRs together with their frequency.

Category	Bad Practice	Is new?	# of PRs
Infrastructure Choices	Wrong Choice of the CI/CD framework being used	☑	1
	Wrong Choice of the caching infrastructure (i.e., VMs versus Containerization)	☑	1
Build Initialization	Inappropriate build environment clean-up strategy		4
Build Process Definition	Monolithic builds are used in the pipeline		3
	Wide and in-cohesive build jobs are used		2
	Unneeded environments are included	☑	1
Build Execution	Pipeline steps/stages are skipped arbitrarily		3
	Tasks are not properly distributed among different build stages		2
	Parallelize build jobs while they should not run concurrently	☑	2
	Independent build jobs are not executed in parallel		1
	Only the last commit is built, aborting obsolete and queued builds		1
	Build steps are not properly ordered		1
Build Triggering	Inactive projects are being polled		1
Build Outcome	A build is succeeded when a task is failed or an error is thrown		1
	A build is not failed as soon as a failure/error is encountered	☑	1
Build Dependencies Management	Including unneeded dependencies		5
	Inappropriate choice about what to cache	☑	4
	Cache size does not fit the environment being available	☑	3
	Dependency management is not used		1
Build Output	Some tasks are executed without clearly reporting their results in the build output		4
	Build reports contain verbose, irrelevant information		2
	Not enough history available for debugging purposes	☑	1
Build Duration	Build time for the "commit stage" overcomes the 10-minutes rule		2
	Unneeded tasks are scheduled in the build process		1
Security	Sudo is used for accessing Docker	☑	2
	Authentication data is hardcoded (in clear) under VCS		1
Build Maintainability	Configuration customized over a specific environment	☑	2
	Build configurations are cloned in different environments		1
	Wrong smoke test ordering	☑	1
	Build artifacts scattered across different locations	☑	1
	Duplicate CI/CD configurations on different CI framework	☑	1
Quality Assurance	Test suite contains flaky tests		1
Delivery Process	Missing check for deliverable		1
Culture	Changes are pulled before fixing a previous build failure		2
	Issue notifications are ignored		1

need to download and b) intermediary build steps by using ccache"). This problem will for sure have an impact on the overall build execution time: *"The fact is that with this new approach I'm seeing a build time 4 times faster."*

In the **Build Process Definition** category, other than having PRs dealing with (i) lack in the definition of cohesive build jobs, defeating the "Fast Feedback" practice, and (ii) build the entire project instead of building each sub-project separately, there is one pull request (#11155[6] in ARDUPILOT) where, to avoiding *"wasting time"* during the build process, it is important to remove some environments that are not needed anymore in the CI/CD process.

As regards the **Build Execution** category, we confirm some of the previously known bad practices, such as do not parallelize builds' execution leading to an increase of the overall build execution time, or adopt inappropriate ordering of build steps, leading to a pipeline that is not able to find bugs following the "Fast Feedback" principle. Furthermore, in the sampled PRs object of this study, there is a new bad practice being identified where tasks/steps that cannot be executed concurrently, are parallelized in the build process. For instance, in PR #1094[7] in CARTOGRAPHER, the developer is pointing out the need to not run concurrently tasks that are dependent from each other (i.e., *"tests in cartographer/cloud that cannot run concurrently with each other"*).

The **Build Outcome** category includes bad practices dealing with the policies used for assigning the status of a completed build. Differently from the previous catalog, there is a pull request (#11529[8] in ARDUPILOT)

[6] https://github.com/ArduPilot/ardupilot/pull/11155
[7] https://github.com/cartographer-project/cartographer/pull/1094
[8] https://github.com/ArduPilot/ardupilot/pull/11529

where the developers are struggling with the delayed feedback received as a consequence of not failing the build process, as soon as a problem is encountered—*"stop us waiting 10 hours for a failure"*.

Build Dependencies Management groups the bad practices related to the strategy adopted for dealing with dependencies, since relying on a poor strategy may result in a subsequent number of useless or unnecessary build failures. Other than confirming the previously defined bad practices, there are two newly introduced bad practices: (i) inappropriate choice about what to cache, and (ii) inappropriate setting of the overall cache size. As regards the former, in PR #298[9] from CARTOGRAPHER, the developer is pointing out the presence of failing builds as a consequence of the loading of empty docker images *"an empty Travis cache sometimes gets saved"*. As regards the latter, instead, PR #4208[10] from ARDUPILOT discusses how to properly choose the size of the cache to use for compilation purposes considering different trade-offs, i.e., *"I thought the whole point of how ccache makes things faster is retaining some info between builds when we change targets. Or, do we clean between each target and thus don't need ccache anyway?"*.

Quite interesting, in the **Build Output** category there is one pull request belonging to the newly introduced bad practice dealing with the inappropriate choice about how many build logs are stored (i.e., how much of the build history is retained) so that it is possible to rely on them for debugging purposes, e.g., identify the presence of flaky behaviour within the build process (PR #11398[11] in PX4-AUTOPILOT).

