

Continuous Integration and Delivery practices for Cyber-Physical systems: An interview-based study

Zampetti, Fiorelli; Tamburri, Damian A.; Panichella, Sebastiano; Panichella, A.; Di Penta, Massimiliano; Canfora, Gerardo

DOI

[10.1145/3571854](https://doi.org/10.1145/3571854)

Publication date

2022

Document Version

Accepted author manuscript

Published in

ACM Transactions on Software Engineering and Methodology

Citation (APA)

Zampetti, F., Tamburri, D. A., Panichella, S., Panichella, A., Di Penta, M., & Canfora, G. (2022). Continuous Integration and Delivery practices for Cyber-Physical systems: An interview-based study. *ACM Transactions on Software Engineering and Methodology*, 32(3), Article 73. <https://doi.org/10.1145/3571854>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

Continuous Integration and Delivery Practices for Cyber-Physical Systems: An Interview-Based Study

1
2
3
4
5 FIORELLA ZAMPETTI, University of Sannio, Italy

6 DAMIAN A. TAMBURRI, Eindhoven University of Technology and the Jheronimus Academy of Data
7 Science (JADS), The Netherlands

8 SEBASTIANO PANICHELLA, Zurich University of Applied Sciences, Switzerland

9 ANNIBALE PANICHELLA, Technical University of Delft, The Netherlands

10 GERARDO CANFORA, University of Sannio, Italy

11 MASSIMILIANO DI PENTA, University of Sannio, Italy

12
13 Continuous Integration and Delivery (CI/CD) practices have shown several benefits for software development
14 and operations, *e.g.*, faster release cycles and early discovery of defects. For Cyber-Physical System (CPS)
15 development, CI/CD can help achieving required goals, such as high dependability, yet it may be challenging to
16 apply. This paper empirically investigates challenges, barriers, and their mitigation occurring when applying
17 CI/CD practices to develop CPSs in 10 organizations working in 8 different domains. The study has been
18 conducted through semi-structured interviews, by applying an open card sorting procedure together with
19 a member-checking survey within the same organizations, and by validating the results through a further
20 survey involving 55 professional developers. The study reveals several peculiarities in the application of
21 CI/CD to CPSs. These include the need for (i) combining continuous and periodic builds, while balancing
22 the use of Hardware-in-the-Loop (HiL) and simulators; (ii) coping with difficulties in software deployment
23 (iii) accounting for simulators and HiL differing in their behavior; and (vi) combining hardware/software
24 expertise in the development team. Our findings open the road towards recommenders aimed at supporting
25 the setting and evolution of CI/CD pipelines, as well as university curricula requiring interdisciplinarity, such
26 as knowledge about hardware, software, and their interplay.

26 CCS Concepts: • **Software and its engineering** → **Maintaining software**.

27
28 Additional Key Words and Phrases: Continuous Integration and Delivery, Cyber-Physical Systems, Empirical
29 Software Engineering

30 ACM Reference Format:

31 Fiorella Zampetti, Damian A. Tamburri, Sebastiano Panichella, Annibale Panichella, Gerardo Canfora, and Mas-
32 similiano Di Penta. 2021. Continuous Integration and Delivery Practices for Cyber-Physical Systems: An
33 Interview-Based Study. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2021), 44 pages.

34
35 Authors' addresses: Fiorella Zampetti, fiorellazampetti@gmail.com, University of Sannio, Via Traiano, 9, Benevento, Italy,
36 82100; Damian A. Tamburri, d.a.tamburri@tue.nl, Eindhoven University of Technology and the Jheronimus Academy
37 of Data Science (JADS), St. Janssingel 92, 's-Hertogenbosch, The Netherlands, 5211 DA; Sebastiano Panichella, [panc@](mailto:panc@zhaw.ch)
38 [zhaw.ch](mailto:panc@zhaw.ch), Zurich University of Applied Sciences, Steinberggasse 13, Winterthur, Switzerland, 8400; Annibale Panichella,
39 a.panichella@tudelft.nl, Technical University of Delft, Van Mourik Broekmanweg 6, Delft, The Netherlands, 2628 XE;
40 Gerardo Canfora, canfora@unisannio.it, University of Sannio, Via Traiano, 9, Benevento, Italy, 82100; Massimiliano Di
41 Penta, dipenta@unisannio.it, University of Sannio, Via Traiano, 9, Benevento, Italy, 82100.

42
43 Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee
44 provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and
45 the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored.
46 Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires
47 prior specific permission and/or a fee. Request permissions from permissions@acm.org.

48 © 2022 Association for Computing Machinery.

49 1049-331X/2021/1-ART1 \$15.00

<https://doi.org/>

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 [32, 51, 73, 74].

In such a context, adopting effective Continuous Integration and Delivery (CI/CD) practices off the DevOps menu is extremely relevant for setting the execution environment, *e.g.*, Hardware-in-the-Loop (HiL) or simulators. Even though 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 [75], its application implies overcoming barriers and challenges [9, 33].

When enacting CI/CD for CPSs, it is expected that further barriers and challenges will arise. In general, existing CI/CD technology cannot be applied to CPSs as is [39]. On the one hand, 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 make the application of CI/CD challenging at best. On the other hand, while for conventional software systems good and bad practices for applying CI/CD have been defined [15, 85], for what concerns CPSs, the practice is still immature to be able to do so. Specifically, the combination of (very diversified and evolving) hardware devices and software, the complex execution scenarios, and the need for simulating hardware components during some build stages introduce new facets that must be considered when setting up a CPS development process, and in particular CI/CD pipelines for CPS development.

This paper aims to empirically investigate the challenges and barriers practitioners encounter while setting up and maintaining a CI/CD pipeline for CPSs, as well as the mitigation strategies adopted to deal with them. Specifically, the study has been conducted through (i) semi-structured interviews with 10 industrial practitioners involved in CPS development for 8 different domains, *i.e.*, aerospace, automotive, energy, healthcare, railways, robotics, identification technology (*i.e.*, Radio Frequency Identification - RFID), and acoustic sensors, (ii) by applying open coding [35] and card sorting [68] to the interview transcripts, (iii) by conducting a member-checking survey within the same organizations involved in the interviews aiming at corroborating the relations between challenges/barriers and mitigation strategies, and (iv) by assessing the relevance of the identified challenges/barriers and related mitigation through a survey involving 55 practitioners involved in CPS development for 9 different domains.

We start by characterizing the CI/CD practices of the interviewed organizations, focusing more on their build automation processes. In doing this, we target three aspects of CI/CD for CPSs, namely (i) the pipeline setting, (ii) the involved phases (*e.g.*, static analysis, testing, delivery, etc.), and (iii) the usage and configuration of simulators and/or HiL. After that, we look at challenges and barriers the organizations encounter, as well as mitigation strategies being adopted to deal with them.

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, as well as to staff projects properly coordinate resources with different skills and expertise and acquire equipment. Furthermore, results highlight directions in which education for CPS development must be improved. This includes not only covering interdisciplinary topics between software development, measurements, and automated control, but also a proper introduction to software architectures and design principles, making CPS development flexible enough when switching between simulators and HiL. Last but not least, we identify areas where further research is required,

among others domain-aware decision making, the integration of simulators and HiL in the pipeline, and further research in the area of test automation and flakiness detection/avoidance. The specificity of each CPS not only makes some lessons hard to generalize (each pipeline tends to be very different from others), but this also poses challenges when leveraging machine learning approaches upon developing recommender systems.

Paper structure. The rest of the paper is organized as follows. As basis for the study, Section 2 discusses the relevant literature. Section 3 describes the study methodology, while Section 4 reports and discusses the study results. Section 5 details the study implications, while threats to the study validity are discussed in Section 6. Finally, Section 7 concludes the paper and outlines future directions.