Since the CI/CD pipeline usually runs on a dedicated server, it is necessary to deal with **security** issues such as (i) the hard-coding of authentication data under version control, and (ii) relying on Sudo when accessing Docker, i.e., new with respect to the original catalog. As an example of the latter, PR #3847[12] in ARDUPILOT reports *"We should try to remove the necessity of sudo commands so we can use cache."*

Finally, the last category for which the sampled pull requests discuss four newly introduced bad practices is **Build Maintainability** that may result in an excessive number of build failures, as well as a limited build maintainability and portability. There are two pull requests highlighting that by relying on a pipeline configuration that is highly customized over a specific environment, it is not easy to make it works in a completely different environment, e.g., rework for making the system runs on a different operating system. Others pull requests discuss (i) the adoption of a wrong ordering while executing the smoke tests, resulting in a lack of consistency between private builds and builds on CI server, and (ii) having build artifacts scattered across different locations (e.g., servers). As regards the latter, PR #242 in FPRIME reports: *"Currently, `fprime-util install` installs files into a variety of different locations, which makes it difficult to archive all the build artifacts.* The last introduced bad practice, instead, might occur when the project relies on several CI/CD frameworks, each one with a specific build process definition. However, it is possible that different CI/CD frameworks are used for running the same build process, hindering their overall maintainability, e.g., each change in the build process affects all the adopted frameworks. Consider, PR #10114[13] in ARDUPILOT, where one contributor clearly reports: *"it just looks like they are overlapping efforts."*

4.4.2 Challenges and Barriers

Table 13 reports for each challenge/barrier discussed among the 364 manually validated pull requests belonging to 10 different open-source CPS projects, whether or not it is obtained as a result of the semi-structured interviews (third column), as well as, the number of pull requests in which it has been discussed (last column). Since that a challenge/barrier may or may not be specific to CPS development, the last column reports in parenthesis how many pull requests describe the related challenge/barrier in a context that may also occur when setting or maintaining a pipeline during the development of traditional software.

[9] `https://github.com/cartographer-project/cartographer/pull/298`
[10] `https://github.com/ArduPilot/ardupilot/pull/4208`
[11] `https://github.com/PX4/PX4-Autopilot/pull/11398`
[12] `https://github.com/ArduPilot/ardupilot/pull/3847`
[13] `https://github.com/ArduPilot/ardupilot/pull/10114`

Table 13: List of barriers/challenges mentioned in the 364 PRs together with their frequency.

Category	Bad Practice	Is new?	# of PRs
Pipeline Properties	CI/CD configuration highly coupled with the environment		6 (4)
	Long build execution time		1 (8)
	Licensing requirements for external APIs	☑	(1)
Thoroughness	Automating HiL testing with several hardware devices	☑	1
	Lack of HiL specific smoke test	☑	1
Flaky Behaviour	Lack of control over resources		3 (1)
	Network issues		1 (1)
	HiL availability		1
Simulators	Limited in their functionality		21
	Functional correctness		21
	Configuration	☑	7
	Deal with real-time properties		5
	Interaction with the environment		3
	Choose among different simulators	☑	2
	GUI-based	☑	(1)
HiL	Memory Management	☑	3
	Costs and scalability		1
	Parallelization and Resources Blockage	☑	1
Resources	Limited availability of software and/or hardware resources		1
Domain	Complex non-functional requirements		1

As can be noticed in Table 13, the sampled pull requests discuss 20 challenges, of which 8 were not discussed by the ten interviewed organizations. As already done with the discussion of the bad practices, in the following there is a detailed description of the 8 newly introduced challenges together with some qualitative examples aimed at better understanding them.

In the **Pipeline Properties** category, there is one PR (# 17717[14] from "PX4-AUTOPILOT") where developers are struggling with licensing issues when incorporating an external driver which requires a library that does not satisfy the licensing strategy adopted by the community. The adherence to specific licensing agreements may limit the third-party simulators that can be used within the current setting of the project.

In the **Thoroughness** category, two newly introduced challenges deal with test automation when using HiL. On the one hand, PR #13722 from PX4-AUTOPILOT highlights the challenge about the definition of a proper strategy to use when testing on HiL in presence of several different hardware devices, i.e., *"Hardware drivers cannot be added to automatic testing because the RPi platform is built up by a lot of different sensor solutions in most case"*. There are cases, especially for open-source projects, where the provided functionality, as well as the overall architecture of the project, is limited and will be updated each time a new request reaches the repository. The latter may result in architectures that are highly customized on specific environments (e.g., specific hardware devices), and are difficult to adapt to new requested ones. On the other hand, in the same project, PR #12282[15] points out a problem when using CI/CD relying on HiL due to a lack of smoke test aimed at inspecting hardware devices—*"run various commands to inspect system"*.

As regards the challenges dealing with the adoption of **Simulators** within the CI/CD pipeline, other than confirming most of the challenges previously identified by analyzing the interviews' transcripts, there are three newly introduced challenges. First of all, there are seven PRs where developers point out problems in terms of the configuration setting of the simulators in the pipeline. For instance, in PR #15206[16] from PX4-AUTOPILOT, developers discuss the adoption of a wrong strategy used to clean-up the server used for running the simulators resulting in silent failures: *"When running gazebo SITL simulation multiple times, gzserver*

[14] https://github.com/PX4/PX4-Autopilot/pull/17717
[15] https://github.com/PX4/PX4-Autopilot/pull/12282
[16] https://github.com/PX4/PX4-Autopilot/pull/15206

will sometimes silently fail. This is due to the fact that gzserver didn't exit cleanly from the previous session." Another challenge being discussed within two different pull requests deals with the choice of the simulators to use when multiple are available. As an example, in PR #14539[17] from PX4-AUTOPILOT, developers discuss the features provided by two different simulators to properly choose whether or not to change the one being used within the pipeline— *"The FlightGear has better support for modeling of rotorcraft than the current PX4's mainstream simulator Gazebo."* Finally, PR #12781[18] in the same project, mentions the need to have the possibility to run a simulator without GUI so that it is possible to reduce the manual intervention required during the build process.

The last two newly introduced challenges belong to the **HiL** category and deal with (i) a wrong usage of the resources available on the hardware devices, i.e., the memory, and (ii) resources blockages occurring when parallelizing over the hardware devices. As regards the former, PR #1645[19] from PAPARAZZI, struggles with having a packaged version of the software that has to run over the hardware that is too big and does not fit the memory available on the device. Finally, for what concerns the latter, PR #16396[20] in PX4-AUTOPILOT, the developer mentions the need to adopt a *"single thread to be safe"* to avoid *"fast Ubuntu machines to effectively lock up."*

4.4.3 Mitigation of bad practices and challenges in open-source projects

Figure 3 shows how developers from open-source projects restructure the CI/CD pipeline to remove the bad practices, while Figure 4 shows the mitigation strategies used for dealing with barriers and challenges when setting or maintaining a pipeline for CPS development.

As described in Section 3.5, while manually validating the 364 sampled pull requests from ten open-source CPS projects, the coders, in the presence of a bad practice or challenge/barrier described in the pull request body, also looked at whether the discussion also describes the strategy adopted to overcome it. In doing this, the coders could choose among the 34 restructuring actions mainly applied in the context of evolving a CI/CD pipeline, as provided by Zampetti et al. [70], as well as the 12 mitigation strategies identified during the coding of interviews' transcripts. Among the 140 PRs discussing at least one bad practice or challenge, 56 (40%) also describe the action applied to overcome the related problem.

By looking at Figure 3, it is possible to state that, very often, open-source developers use a mitigation strategy that is specific to each bad practice. Indeed, for addressing 22 different bad practices, developers adopt 15 different restructuring actions, of which three were not already identified in the previous study related to pipeline evolution [70], i.e., repair smoke test, centralize code compilation, and increase the overall cache size. Going deeper on the actions adopted to remove bad practices impacting the pipeline setting and maintainability, eight out of 15 actions are not bad-practice specific, meaning that it is possible to adopt the same restructuring action to deal with different bad practices. As an example, developers remove environments, tasks, and dependencies from the pipeline to reduce the overall build execution time (1), to remove unneeded dependencies (2), and unneeded environments (1). Furthermore, as regards the actions used to reduce the overall build time, developers change the configuration of the cache (2), increase the overall cache size (1), parallelize the overall bud execution process (2), as well as, remove unneeded environments from the build process definition, i.e., cleanup build matrix (2).

Moving the attention on the way open-source developers overcome challenges and barriers when setting or maintaining a pipeline for CPS development, from Figure 4, it is possible to state that, as for the bad practices, the mitigation seems quite specific to the challenges they are aimed to overcome. Furthermore, out of eight mitigation strategies, four have not been reported by the interviewed organizations, i.e., adding better

[17] https://github.com/PX4/PX4-Autopilot/pull/14539
[18] https://github.com/PX4/PX4-Autopilot/pull/12781
[19] https://github.com/paparazzi/paparazzi/pull/1645
[20] https://github.com/PX4/PX4-Autopilot/pull/16396