The study material (after redacting interview transcripts) is available online [84].

2 RELATED WORK

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. Note that this is not an exhaustive systematic literature review on the topic, but rather it points out papers discussing challenges in CPS development and in CI/CD. Finally, it is important to highlight that, while challenges related to CPS development are already investigated from previous literature, to the best of our knowledge, there is very limited empirical evidence on how such challenges translate when setting a CI/CD pipeline for CPS development.

2.1 Development of Cyber-Physical Systems

CPSs are more complex and difficult to design, develop, test, and integrate than conventional software systems [32, 51, 73, 74]. Specifically, Törngren *et al.* [74] 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 verification of safety requirements [27], as well as testing encapsulating model-in-the-loop (MiL) [66], software-in-the-loop (SiL), and hardware-in-the-loop (HiL) [2]. With respect to previous studies, we investigate how CPS complexity impacts the setting of CI/CD pipelines, and how developers deal with such complexity.

Considering the costs, risks, and complexity of conducting system testing in a real environment [12, 40], simulation is becoming one of the cornerstones in developing and validating CPSs. CPS developers mainly rely on basic simulation models [29, 67], as well as rigid body [50, 86] and soft body simulation environments [25, 62]. The usage of CPS simulation environments enables automated test generation and execution [37, 54]. However, the limited budget allocated for testing activities and the virtually infinite testing space pose challenges for adequately exercising the CPS behavior [4, 20, 82]. We complement previous studies by looking at the challenges, barriers, and related mitigation strategies when integrating and combining simulators and HiL in CI/CD to support the development, V&V, and evolution of CPSs.

Related to DevOps applications in a CPS context, Park *et al.* [56] 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 twin and the physical environment. Instead of only looking at automating the production process, we focus more on the CI/CD process for CPS development and evolution.

Finally, Mårtensson *et al.* [52] 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

Table 1. Challenges in CPS development from previous literature.

Ref.	CPS-related development challenges
[74]	Environment complexity, co-designing hardware and software
[27]	Test generation/automation, verification of safety requirements
[2]	Integration of MiL, SiL, and HiL
[12, 40]	Where testing is performed (HiL vs. simulators)
[25, 29, 50, 62, 67, 86]	Implementation of simulators
[4, 20, 82]	Simulator challenges/adequacy
[52]	Standards, long build, security, architecture, test environments of embedded systems
[56]	Digital twin adoption in manufacturing and related design challenges

to adopt CI for embedded systems. While using different research methods, our study is wider than Mårtensson *et al.* (10 semi-structured interviews, plus an external survey with 55 participants vs. case studies with 2 companies), and considers the whole CI/CD process from development to delivery to the customer side. Finally, while we confirm findings from Mårtensson *et al.* [52], our study deepens the analysis of different CI/CD aspects (*e.g.*, setting, phases, and execution environment) for CPSs, and not only in relation to seven CI cornerstones.

Table 1 summarizes the main challenges 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. 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.* [34] found that CI is becoming very popular in open source projects. The latter is also true in industry, even if Ståhl and Bosh [69, 70] found that there is not a uniform adoption of CI in industry. Furthermore, Vasilescu *et al.* [75] showed that CI practices improve developers' productivity without negatively impacting the overall code quality. Finally, Ståhl *et al.* [71], in a study involving three companies, found that the lack of traceability may prevent the application of CI in conventional software systems.

From a different perspective, Elazhary *et al.* [18] looked at the extent to which companies follow the CI practices by Fowler and Foemmel [21] 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, relate to CI/CD application for CPS development. In a different study, Elazhary *et al.* [17] 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.* [79] 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).

Finally, deepening the continuous delivery practice, Chen [10] analyzed four years of CD adoption in a multi-billion-euro company, and identified a list of challenges related to CD adoption. Savor *et al.* [64], instead, by analyzing 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.

197 Differently from previous studies, our goal is to shed light on the CI/CD process focusing on the
198 peculiarities of CPS development.

199 2.3 CI/CD barriers and bad practices

201 Different authors studied barriers and/or challenges in adopting CI/CD. These were initially iden-
202 tified by Duvall *et al.* [13], and are related to the need for maintaining a fully automated build
203 process, handling dependencies, having different levels of builds, and coping with different target
204 environments.

205 Hilton *et al.* [33] studied barriers developers encounter when moving toward CI, *i.e.*, quality
206 assurance, security, and flexibility. Olsson *et al.* [55], instead, looked at the challenges faced while
207 migrating towards CD: the complexity of the deployment environment, the need to achieve timely
208 delivery, and the lack of a complete overview of all the development projects.

209 Previous research also found that CI/CD may be wrongly applied, leading to bad practices.
210 Specifically, CI/CD antipatterns have been defined by Duvall [15], and empirically elicited by
211 Zampetti *et al.* [85] from interviews and Stack Overflow posts. Our study is complementary to
212 that although, where appropriate, we compare the practices observed in our context (CPS-specific)
213 with bad practices recommendations from previous studies. Researchers have developed different
214 kinds of tools to detect and remove antipatterns from CI configuration files [23, 78], analyzing the
215 pipeline aging by observing its execution [77], skipping builds [3], or coping with security-related
216 issues in infrastructure-as-code [60].

217 To the best of our knowledge, there is no such broad investigation on the application of CI/CD
218 in CPS development and evolution, as well as the challenges and barriers faced together with
219 mitigation strategies to overcome them.

221 3 EMPIRICAL STUDY DEFINITION AND PLANNING

222 The *goal* of this study is to investigate the CI/CD practices for CPS development, to identify
223 challenges and barriers encountered in such practices, together with mitigation strategies adopted
224 to overcome them. The *perspective* is of researchers interested to support developers in configuring
225 CI/CD pipelines for CPSs, and practitioners setting, using and evolving CI/CD pipelines for CPS
226 development. The *context* from which we have inferred the set of challenges and barriers with related
227 mitigation strategies encountered when setting or evolving CI/CD pipeline for CPS development
228 consists of 10 organizations developing CPSs for 8 different domains. To assess the identified set
229 of challenges/barriers and related mitigation strategies, we have surveyed 55 practitioners (not
230 involved in the first step of this study) developing CPSs for 9 different domains.

231 We start by creating organizational profiles by looking at the CI/CD practices adopted by the
232 interviewed organizations, and in general all the practices the organizations are adopting to
233 automate different stages of a build. Specifically, we look at the conditions that determine (i) the
234 setting of the CI/CD pipeline, *e.g.*, whether an incremental build is used, when the build is triggered,
235 or whether build matrices are used; (ii) the phases instantiated in the pipeline, *e.g.*, static analysis,
236 various testing levels, or deployment; and (iii) the use of simulators and HiL in the context of the
237 CI/CD pipeline.

238 The study addresses the following two research questions:

- 239
- 240 • **RQ₁**: *What are the challenges and barriers respondents encounter, and how do developers*
241 *deal with them?* After having characterized the CI/CD and build automation practices, we
242 investigate the challenges (*e.g.*, the need to cope with a slow build or flakiness, or with phases
243 not easy to automate) and barriers (*e.g.*, limited availability of software and/or hardware
244 resources) encountered by the interviewed organizations when dealing with the setting

or evolution of the CI/CD pipeline for CPS development. Furthermore, we highlight the strategies (e.g., to adopt a pipeline that relies on both simulators and HiL in different build stages) adopted by the interviewed organizations to deal with challenges and barriers. We validated the relations between challenges/barriers and mitigation strategies through a member-checking survey with the same interviewees or practitioners belonging to the same team of the interviewees involved in the semi-structured interviews.