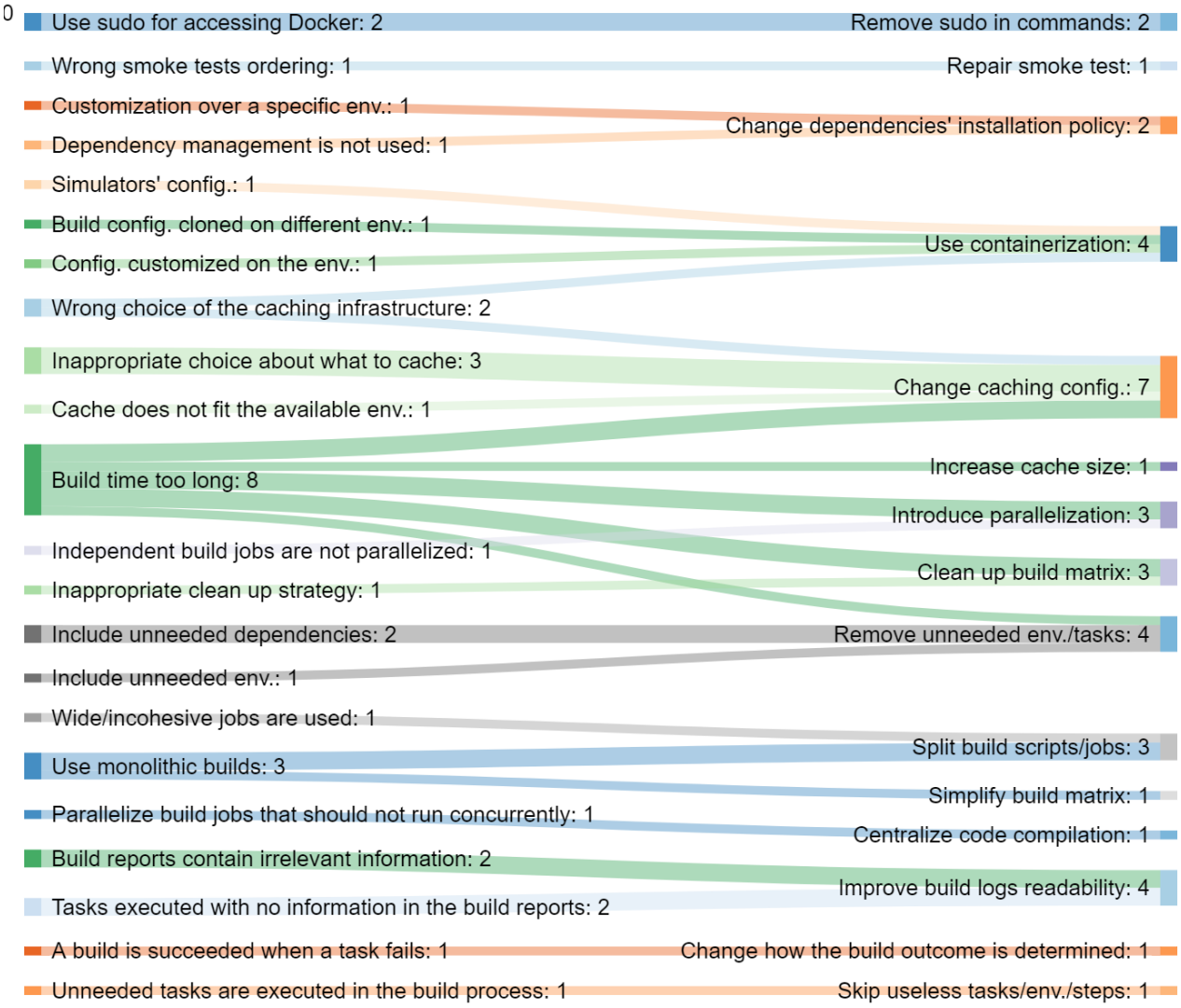

Figure 3: Relations between bad practices and mitigation

simulators, migrating to a new CI/CD framework, changing the simulator setting, and changing the HiL configuration. Quite obviously, in the presence of simulators limited in their functionality, open-source developers tend to use mocking (5) or else adopt a "better" simulator (2), i.e., providing the missing functionality. As regards the way open-source developers deal with the presence of non-deterministic behaviour in the build process, results are in line with the ones provided by the 10 interviewed organizations: for flakiness due to networking, developers use "retry" or "timeout" that is also used when the flakiness belongs to a lack of control over the resources used by the build process. As an example, PR #12373[21] from PX4-AUTOPILOT discusses the need to use timeout since that *"Sometimes Jenkins misses the initial boot after upload and gets stuck"*, while PR #2234[22] in PAPARAZZI, uses timeout since that *"2 seconds is a bit to[o] short in some cases when the router takes longer to reply. 5 seconds gives much better results."*

Open-source developers change the configuration of the simulators in their pipeline to deal with issues in the configuration itself hindering the overall build process definition and execution (2), lack of trustworthiness for their functional correctness (2) and overcome the limits due to the adoption of GUI-based simulators (1). As regards the latter, PR #12781[23] in PX4-AUTOPILOT disables the GUI by changing a parameter within

[21] https://github.com/PX4/PX4-Autopilot/pull/12373
[22] https://github.com/paparazzi/paparazzi/pull/2234
[23] https://github.com/PX4/PX4-Autopilot/pull/12781