- **RQ₂**: *How relevant are the identified CI/CD challenges/barriers and their mitigation for practitioners involved in CPS development?* While in the previous research question we identified a set of challenges and barriers with related mitigation strategies as experienced by our interviewees, this research question aims at performing an external validation by surveying practitioners involved in the setting, evolution or usage of a CI/CD pipeline for CPS development.

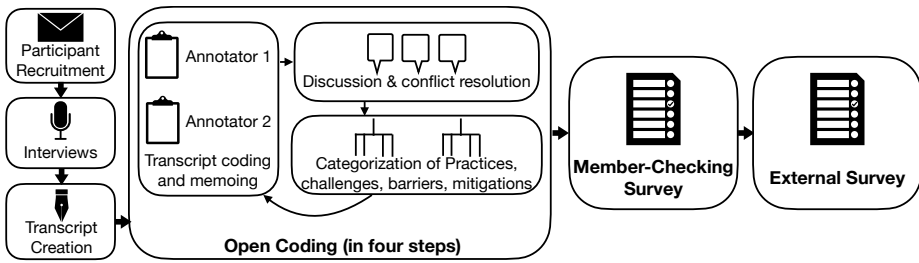


Fig. 1. Study methodology.

The study methodology used for addressing the research questions is depicted in Figure 1 and described in the following. After having recruited participants to be involved in the semi-structured interviews through personal knowledge, we conducted the interviews and transcribed their content. Note that, since this is an exploratory study, we prefer to rely on convenience sampling, as previously done in literature [18, 33]. This is because practitioners involved in CPS development represent a hidden population, therefore we did not have a sampling frame [8]. The latter helps us to conveniently reach a suitable number of study participants. After that, we performed an incremental (in four steps) open coding [35] of the transcripts, discussing the independent coding of multiple annotators, solving conflicts, and creating, through a card sorting strategy [68], categorizations for practices, as well as for barriers, challenges, and mitigation strategies. The relationships between challenges/barriers and mitigation strategies have been validated through a member-checking survey, and, finally we performed a further survey to validate our findings beyond the interview context.

3.1 Data Collection: Semi-structured interviews

We defined the interview structure through an iterative process, which started from the existing knowledge on the topic summarized in Table 1 (see Section 2). From such knowledge, all theoretical pending points were distilled and matched with interview structure areas and questions for each interview structure area. As summarized in Table 2, we start with demographics about the organization and the interviewee, and get a first glance at the 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

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

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 [11]. After that, we investigated how the interviewees configure the overall CI/CD pipeline in terms of build triggering strategies and the possibility to handle different pipeline configurations, each one environment-specific. The interview ends with general questions about the main benefits achieved, barriers encountered, and challenges to tackle when configuring and evolving the CI/CD pipeline.

Table 3. Participant Demographics

Org _{ID}	Organization			CPS Exp. (Y)
	Domain	Size	Role	
O ₁	Aerospace	Small	R&D Manager	8
O ₂	Healthcare	Large	DevOps Architect	18
O ₃	Acoustic Sensors	Small	SW and HW Integrator	6
O ₄	Robotics	Medium	Team Leader	7
O ₅	Automotive	Large	R&D Manager	20
O ₆	Aerospace	Large	R&D Manager	20
O ₇	Railways	Large	SW and HW Integrator	10
O ₈	Railways	Micro	Team Leader	25
O ₉	Identification Technology	Micro	Software Engineer	3
O ₁₀	Energy	Large	Project Leader	5

Interview participant selection. The interview participants have been selected based on personal knowledge, with the goal of identifying experienced practitioners over the theoretical constructs (CI/CD pipelines for CPS) under investigation. The resulting study size (10) is not particularly high, yet on the same order of magnitude as similar interview-based studies on CI/CD [18, 33] (although the study by Hilton *et al.* was followed by a larger survey). It has to be considered also that, differently from previous studies, we targeted a very specific development domain and technology (i.e., application of CI/CD for CPSs in industrial settings). After participants accepted our invitation, we gave them an overview of the questions to expect in the interview, to allow them to gather any additional information.

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

352 As it will be clearer later, we intentionally selected participants having different maturity levels
353 in the implementation of a CI/CD pipeline for CPSs. That is, we also included organizations that,
354 while having experience in setting CI/CD pipelines, only partially automated CPS builds, without
355 having a full-fledged CI/CD pipeline. This allowed us to understand, in those cases, how they
356 automated certain phases, as well as the reasons why they are still facing challenges in having a
357 complete CI/CD pipeline.

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

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

371

372

373 3.2 Data Analysis from interview transcripts

374 Two authors, experts of the domain, (hereinafter referred to as "coders"), independently used
375 online spreadsheets to assign codes (*i.e.*, open coding) to sentences in the transcripts. The coding
376 was carried out following the approach illustrated by Hoover [35], *i.e.*, annotating a code near
377 sentences of the transcript. A code is defined as a mnemonic label identifying a concept defined in
378 a text fragment, *e.g.*, by applying the label 'TEST' to any part of text reflecting a software testing
379 activity. Wherever appropriate, the coder added a memo that could be leveraged to better explain
380 the observed phenomenon, as well as to identify possible relationships between codes dealing with
381 different aspects of the CI/CD pipeline setting and evolution.

382 Open coding has been performed over four subsequent sessions by arranging the 10 interview
383 transcripts into four groups. Each group included two, three, four, and one interview, respectively.
384 After each coding session, the coders held a discussion meeting, in which similar codes created
385 by multiple coders were merged, and conflicts were resolved. After each round, we computed
386 the Krippendorff α [45] to determine the achieved level of agreement. The obtained α values for
387 the four iterations were 0.65 (close to the minimum acceptability of $\alpha = 0.66$ [45]), 0.71, 0.69, and
388 0.86 (substantial agreement). Starting from the second iteration, the coders could reuse, through a
389 drop-down cell, codes created during previous iterations, or create new ones. Note that, to further
390 limit agreement by chance, each code annotation was reviewed during the meetings, not just the
391 disagreements.

392

393 During the discussion meetings, broad groups of codes were also defined. For instance, we
394 distinguished codes belonging to the CI/CD pipeline from those related to the development process.
395 Also, we started grouping codes belonging to different phases of the pipeline, and codes related
396 to challenges, barriers, and mitigation strategies. Such a categorization started during the first
397 discussion meeting and then was refined over the next ones. After the first two sessions of the open
398 coding (after the first session the set of codes was too immature for this purpose), three researchers
399 iteratively produced—by adopting a card sorting strategy [68]—the first version of a mind map
400 grouping codes into categories. Such a mind map has been used as a support to ease the subsequent
401 open coding phases and to evaluate the extent to which non-leaf nodes were saturated. Note that
402 we do not expect a full saturation [63] in this study, due to the high diversity of the considered
403 application domains. The mind map was then refined after each subsequent coding phase.

404 Overall, we identified a set of 179 codes, which led to the construction of a categorization of
405 codes explaining the phenomenon, organized across 43 high-level categories.

406 Finally, the two coders performed three iterations over the transcripts, codes, and memos to
407 derive relations between different codes. For instance, it is possible that process constraints (*e.g.*, the
408 need to use a specific type of simulator or tool imposed by the domain, or to adopt certain coding
409 standards), introduce challenges while setting the pipeline (*e.g.*, the need to cope with phases not
410 easy to automate, or slow build and flakiness) that may be addressed by relying on a particular
411 mitigation strategy (*e.g.*, push small changes when using incremental builds). As an example, when
412 talking about flaky behavior experienced in the build process, O₄ mentioned that: “*of course, we*
413 *have some retry for network issues*”, while “*in case of resources problems we do not have retries, but the*
414 *pipeline maintainers can open issues aimed at solving the problem*. The outcome of this step consists
415 of 90 relations from 128 sentences. We will present how different codes relate to each other and are
416 spread among different organizations through storytelling.

417 418 **3.3 Member-checking survey to validate relations between challenges/barriers and** 419 **mitigation strategies**

420 To verify our understanding of how the interviewed organizations act to address the challenges
421 and barriers encountered while setting and maintaining the CI/CD pipeline for CPS development,
422 we conducted a member-checking survey by involving the interviewees themselves, or people
423 working in the same team of the interviewees. Asking outside the same team, especially in large
424 organizations, would have reached completely different projects or even different domains, even
425 unrelated to CPS, making the member-checking worthless.

426 The survey has been designed by following guidelines for survey design and operation from
427 social science [31] and software engineering [41–44, 57]. Specifically, the survey contains:

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

435
436 Since the main goal of the survey is to validate our correct understanding of the challenges/barriers
437 and related mitigation strategies, we asked the participants to provide their personal contacts
438 (among them the name of the organization) mainly for traceability purposes.

439 For each section in the survey, we start by asking whether or not the challenge/barrier has
440 been faced at least once by the team they are working with. Specifically, instead of using a yes/no

441

442 question, we added a third option aimed at highlighting those cases where the challenge/barrier
443 cannot be encountered due to the development process adopted by the organization. For instance,
444 if an organization does not use HiL in its development process, it will never experience problems
445 due to the high cost or lack of scalability of the hardware devices. If the challenge/barrier has
446 been encountered at least once within the organization, we list a set of questions, each one aimed
447 at investigating the adoption of the identified mitigation strategy to overcome the previously
448 presented challenge/barrier. Specifically, the respondent could choose between three different
449 options: (i) yes, and we used it, (ii) yes, but we never used it, and (iii) no. Two out of 17 questions
450 dealing with mitigation strategies provide only two options: (i) yes, it happened, and (ii) no, it never
451 happened. At the end of each section, respondents could use an optional free comment field to
452 provide additional mitigation strategies adopted for overcoming the related challenge/barrier.

453 The questionnaire was administrated through Survey Hero¹, and we kept the questionnaire open
454 for 12 weeks. Note that nobody reported having particular issues (e.g., privacy issues) with the
455 used survey administration tool. Furthermore, because of constraints imposed during the survey
456 administration, we had to keep it anonymous.

457 After closing the survey, we obtained eleven responses from the 10 organizations involved in the
458 semi-structured interviews. Specifically, for O₅ we obtained two different responses, even if one of
459 them did not provide demographic information. Among the 10 respondents providing their personal
460 contacts, four of them have also participated in the semi-structured interviews. Furthermore, 4
461 are R&D Managers, 3 are software and hardware integrators, 2 are DevOps Architects, and 1 is
462 a DevOps QA Engineer. In terms of years of experience with CPS development, five respondents
463 have between 1 and 5 years of experience, two respondents between 5 and 10, and the remaining
464 two more than 10 years. Seven out of nine participants (the ones answering this specific question),
465 declare that their organization already has in place a CI/CD pipeline used while developing CPSs (1
466 introduced it less than one year ago, 1 has a mature pipeline introduced more than five years ago,
467 while 5 between one and five years ago), and in terms of the way they interact with the pipeline,
468 among the six participants who answered this question, 1 only uses the CI/CD pipeline, 2 are
469 involved in its setting and maintaining, and 3 set, maintain, and use it for their development tasks.

470

471

3.4 Evaluation through an external survey

472

To address RQ₂ we conducted a survey involving practitioners using (or trying to set up) a CI/CD
473 pipeline for CPS development in their organization. To recruit participants we used two different
474 sources:

475

476

477

478

479

480

481

482

483

484

485

486

487

- (1) *Snowball sampling* [30], i.e., we shared the survey link to some personal contacts and encouraged them to indicate us further participants. This choice has been dictated because, while we had a relatively limited set of contacts reachable with our knowledge, snowballing could help us to reach the relevant people (those involved in CPS development by relying on a CI/CD pipeline).
- (2) *An infrastructure for recruiting survey participants*, namely *Prolific*². This platform allows to reach additional participants, by paying a small fee. The platform has a participant screening facility (we required participants to have at least a bachelor's degree in computer science or similar, and knowledge about relevant software development technology, including versioning, monitoring, virtualization, and testing). Also, similarly to what was done in the member-checking survey, we collected further information about CI/CD competences to further filter participants. At the same time, we are aware that with *Prolific* we have less

488 ¹<https://www.surveymhero.com/>

489 ²<https://www.prolific.co>

490

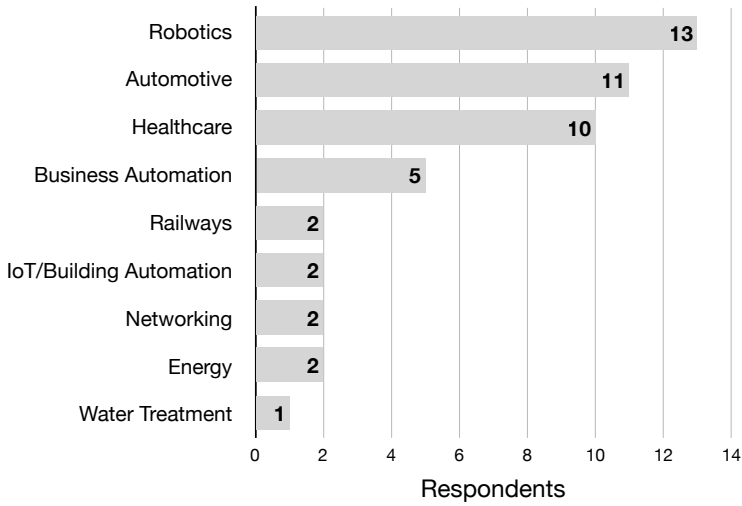


Fig. 2. CPS domains from the external validation survey

control over the participants' reliability than with snowballing. To mitigate this problem, our online package contains separate results belonging to the snowball sub-sample and the *Prolific* sub-sample.

The online survey presented to the participants has: (i) an introduction explaining the study goals, *i.e.*, to assess a catalog of challenges and barriers concerning the setting and maintaining of a Continuous Integration (CI) and Continuous Delivery (CD) pipeline for CPS development; (ii) 14 sections in which we validate challenges, barriers and mitigation strategies; and (iii) a demographic section similar to the one described in Section 3.3.

We started by asking, for each challenge/barrier (properly grouped in categories), whether they have ever encountered it as a factor preventing/limiting the setting up of a CI/CD pipeline, or, if the participant did not encounter it, whether she perceives the challenge/barrier as a real impediment. Specifically, the respondent could choose between four different options: (i) yes, it is relevant (and I encountered it), (ii) yes, it is relevant (but I never encountered it), (iii) no, I do not consider it as relevant, and (iv) does not apply to my context. If at least one of the challenges/barriers in the category was felt as relevant to the respondent, the survey shows a new section asking about the mitigation strategies used (or felt as relevant) to address the previously selected challenges/barriers by using a multiple choice answer. Specifically, the respondent could choose among the mitigation strategies we obtained as a result of RQ₁, but could also add new (unseen) mitigation strategies. Finally, the survey contains an open-ended question aimed at collecting other challenges/barriers that did not apply to the 10 interviewed organizations.

Also in this case, the survey has been administrated through Survey Hero, and nobody reported having particular issues with this administration tool. For the snowball sample, the questionnaire has been left open for one month, and due to constraints imposed during the survey administration, we kept it anonymous. For what concerns *Prolific*, we obtained the requested responses within the same days the survey has been opened.