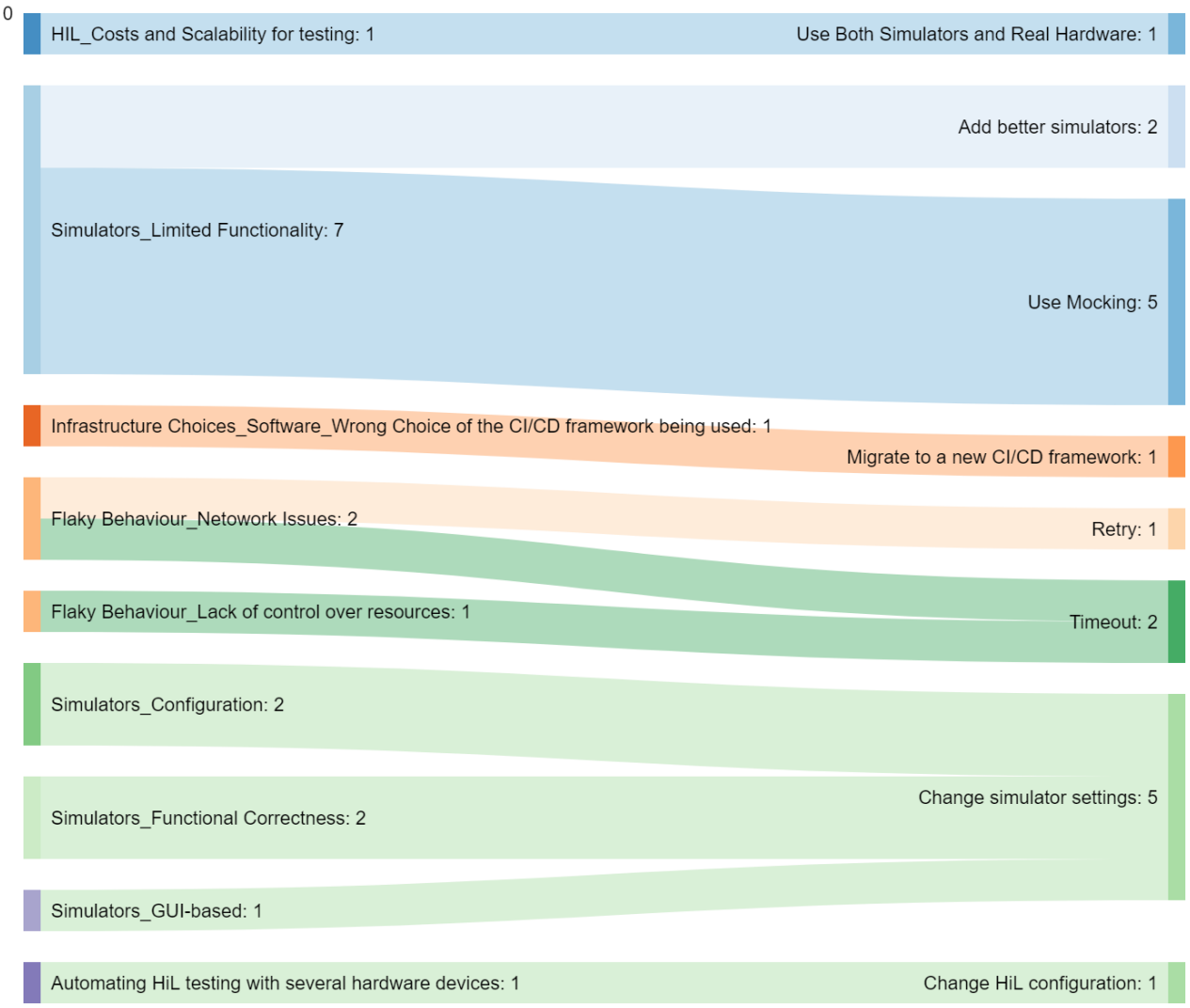

Figure 4: Relations between challenges and mitigation

the simulator's configuration, i.e., *"disable GUI using HEADLESS=1" [...] Now we just need to read the HEADLESS env variable to use it.*

5 Conclusions

In this deliverable, we investigated the adoption, usage, and evolution of CI/CD pipelines for CPS development, by focusing on three different aspects: pipeline setting/triggering, phases and their configuration, and execution environment, e.g., usage and setting of simulators and HiL. The deliverable is based on the analysis of the interviews' transcripts from 10 companies developing CPSs in eight different domains, as well as a manually validated sample of 364 pull requests coming from 10 CPS open-source projects hosted on GitHub, using at least one CI/CD framework.

Crucially, the study behind the deliverable identifies challenges and barriers that industrial and open-source DevOps teams face when adopting or moving to CI/CD for CPS development highlighting that the configuration is highly dependent on the domain.

- **Simulation and hardware mocking pros and cons should be carefully pondered and, where needed, combined with HiL.** Different CPS domains use HiL at different stages. This implies a more/-

less intensive usage of simulators, and varying adoption of in-house implemented simulators or third-party ones. In all cases, simulators may be incomplete, error-prone, or not able to capture some hardware properties, whereas HiL can be expensive or difficult to integrate into the pipeline. Wherever possible, such problems can be mitigated through the use of simulators in continuous builds and HiL in periodic builds, at the cost of late deployment on hardware. In such a context, practitioners are encouraged to consider the technological needs for a CPS pipeline, such as pondering the use of virtualization or containerization. Also, it is desirable to design architectures to ease HiL/simulators switch in the pipeline, and decision-making approaches, e.g., leveraging monitoring and test data, to drive the pipeline reconfiguration, or to find the right level of CPS validation and assessment within CPS CI/CD pipelines [37]. Furthermore, the benefit trade-off analysis for configuring the pipeline should also account for better monitoring, which suggests approaches for combining conventional CI/CD dashboard with monitoring of properties collected from HiL and simulators.

- **The definition and automatic check of the oracles imply complex reasoning on hardware outputs/-multimedia sources.** The CPS execution environment (e.g., simulators or HiL) drastically complicates the definition and automatic check of oracles. The latter requires to ponder several factors: (i) the test scenario (or requirement to assess); (ii) the accepted level of realism in simulations; (iii) the readiness-level or maturity of the hardware proxies used in the pipeline; and (iv) the output sources, e.g., based on actual sensors' data or mocking/synthetic data. Other than that, the oracles are often not scalar values as in traditional software but ranges of values (e.g., time intervals) or signals as highlighted in existing studies on testing for CPS [45, 6]. Looking more broadly at configuring V&V phases within the CPS pipeline, it is important to help practitioners with the early discovery of defects through static analysis. This requires a clear fault modeling in the CPS context (as the ones for autonomous cars [26] and unmanned vehicles [68]), but also to develop CPS-specific linters.
- **The root causes of CPS flakiness are more complex and diverse than for conventional software; specific detection and mitigation strategies are needed.** The complex technological stack, the behavior of simulators and HiL, their (sometimes uncontrollable) unavailability or lack of accessibility, and the mechanisms used to collect test outputs (e.g., sensors or video cameras) require not only to better monitor all possible elements causing flakiness, but also to combine and enhance various mitigation approaches, including checking the status of HiL/simulators, and leveraging the "usual" retries. Our findings highlight that, flaky behaviors in CPS are often due to the complex interacting environment (e.g., lack of complete control on the hardware status) rather than on the order by which the tests are executed, implying that CPS-specific detectors are needed.
- **There is no silver bullet for structuring CI/CD pipeline across different domains.** The "no silver bullet" concept outlined by Brooks [9] is emphasized when it comes to CPS development; even within a single domain, a given program must be able to run/be interfaced with multiple/diverse hardware. Thus, the pipeline often needs ad-hoc configuration, e.g., through build matrices for coping with different environments enacted by simulators or HiL. Furthermore, very often CPS development concerns safety-critical systems, which implies following certain SIL levels [8, 23], such as a pipeline may belong to a validation organization that may have limited/no access to hardware, simulators, or even to the production code. In these complex scenarios, it is highly desirable to develop approaches to recommend, or automatically create, pipeline configurations based on data from similar systems and by learning from past builds and changes within the same pipeline.

We believe that the findings presented in this deliverable are a first step towards supporting DevOps teams in properly use and configure CI/CD for CPS.

6 Appendix

Table 14: Final Catalog of barriers and challenges

Category	ID	Challenge/Barrier	Pull requests (P), Interviews (I)
Resources	B_1	Limited human resources	I
	B_2	Limited availability of software and/or hardware resources	I, P
Domain	B_3	Complex non-functional requirements	I, P
	B_4	HiL not usable, e.g., for safety or security reasons	I
	B_5	Security configuration prevents CD	I
Pipeline Properties	PC_1	Long build execution time	I, P
	PC_2	Build time estimation	I
	PC_3	Static code analysis tools configuration	I
	PC_4	Lack to access the production code from the pipeline	I
	PC_5	CI/CD configuration highly coupled with the environment	I, P
	PC_6	Re-usability of build artifacts	I
	PC_{30}	Licensing requirements for external APIs	P
Thoroughness	PC_7	Development environment detached from the execution environment	I
	PC_8	Detecting deployment-related errors	I
	PC_9	Continuous installation	I
	PC_{10}	Closing the loop introduces performance degradation	I
	PC_{11}	Complexity in closing the loop due to uncontrollable factors	I
	PC_{12}	Complexity in closing the loop due to data collection from the field	I
	PC_{31}	Automating HiL testing with several hardware devices	P
	PC_{32}	Lack of HiL specific smoke test	
Simulators	PC_{13}	Limited in their functionality	I, P
	PC_{14}	Functional correctness	I, P
	PC_{15}	Deal with real-time properties	I, P
	PC_{16}	Interaction with the environment	I, P
	PC_{17}	Accessibility	I
	PC_{33}	Configuration	P
	PC_{34}	Choose among different simulators	P
	PC_{35}	GUI-based	P
HiL	PC_{18}	Availability	I
	PC_{19}	Automated deployment on HiL	I
	PC_{20}	Test Automation on HiL	I
	PC_{21}	Costs and scalability	I, P
	PC_{22}	HiL standalone	I
	PC_{37}	Memory Management	P
	PC_{38}	Parallelization and Resources Blockage	P
Flaky Behaviour	PC_{23}	Dependency installation	I
	PC_{24}	Features' interaction	I
	PC_{25}	HiL availability	I
	PC_{26}	HiL inputs	I
	PC_{27}	Lack of control over resources	I, P
	PC_{28}	Network issues	I, P
	PC_{29}	Timing issues	I, P

Table 15: Final catalog of mitigation

Category	M_{ID}	Action	Pull requests (P), Interviews(I)
Flakiness	M_1	Adjust pipeline configuration	I
	M_2	Check device availability/status	I
	M_3	Fix the code	I
	M_4	Rewrite flaky tests	I
	M_5	Use retry	P, I
	M_6	Use timeout	P, I
	M_7	Use a high granularity over the test oracle	I
Slow Builds	M_8	Use a CI/CD configuration based on the type of change	I
	M_9	Use incremental builds and small changes	I
	M_{10}	Structure the repository in modules	I
	M_{11}	Use parallelization	P, I
	M_{12}	Use nightly builds	I
	M_{13}	Test Case prioritization	I
Simulators	M_{14}	Rely on both simulators and HiL	P, I
	M_6	Use timeout	I
	M_{15}	Add better simulators	P
	M_{16}	Use mocking	P, I
	M_{17}	Change simulators' setting	P
HiL	M_{14}	Rely on both simulators and HiL	P, I
	M_{18}	Change HiL Configuration	P
	M_{19}	Green build rule	I
Continuous Installation	M_{20}	Use containerization	I
Resources	M_9	Use incremental builds and small changes	I
	M_{13}	Test Case prioritization	I
Domain	M_{16}	Use mocking	I

References

[1] MISRA:2012 Guidelines for the use of the C language in critical systems. https://www.academia.edu/40301277/MISRA_C_2_012_Guidelines_for_the_use_of_the_C_language_in_critical_systems.pdf. Accessed: 2021-08-30.

[2] Sara Abbaspour Asadollah, Rafia Inam, and Hans Hansson. A survey on testing for cyber physical system. In *Testing Software and Systems*, pages 194–207, Cham, 2015. Springer Intern. Publishing.

[3] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. Which commits can be CI skipped? *IEEE Trans. Software Eng.*, 47(3):448–463, 2021.

[4] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C Briand, and Thomas Stifter. Testing autonomous cars for feature interaction failures using many-objective search. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 143–154. IEEE, 2018.

[5] Zuleyha Akgun, Murat Yilmaz, and Paul M. Clarke. Assessing application lifecycle management (ALM) potentials from an industrial perspective. In *27th European Conference on Systems, Software and Services Process Improvement, Düsseldorf, Germany*, pages 326–338, 2020.

[6] Aitor Arrieta, Shuai Wang, Ainhoa Arruabarrena, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Multi-objective black-box test case selection for cost-effectively testing simulation models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1411–1418, 2018.

[7] Roberto Bagnara, Abramo Bagnara, and Patricia M Hill. The misra c coding standard and its role in the development and analysis of safety-and security-critical embedded software. In *International Static Analysis Symposium*, pages 5–23. Springer, 2018.

[8] Ron Bell. Introduction to iec 61508. In *ACM International Conference Proceeding Series*, volume 162, pages 3–12. Citeseer, 2006.

[9] Frederick P. Brooks. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.

[10] L. Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54, 2015.

[11] Lianping Chen. Continuous delivery: Overcoming adoption challenges. *Journal of Systems and Software*, 128:72 – 86, 2017.

[12] Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: real-world challenges and research innovations. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 4–5, 2019.

[13] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. In *1st Annual Conference on Robot Learning, CoRL 2017*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 2017.

[14] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. Perceval: Software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 1–4, 2018.