In the end, we obtained 19 responses from the snowball sampling, and 50 further responses from *Prolific*. However, through a screening of the participants' answers we discarded 14 responses from *Prolific*, *i.e.*, (i) it was difficult to infer whether or not the participant works for CPS development,

540 e.g., education or applications for cosmetic stores, and (ii) the participant declares to not have a
541 CI/CD pipeline in place within the organization, and at the same time declares that the CI/CD
542 pipeline has been adopted only recently. As a result, we obtain a final set of 55 valid responses
543 covering 9 different application domains (as shown in Figure 2).

544 Among the respondents providing demographic information (51), in terms of the role played in
545 their organization, there are: 23 software and hardware integrators, 13 R&D Managers, 7 DevOps
546 Architects, 5 software developers/testers, 1 Project Manager, and 1 CTO (Chief Technology Officer).
547 In terms of years of experience with CPS development, 19 respondents have less than 1 year of
548 experience, 27 between 1 and 5, two respondents between 5 and 10, and the remaining three more
549 than 10 years. 47 out of 51 respondents declare that their organization already has in place a CI/CD
550 pipeline used while developing CPS (19 introduced it less than one year ago, seven have a mature
551 pipeline introduced more than five years ago, while 21 between one and five years ago.) Finally, in
552 terms of the way our respondents interact with the pipeline, 31 only use the CI/CD pipeline, six are
553 involved in its setting and maintaining, and the remaining 10 set, maintain, and use the pipeline for
554 their development tasks. Finally, among the respondents who declare that their organization does
555 not have a CI/CD process in place for CPS development, three declare being involved in setting it.

556 4 STUDY RESULTS

557 In the following, we report and discuss the results addressing the RQs defined in Section 3. To
558 properly contextualize challenges, barriers, and their mitigation strategies, it is important to
559 summarize the development process of the interviewed organizations. Specifically, Section 4.1
560 briefly describes, for each organization participating in the semi-structured interviews, the CPS
561 development process, focusing more on the adoption of CI/CD pipelines and, in general, on their
562 level of build automation. The interested reader could find more details in the Appendix.

563 4.1 Contextualization: Organization Profiles

564 Table 4 provides an overview of the main analyzed dimensions for the 10 organizations considered
565 in our study. In the following we briefly describe them.

566 4.1.1 O_1 (Aerospace). O_1 is involved in verification and validation (V&V) tasks for aerospace
567 software (i.e., on-board software for satellites), hence their CI/CD pipeline is only for V&V and
568 not for development. The standards in the aerospace domain enforce the adoption of conventional
569 programming languages, i.e., “We mainly use ANSI C-99 following the MISRA rules”, as well as the
570 need for certifying software.

571 O_1 started to adopt CI/CD practices for CPSs less than one year ago. Due to the application
572 domain and the related standards and certification constraints, the pipeline compiles the software
573 provided and developed by the customer, relies on SonarQube for static code analysis checks,
574 and executes unit and robustness tests to “check how the system behaves/reacts in the presence of
575 unexpected inputs”. The triggering of the pipeline is almost manual, even if there are scheduled
576 nightly builds for running test suites requiring a long time to complete.

577 Finally, O_1 cannot involve HiL in the pipeline, as it would require a clean room not accessible
578 from the outside. Instead, it relies on third-party simulators provided by the customer, reduc-
579 ing the costs/efforts needed to develop the simulators from scratch, as well as guaranteeing the
580 trustworthiness of the outcome being produced.

581 4.1.2 O_2 (Healthcare). O_2 is a large organization involved in the healthcare domain. It adopts
582 conventional programming languages, i.e., mainly C# and C++ during the development process.

583 O_2 has a CI/CD pipeline in place for CPS development that has been introduced 4 years ago, and
584 they are still improving it. Furthermore, based on its application domain, O_2 is constrained to “follow
585
586
587
588

Table 4. Summary of the CPS development process adopted within the 10 interviewed organizations (*i.e.*, O_{ID}). The ✓ (✗) occurs when the property (does not) apply to the organization; the — represents cases where the property is not applicable/available for the organization; ● means that the phase is automatized within the pipeline; ◐ means that the phase is automatized but not included in the pipeline; ○ means that the phase is done manually.

Property		Organizations									
		O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}
Prog. Language		C	C# C++	C C++	C++ Python	RTJ	C C++	C	C C++	C# Java	Java Python
Pipeline Maturity		< 1	[1, 5)	✗	—	< 1	—	[1, 5)	✗	✗	≥ 5
Phases	Static Analysis	●	●	✗	●	✗	●	○	○	✗	●
	Unit Test	●	●	○	●	●	●	●	○	◐	●
	Int. Test	✗	●	○	●	✗	●	●	○	◐	●
	System Test	✗	●	✗	●	✗	◐	◐	○	○	✗
	Non-Func. Test	●	◐	○	○	◐	○	◐	○	◐	●
	Deploy	✗	●	◐	●	●	●	●	◐	◐	●
Triggering	Continuous		✓	—	✓	✓	✓	✓	—	—	
	Incremental		✓	—					—	—	✓
	Nightly	✓	✓	—	✓	✓			—	—	
Pipeline Config.	Env.	Stable	Domain specific	—	Stable	—	Device specific	Device specific	—	—	Stable
	Staged Builds	✗	✓	—	✗	✗	✗	✓	—	—	✗
Mocking		✗	✗	—	✗	✗	✗	✗	—	—	✓
Simulators		Ext.	Int.	—	Ext.	Int.	Int.	Int.	Int.	Int.	Int.
HiL		✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Containerization	VMs	—	—	—	—	✓	—	✓	—	✗	✗
	Docker	—	—	—	Deploy and HiL	✗	✗	—	—	Deploy	HiL Deploy

medical application frameworks providing a base set of rules in terms of how to build applications and how to integrate them”.

O_2 adopts both incremental and nightly builds. While nightly builds leverage HiL and run three different types of testing, namely unit/component, sub-system, and system testing, incremental builds leverage self-developed simulators to provide developers fast feedback about the impact of their changes, *i.e.*, only a subset of the whole set of functional tests are executed. Furthermore, both incremental and nightly builds run static code analysis tools. Finally, nightly builds imply an automated deployment on a “real” Computed Tomography (CT) scanner, *i.e.*, “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”.

4.1.3 O_3 (Acoustic Sensors). O_3 is involved in CPS innovation for the industry, among others, the development of the SPL Noise Meter Board, by using conventional programming languages, *i.e.*, Python for testing and C, C++ for micro-controllers development. Each team is composed of both software and hardware experts who work together.

O_3 does not have a CI/CD pipeline for CPS development, however, the deployment is fully automated, while the testing is manual, *i.e.*, impossibility to automatically test acoustic signals. Finally, at the moment, O_3 only uses real hardware devices, yet they wish to include simulators in their CI/CD process.

638 4.1.4 *O₄ (Robotics)*. *O₄* is involved in the development of autonomous robots, and similarly to
639 *O₃*, each team accounts for both hardware and software experts. In their development process,
640 *O₄* mainly adopts C++, together with Python for users' interfaces and for interacting with the
641 hardware devices.

642 *O₄* has a fully containerized (using Docker) pipeline for CPS development. It relies on continuous
643 and nightly builds for running regression testing activities on already packaged components and
644 for deployment to the customers. Furthermore, continuous builds also execute static code analysis
645 tools to inform developers about code quality degradation, and unit tests relying on simulators. The
646 application domain does not introduce certification constraints, while it hinders the automation of
647 non-functional testing within the pipeline. Finally, *O₄* relies on third-party simulators and HiL into
648 different stages of the whole CI/CD process.