[15] Paul Duvall, Stephen M. Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

[16] Paul M. Duvall. Continuous integration. patterns and antipatterns. *DZone refcard #84*, 2010.

[17] Paul M. Duvall. Continuous delivery: Patterns and antipatterns in the software life cycle. *DZone refcard #145*, 2011.

[18] Moritz Eck, Fabio Palomba, Marco Castelluccio, and Alberto Bacchelli. Understanding flaky tests: the developer's perspective. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 830–840, 2019.

[19] Omar Elazhary, Margaret-Anne D. Storey, Neil A. Ernst, and Elise Paradis. ADEPT: A socio-technical theory of continuous integration. In *43rd IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2021, Madrid, Spain, May 25-28, 2021*, pages 26–30, 2021.

[20] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A Ernst, and Margaret-Anne Storey. Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering*, 2021.

[21] Erik Flores-García, Goo-Young Kim, Jinho Yang, Magnus Wiktorsson, and Sang Do Noh. Analyzing the characteristics of digital twin and discrete event simulation in cyber physical systems. In *Advances in Production Management Systems. Towards Smart and Digital Manufacturing*, volume 592 of *IFIP Advances in Information and Communication Technology*, pages 238–244. Springer, 2020.

[22] Martin Fowler and Matthew Foemmel. Continuous integration. `https://web.archive.org/web/20200522100521/https://martinfowler.com/articles/originalContinuousIntegration.html`. Accessed: 2021-11-21.

[23] Heinz Gall. Functional safety iec 61508 / iec 61511 the impact to certification and the user. In *2008 IEEE/ACS International Conference on Computer Systems and Applications*, pages 1027–1031, 2008.

[24] Keheliya Gallaba and Shane McIntosh. Use and misuse of continuous integration features: An empirical study of projects that (mis) use travis ci. *IEEE Trans. Software Eng.*, 46(1):33–50, 2018.

[25] Alessio Gambi, Tri Huynh, and Gordon Fraser. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 257–267. ACM, 2019.

[26] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, and Qi Alfred Chen. A comprehensive study of autonomous vehicle bugs. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 385–396, 2020.

[27] Smitha Gautham, Athira Varma Jayakumar, Abhi Rajagopala, and Carl Elks. Realization of a model-based devops process for industrial safety critical cyber physical systems. In *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 597–604, 2021.

[28] Jairo Giraldo, Esha Sarkar, Alvaro A. Cardenas, Michail Maniatakos, and Murat Kantarcioglu. Security and privacy in cyber-physical systems: A survey of surveys. *IEEE Design & Test*, 34(4):7–17, August 2017.

[29] Carlos A. González, Mojtaba Varmazyar, Shiva Nejati, Lionel C. Briand, and Yago Isasi. Enabling model testing of cyber-physical systems. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, page 176–186, New York, NY, USA, 2018. Association for Computing Machinery.

[30] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[31] Philipp Helle, Wladimir Schamai, and Carsten Strobel. Testing of autonomous systems - challenges and current state-of-the-art. *INCOSE International Symposium*, pages 571–584, 2016.

[32] Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. Trade-offs in continuous integration: Assurance, security, and flexibility. In *Proceedings of the 25th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2017*, 2017.

[33] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016.

[34] David L. Hoover. Textual variation, text-randomization, and microanalysis. In *DH*, pages 223–225. Alliance of Digital Humanities Organizations (ADHO), 2016.

[35] Suzette Johnson, Harry Koehneman, Diane LaFortune, Dean Leffingwell, Stephen Magill, Steve Mayner, Avigail Ofer, Robert Stroud, Anders Wallgren, and Robin Yeman. *Industrial DevOps - Applying DevOps and Continuous Delivery to Significant Cyber-Physical Systems*. National Academies Press, 2017.

[36] Nidhi Kalra and Susan Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 12 2016.

[37] Rick Kazman, Mark Klein, and Paul Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Carnegie Mellon Uiversity, Software Engineering Institute, 2000.

[38] Klaus Krippendorff. Reliability in content analysis: Some common misconceptions and recommendations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 30(3):411–433, 2004.

[39] Wing Lam, Patrice Godefroid, Suman Nath, Anirudh Santhiar, and Suresh Thummalapenta. Root causing flaky tests in a large-scale industrial setting. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, page 101–111. ACM, 2019.

[40] Wing Lam, Stefan Winter, Angello Astorga, Victoria Stodden, and Darko Marinov. Understanding reproducibility and characteristics of flaky tests through test reruns in java projects. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE, Coimbra, Portugal, October 12-15, 2020*, pages 403–413, 2020.

[41] Wing Lam, Stefan Winter, Anjiang Wei, Tao Xie, Darko Marinov, and Jonathan Bell. A large-scale longitudinal study of flaky tests. *Proc. ACM Program. Lang.*, 4(OOPSLA):202:1–202:29, 2020.

[42] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[43] Ivano Malavolta, Grace Lewis, Bradley Schmerl, Patricia Lago, and David Garlan. How do you architect your robots? state of the practice and guidelines for ros-based systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '20, page 31–40, New York, NY, USA, 2020. ACM.

[44] Torvald Mårtensson, Daniel Ståhl, and Jan Bosch. Continuous integration applied to software-intensive embedded systems–problems and experiences. In *International Conference on Product-Focused Software Process Improvement*, pages 448–457. Springer, 2016.

[45] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 372–384. IEEE, 2020.

[46] Helena Holmstrom Olsson, Hiva Alahyari, and Jan Bosch. Climbing the "stairway to heaven" – a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *Proceedings of the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '12, pages 392–399, Washington, DC, USA, 2012. IEEE Computer Society.

[47] Heejong Park, Arvind Easwaran, and Sidharta Andalam. Challenges in digital twin development for cyber-physical production systems. In *Cyber Physical Systems. Model-Based Design*, pages 28–48, Cham, 2019. Springer International Publishing.

[48] Gustavo Pinto, Breno Miranda, Supun Dissanayake, Marcelo d'Amorim, Christoph Treude, and Antonia Bertolino. What is the vocabulary of flaky tests? In *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, pages 492–502, 2020.

[49] Akond Rahman, Chris Parnin, and Laurie Williams. The seven sins: Security smells in infrastructure as code scripts. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 164–175. IEEE Press, 2019.

[50] Steven P. Reiss. Automatic code stylizing. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, pages 74–83, 2007.

[51] Vincenzo Riccio and Paolo Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '20. Association for Computing Machinery, 2020.

[52] Terry Rowlands, Neal Waddell, and Bernard McKenna. Are we there yet? a technique to determine theoretical saturation. *J. Comput. Inf. Syst.*, 56(1):40–47, 2016.

[53] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. Continuous deployment at facebook and OANDA. In *Companion proceedings of the 38th International Conference on Software Engineering (ICSE Companion)*, pages 21–30, 2016.

[54] August Shi, Jonathan Bell, and Darko Marinov. Mitigating the effects of flaky tests on mutation testing. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA*, pages 112–122, 2019.

[55] Hesham Shokry and Mike Hinchey. Model-based verification of embedded software. 2009.

[56] Sebastian Sontges and Matthias Althoff. Computing the drivable area of autonomous road vehicles in dynamic road scenes. *IEEE Trans. Intell. Transp. Syst.*, 19(6):1855–1866, 2018.

[57] Donna Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[58] Daniel Ståhl and Jan Bosch. Automated software integration flows in industry: a multiple-case study. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 54–63. ACM, 2014.

[59] Daniel Ståhl and Jan Bosch. Modeling continuous integration practice differences in industry software development. *J. Syst. Softw.*, 87:48–59, 2014.

[60] Daniel Ståhl, Kristofer Hallén, and Jan Bosch. Continuous integration and delivery traceability in industry: Needs and practices. In *42th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016, Limassol, Cyprus, August 31 - Sept. 2, 2016*, pages 68–72. IEEE Computer Society, 2016.

[61] Sirasak Tepjit, Imre Horváth, and Zoltán Rusák. The state of framework development for implementing reasoning mechanisms in smart cyber-physical systems: A literature review. *Journal of Computational Design and Engineering*, 6(4):527–541, 04 2019.

[62] Martin Törngren and Ulf Sellgren. *Complexity Challenges in Development of Cyber-Physical Systems*, pages 478–503. Springer International Publishing, Cham, 2018.

[63] Miriam Ugarte Querejeta, Leire Etxeberria, and Goiuria Sagardui. Towards a devops approach in cyber physical production systems using digital twins. In *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, pages 205–216, Cham, 2020. Springer International Publishing.

[64] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar T. Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *ESEC/SIGSOFT FSE*, pages 805–816. ACM, 2015.

[65] Carmine Vassallo, Sebastian Proksch, Harald Gall, and Massimiliano Di Penta. Automated reporting of anti-patterns and decay in continuous integration. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, Canada, May 25 - 31, 2019*. IEEE, 2019.

[66] Carmine Vassallo, Sebastian Proksch, Anna Jancso, Harald C Gall, and Massimiliano Di Penta. Configuration smells in continuous delivery pipelines: a linter and a six-month study on gitlab. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 327–337, 2020.

[67] Carmine Vassallo, Fiorella Zampetti, Daniele Romano, Moritz Beller, Annibale Panichella, Massimiliano Di Penta, and Andy Zaidman. Continuous delivery practices in a large financial organization. In *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 41–50, 2016.

[68] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei Sui. An exploratory study of autopilot software bugs in unmanned aerial vehicles. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 20–31, 2021.

[69] Rajaa Vikhram Yohanandhan, Rajvikram Madurai Elavarasan, Premkumar Manoharan, and Lucian Mihet-Popa. Cyber-physical power system (CPPS): A review on modeling, simulation, and analysis with cyber security applications. *IEEE Access*, 8:151019–151064, 2020.

[70] Fiorella Zampetti, Salvatore Geremia, Gabriele Bavota, and Massimiliano Di Penta. Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 471–482. IEEE, 2021.

[71] Fiorella Zampetti, Carmine Vassallo, Sebastiano Panichella, Gerardo Canfora, Harald C. Gall, and Massimiliano Di Penta. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering*, 25(2):1095–1135, 2020.

[72] Eleni Zapridou, Ezio Bartocci, and Panagiotis Katsaros. Runtime verification of autonomous driving systems in carla. In *Runtime Verification*, pages 172–183, Cham, 2020. Springer International Publishing.

[73] Peilun Zhang, Yanjie Jiang, Anjiang Wei, Victoria Stodden, Darko Marinov, and August Shi. Domain-specific fixes for flaky tests with wrong assumptions on underdetermined specifications. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 50–61, 2021.

[74] Celal Ziftci and Diego Cavalcanti. De-flake your tests : Automatically locating root causes of flaky tests in code at google. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 736–745, 2020.

[75] Behrouz Zolfaghari, Reza M. Parizi, Gautam Srivastava, and Yoseph Hailemariam. Root causing, detecting, and fixing flaky tests: State of the art and future roadmap. *Softw. Pract. Exp.*, 51(5):851–867, 2021.