649 4.1.5 *O₅ (Automotive)*. *O₅* is a large organization operating in the automotive domain working
650 on the software-focused driving platform. This is the only organization in our study relying on
651 real-time languages, *i.e.*, real-time Java to cope with scheduling requirements of embedded systems.

652 *O₅* already has a CI/CD pipeline in place mainly for deployment purposes, even if it is working
653 on improving it. However, unlike the others, *O₅* relies on virtual machines instead of using Docker
654 containers. Moreover, *O₅* does not test all the developed modules together since it *"deploy[s]*
655 *individual bundles to a platform."*

656 Finally, since *O₅* develops software for embedded entertainment in the automotive domain, the
657 HiL is only available for a final validation on the customer's side, so most of the work is done
658 relying on virtual environments.

659 4.1.6 *O₆ (Aerospace)*. *O₆* operates in the aerospace domain, and it is mainly involved in developing
660 and refining the routing algorithm for the Free Route Airspace (FRA). Similarly to *O₁*, it relies on
661 conventional languages: *"C and C++ [are] used for the back-end."*

662 *O₆* already has a CI/CD pipeline including static code analysis, unit testing, integration testing,
663 and deployment. Similarly to *O₁*, it is required that the developed code satisfies strict certification
664 requirements that are mainly checked by relying on code coverage tools. However, differently
665 from other organizations, *O₆* does not rely on nightly builds, meaning that also time-intensive
666 tasks are executed at each change: *"even the slow builds are continuously built."* Finally, the pipeline
667 provides a monitoring mechanism for what concerns aspects of the real-time operating system
668 such as scheduling and memory that *"gives us the possibility to collect feedback/evidence that may*
669 *help us in obtaining the certifications."*

670 As regards HiL and simulators, *O₆* relies on both, however it does *"not have simulators and HiL*
671 *in the same pipeline mostly for certification issues."*

672 4.1.7 *O₇ (Railways)*. *O₇* is involved in delivering software for railways, *i.e.*, Train Control Manage-
673 ment System (TCMS). In terms of programming languages being used, the interviewee mentions
674 the need of adapting the programming language to the device on which the software has to be
675 executed.

676 *O₇* already has a CI/CD pipeline in place for CPS development that, at the moment, is in a
677 continuous improvement state. Based on the application domain, *O₇* adopts staged builds following
678 the "green-build rule". In the first stage, the build process is executed on a virtual machine, and in
679 the presence of a green status, all the components are deployed together, enabling the execution on
680 the virtual train. In the presence of a green status, it is possible to move to the next stage that relies
681 on the hardware test track, *"where [there is] the whole set of devices and even some more that [are*
682 *not] in the virtual train."* Finally, in the presence of a green status it is possible to run the last stage
683 relying on a real train. All the stages include functional testing, while the deployment is automated
684
685
686

only for the first stage. Based on the previous statements, it is possible to conclude that O_7 adopts both simulators and HiL in different stages of the build process, with the use of HiL occurring only in the last stage of the pipeline.

4.1.8 O_8 (Railways). O_8 is involved in the railways domain, *i.e.*, the development of a specific component used for transmitting data between on-board and ground applications. O_8 uses C and C++ (*i.e.*, conventional programming languages), and it has strict certification requirements, *e.g.*, compliance with the railway standards and specifications.

O_8 does not have a CI/CD pipeline in place for CPS development and it has, in general, little automation in the development process, *i.e.*, only the adherence to standards and specifications is automated. Finally, due to the high cost of the hardware devices in this particular domain, O_8 mainly relies on simulators that are self-developed. However, once per week, O_8 performs a testing session with a real “train running in a real environment with real traffic”.

4.1.9 O_9 (Identification Technology). O_9 is involved in “develop[ing] software relying on identification technologies such as RFID [(Radio Frequency IDentification),] Bluetooth low energy or bar codes”, relying on conventional programming languages such as Java and C#.

Due to a lack of culture for setting a pipeline dealing with sensors and actuators, O_9 does not have a CI/CD pipeline for CPS development. However, the testing phases are almost fully automated. For what concerns the deployment of CPS-related software, O_9 relies on Docker for creating images that are manually deployed onto the servers. The development process also features a monitoring component for the internal development platform and customers’ devices, to notify about anomalies and errors, as soon as they occur. Finally, the development process considers both (self-developed) simulators and HiL.

4.1.10 O_{10} (Energy). O_{10} is involved in the development of prototypes and proof of concepts for the energy domain. It has a mature (*i.e.*, introduced in 2016) pipeline for CPS development that uses conventional programming languages, mostly Java and Python.

Other than having a compilation phase, the CI/CD pipeline is aimed at executing static code analysis tools and linters, unit, and integration tests, followed by a deployment phase where the packaged version of the software is usually stored into an artifact repository as a docker image. O_{10} does not rely on nightly builds, while it only uses incremental builds.

O_{10} does not need to run the software on embedded devices, implying that O_{10} , other than simulating the hardware when needed, mainly replaces it with mock-ups. Only when the real devices are available and it is safe to use them for testing, O_{10} uses Docker images for checking the correct behavior over the real devices.

4.2 RQ₁: What are the challenges and barriers respondents encounter, and how do developers deal with them?

This research question describes barriers and challenges emerging from the semi-structured interviews. We start by describing the challenges related to the CPS development process in general. Then, we describe barriers and challenges encountered when setting and maintaining the CI/CD pipeline for CPS development, together with the related mitigation strategies. Note that we did not find mitigation strategies for all the barriers and pipeline-related challenges and, as described in Section 3.3, the member-checking survey only considers the barriers/challenges for which there was an explicit mitigation strategy reported by at least one of the interviewed organizations.

4.2.1 Process-related challenges. Table 5 reports the process-related challenges identified in our interviews, together with the traceability among which challenge has been encountered by which organization. It is important to remark that process-related challenges may not be specific to CI/CD,

Table 5. Process-related challenges

Category	ID	Challenge	Organizations
General	PRC ₁	Cycle-time reduction	O ₂
	PRC ₂	Onboard developers	O ₇
Culture	PRC ₃	Limited CI/CD culture	O ₁
	PRC ₄	Limited CI/CD culture for CPS development	O ₉
Environment	PRC ₅	Complexity of the environment	O ₂ , O ₄ , O ₅ , O ₆ , O ₈
	PRC ₆	Variability of the environment	O ₇
	PRC ₇	Lack of redundancy in the environment	O ₇
Testing	PRC ₈	Test cases manually derived	O ₅ , O ₈
	PRC ₉	Test cases manually executed	O ₃ , O ₉
	PRC ₁₀	Different interpretations for the same requirements	O ₇
	PRC ₁₁	Need a controlled environment for test automation	O ₃ , O ₄
	PRC ₁₂	Complexity in oracle specification for test automation	O ₃ , O ₅ , O ₆ , O ₈ , O ₉
	PRC ₁₃	Complexity for deriving integration tests	O ₁₀
Deployment	PRC ₁₄	Complexity for deriving safety tests	O ₄ , O ₅ , O ₈
	PRC ₁₅	Late deployment	O ₂
Simulators	PRC ₁₆	Expensive deployment	O ₇
	PRC ₁₇	Lack of trustworthiness for simulators	O ₃
	PRC ₁₈	Complexity for oracle automation with simulators	O ₈

but are, more in general, challenges in the development process that, based on what was reported by the interview participants, have an impact on setting up and maintaining a CI/CD pipeline.

The challenges have been grouped into six different categories, *i.e.*, general, culture, environment, testing, deployment, and simulators. For each category, in the following, we provide 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 organizations. One of the main benefits of adopting a CI/CD pipeline is related to the overall cycle time reduction (PRC₁). However, even if O₂ has already invested effort and money in reducing the 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.”* While this challenge also applies to conventional software, when it comes to the CPS context, the challenge is exacerbated mainly due to the need of interacting with both HiL and simulators. In this regard, O₂ mentioned that the cycle time cannot be easily reduced due to (i) the high costs for the infrastructure, and (ii) the translation of test strategies to hardware devices being very demanding”.

O₇ is facing problems when trying to onboard new developers (PRC₂) mainly due to the complexity of the railways’ domain, as also found by Törngren *et al.* [74]. The interviewee stressed that in the railways’ domain it is crucial to follow specific standards that need to be known and properly understood by developers and testers.

Culture. This category groups two challenges related to the presence of a limited CI/CD culture in the development teams. This may limit the possibility of properly leveraging CI/CD facilities throughout the development process. O₁ reports the adoption of a pipeline that only includes tasks that are easy to automate mainly due to *“lack of knowledge”* (PRC₃), as also found by Zampetti *et al.* [85]. Instead, while O₉ 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”*), 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 interaction between software and hardware components (PRC₄). Specifically, there is a need for knowledge on how to properly account for the

785 inclusion and setting of both HiL and simulators in the CI/CD pipeline configuration, as well as
786 how to include a feedback mechanism to gather information directly from the field.

787 **Environment.** This category features three different challenges dealing with the characteristics
788 of the physical environment in which the developed code has to be deployed.

789 Among them, only PRC₅, *i.e.*, environment complexity, is mentioned by multiple organizations
790 (five out of ten), while the remaining two only come from O₇. The complexity of the environment
791 impacts the execution environment being set (*i.e.*, simulators or HiL). The unavailability of third-
792 party simulators (and the need for self-developing them) impacts the ability to simulate certain
793 behaviors, or even in deviations between HiL and simulated environments. The consequence is
794 that builds executed on simulators will have a different outcome when run on HiL. For instance, O₄
795 mentioned: *“Walking is not so easy to simulate so we need a real walking robot for spotting bugs”*,
796 while O₈ stated: *“It could be difficult, demanding and expensive to have a one-to-one relationship*
797 *between simulators and real systems”*. Our findings stress what is already known from previous
798 literature in terms of relying on simulated environments, *i.e.*, the testing over simulators may fail
799 to expose problems that would only manifest when running the system on the real hardware [52].

800 O₇ faces a problem related to the high environment variability (PRC₆) [74], due to trains hav-
801 ing different characteristics: *“We can rarely copy-paste software that has to run on different train*
802 *architectures.”* At the same time, O₇ also faces a challenge due to the structure of its development
803 process that is not cloud-based and has no redundancy (PRC₇), implying that *“in the presence of*
804 *network issues or server issues we are totally black and this is affecting everyone.”*

805 **Testing.** This category groups seven challenges. O₅ and O₈ mention as a challenge the substantial
806 manual effort required for the test case specification process (PRC₈). O₃ and O₉, instead, felt the
807 manual execution of testing activities to be challenging, *i.e.*, PRC₉ (*e.g.*, *“Another big barrier is related*
808 *to the test case execution that, at the moment, we are doing manually since both the environment*
809 *setting and the oracle definition require manual intervention”* for O₉). Our findings confirm what is
810 already pointed out by Mårtensson *et al.* [52] in terms of the presence of complex user scenarios
811 implying the need of manual testing.

812 O₇ found it difficult to automate the test case specification mainly because the standards might
813 be interpreted differently by different developers, and both might be correct (PRC₁₀) — *“how do you*
814 *read the standard? The standard is interpreted so the same requirement can be differently interpreted*
815 *by different people (a challenge for automation).”* A different challenge experienced by O₃ and O₄ is
816 related to the need for a controlled test environment (PRC₁₁) impacting the execution environment
817 to be used in the pipeline. For instance, O₃ mentioned: *“Since the output of the system is sound and*
818 *the test should check the sound quality it is better to have it in a controlled environment that makes*
819 *use of simulation.”*

820 Another test automation challenge is related to oracle specification (PRC₁₂), as mentioned by five
821 out of 10 organizations. The impossibility of specifying an automated oracle hinders what kinds
822 of tests one can run in the pipeline. This may happen, for instance, when one needs to evaluate a
823 signal received from a sensor, *i.e.*, *“The main challenges for automatizing the test execution: a good*
824 *way to model the test itself and have an oracle that can compare with the actual behavior.”* stated
825 by O₃. This aspect has already been mentioned by Mårtensson *et al.* [52], however, while they
826 only talked about usability testing, we stress more the impediment in automatically determining
827 and checking the test oracles, also for functional testing mainly due to outcome coming from real
828 hardware devices working in a real environment with many external factors to control for, *e.g.*, to
829 check the quality of the acoustic signal coming from sensors (O₃).

830 The remaining two challenges are related to difficulties encountered when specifying/deriving
831 integration (PRC₁₃) and safety (PRC₁₄) tests. As regards the former, O₁₀ develops prototypes
832 requiring the interconnection of many different sub-components. This makes it difficult to determine
833

Table 6. Pipeline-related barriers

Category	ID	Barrier	Organizations
Resources	B ₁	Limited human resources	O ₈ , O ₉
	B ₂	Limited availability of software and/or hardware resources	O ₁ , O ₂ , O ₃ , O ₄ , O ₅ , O ₆ , O ₇ , O ₈ , O ₉ , O ₁₀
Domain	B ₃	Complex non-functional requirements	O ₆
	B ₄	Security configuration prevents CD	O ₂
	B ₅	HiL not usable, e.g., for safety or security reasons	O ₁ , O ₂ , O ₈ , O ₉ , O ₁₀

the expected system behavior: “It is quite hard to derive integration test cases due to the complex combination of all different parts.” As regards the specification of safety tests, in agreement to what indicated by Gautham *et al.* [27], O₄, O₅, and O₈ pointed out the complexity to identify situations “that could never happen.” or “that you do not expect to happen.” Checking for safety requirements is highly important, especially in those domains, such as aerospace and railways, where the safety integrity level (SIL) of the system must be equal to or higher than three.

Deployment. This category features two challenges occurring when deploying software on the customers’ side. Having deployment too late in the development process (PRC₁₅) may result in installation issues (PC₉ in Table 7), as experienced by O₂: “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 behavior is not being considered in the early stages of development.”

Then, there are cases where the deployment is expensive (PRC₁₆) in terms of time and effort needed to complete it. This impacts both the type of execution environment adopted within the pipeline, as well as the build triggering strategy. As experienced by O₈ in the railways’ domain, the deployment on a 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.”

The late and expensive deployment is strictly related to the CPS nature. Indeed, as already highlighted in Section 4.1, the organizations deploy on real hardware devices only during the last stages of the overall CI/CD process, mainly due to the high costs of the hardware in specific domains such as railways and aerospace.

Simulators. The last category, among the process-related challenges, deals with the usage of simulators. O₃ 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 behavior of the system in a real environment (PRC₁₇). Finally, as reported by O₈, some scenarios cannot rely on simulators. Specifically, if it is complex for a human to specify the expected behavior for some scenarios, of course, it is not possible to rely on simulators that can emulate the same behavior (PRC₁₈).

4.2.2 Barriers for CI/CD pipeline setting and maintaining and related mitigation. Table 6 summarizes the five barriers encountered by the ten organizations when applying CI/CD to CPSs. These barriers have been grouped into two categories, described in the following.

Resources. This category groups the barriers dealing with limited availability of human (B₁) and software and/or hardware resources (B₂), both influencing the type of execution environment adopted within the pipeline. While we are aware that those barriers can also apply to conventional software systems, the barriers worsen for CPS development, where it is mandatory (i) to rely on simulators, mostly self-developed where you need high expertise about the domain, and (ii) to use HiL that is very expensive in particular CPS domain such as railways and aerospace. For instance, O₈ mostly relies on HiL due to limited availability of human resources having the skills needed to develop/configure simulators — “given the needs and the budget of our company, it’s much better for

883 *more complex scenarios to rely on the hardware in the loop and only use simulations when whatever*
884 *needs to be simulated is very simple.”*

885 All the interviewed organizations reported the limited availability of software and hardware
886 resources. Specifically, O₆ mentioned: “*Based on the fact that in the avionics domain the cost of the*
887 *hardware is very expensive, we do most of the work in simulated environments*”, while O₇ stated that
888 “*Resources for the hardware devices (hardware test tracks and testbeds as real trains) represent an issue*
889 *for us. We have a limited number of test tracks.*”

890 As reported in Table 8, the analysis of the interviews’ transcripts (see Table 8) has elicited two
891 mitigation strategies: (i) prioritize and select the test cases to be included within the pipeline (*i.e.*,
892 “*Some strategies rely on genetic algorithms to optimize the resources available for the testing execution*
893 *environment*” from O₁), and (ii) adopt incremental builds mainly relying on impact analysis, as
894 reported by O₂ – “*for what concerns rolling builds we try to limit the amount of testing being executed*
895 *in them to be as fast as possible.*” The member-checking survey confirms the previous findings, and,
896 as shown in Table 8, six out of ten organizations (O₁, O₂, O₅, O₆, O₇, O₁₀) report to rely on test
897 prioritization, while O₃, O₄, O₈, and O₉ consider it useful while having never used it. As regards
898 the adoption of incremental builds, instead, O₁, O₂, O₄, O₇ and O₉ mention its adoption, while O₈
899 considers it a useful approach to deal with limited hardware/software resources.

900 Alternative solutions reported in the member-checking survey to cope with limited availability
901 of resources are “*architectural changes with improved testing concepts*” (O₇), and, unsurprisingly,
902 “*platform virtualization*” (O₅).

903 **Domain.** This category includes three different barriers, two of them highlighted by only one
904 organization. Specifically, B₃ and B₄ are related to difficulties arising when automating certain
905 phases in the CI/CD pipeline. For instance, O₆ had to cope with the use of a real-time operating
906 system which made task automation difficult, *i.e.*, “*the complexity of integrating within the pipeline*
907 *the execution of nonfunctional testing and system testing*”, while O₂ could not implement automated
908 deployment due to security policies for the healthcare domain: “*We cannot deploy at the moment*
909 *because a change in the security configuration of the software prevented our standard [deployment]*
910 *process.*”

911 B₅ is related to coping with a complex execution environment. Specifically, O₁₀ mentions that they
912 could not integrate HiL in the CI/CD pipeline for safety reasons, and adopts simulation/mocking
913 for the hardware devices to overcome it. As shown in Table 8, all the organizations facing this
914 barrier used the same mitigation strategy to deal with it. Furthermore, O₂ mentions the possibility
915 to rely on “*digital twin hardware that avoids the safety issues (no moving parts, no radiation) but*
916 *simulates the hardware to some much better.*”

917
918 **4.2.3 Pipeline-related challenges and related mitigation.** Table 7 summarizes the pipeline-related
919 challenges faced by the 10 organizations. The challenges have been grouped into five categories,
920 each one related to a specific aspect of the CI/CD pipeline setting and evolution, *i.e.*, pipeline
921 properties, thoroughness, simulators, HiL, and flaky behavior. In the following, we discuss each
922 identified challenge, together with some examples from the study participants’ experiences, and
923 related mitigation strategies.

924 **Pipeline Properties.** This category accounts for six different challenges, two of which deal with
925 the build execution time (PC₁ and PC₂), while the remaining four are related to the overall pipeline
926 configuration. Four out of 10 organizations faced long build execution time, influencing the type of
927 tasks automatized within the pipeline. For example, O₆ mentioned: “*Slow builds hinder the inclusion*
928 *of running non-functional testing in the pipeline.*” While this is also considered a relevant challenge
929 for conventional applications [14, 77, 85], for CPSs the problem can be further exacerbated when
930 deploying and executing software on simulators or HiL. The latter confirms what is already found
931

Table 7. Pipeline-related challenges

Category	ID	Challenge	Organizations
Pipeline Properties	PC ₁	Long build execution time	O ₁ , O ₂ , O ₄ , O ₅ , O ₆ , O ₇ , O ₈
	PC ₂	Build time estimation	O ₉
	PC ₃	Static code analysis tools configuration	O ₃ , O ₇
	PC ₄	Lack to access the production code from the pipeline	O ₁
	PC ₅	CI/CD configuration highly coupled with the environment	O ₂ , O ₅
	PC ₆	Reusability of build artifacts	O ₂
Thoroughness	PC ₇	Development environment detached from the execution environment	O ₁
	PC ₈	Detecting deployment-related errors	O ₂ , O ₆
	PC ₉	Continuous installation	O ₂ , O ₃ , O ₄ , O ₅ , O ₇ , O ₈
	PC ₁₀	Closing the loop introduces performance degradation	O ₅
	PC ₁₁	Complexity in closing the loop due to uncontrollable factors	O ₄ , O ₉
	PC ₁₂	Complexity in closing the loop due to data collection from the field	O ₅
Simulators	PC ₁₃	Limited in their functionality	O ₁ , O ₂ , O ₄ , O ₅ , O ₇ , O ₈ , O ₉ , O ₁₀
	PC ₁₄	Functional correctness	O ₅ , O ₆ , O ₇ , O ₁₀
	PC ₁₅	Deal with real-time properties	O ₅ , O ₉
	PC ₁₆	Interaction with the environment	O ₂ , O ₃ , O ₄ , O ₅ , O ₆ , O ₇ , O ₈ , O ₉
	PC ₁₇	Accessibility	O ₁ , O ₅ , O ₇ , O ₉ , O ₁₀
HiL	PC ₁₈	Availability	O ₁₀
	PC ₁₉	Automated deployment on HiL	O ₇ , O ₈ , O ₉
	PC ₂₀	Test Automation on HiL	O ₂ , O ₄ , O ₆ , O ₇ , O ₉
	PC ₂₁	Costs and scalability	O ₁ , O ₂ , O ₃ , O ₄ , O ₅ , O ₇ , O ₈ , O ₉
	PC ₂₂	Dependency installation	O ₄
Flaky Behavior	PC ₂₃	Features' interaction	O ₂
	PC ₂₄	HiL availability	O ₁₀
	PC ₂₅	HiL inputs	O ₅ , O ₁₀
	PC ₂₆	Lack of control over resources	O ₂ , O ₄ , O ₅ , O ₆ , O ₇ , O ₉ , O ₁₀
	PC ₂₇	Network issues	O ₁ , O ₂ , O ₄ , O ₅ , O ₆ , O ₇ , O ₉ , O ₁₀
	PC ₂₈	Timing issues	O ₄ , O ₁₀

by Mårtensson *et al.* [52] highlighting how working with a highly integrated (tightly coupled) system, a small delivery to the main track may cause building and linking of a large part of the system resulting in long build times. The latter has been also mentioned by O₂ where there is a single integration branch where the components developed by their 70 teams are integrated into a single join point: *i.e.*, “each component has a test service so running unit tests is very fast but we have a huge amount of high-level testing that is easy to write but kills us in terms of execution time”. By looking at the result of the survey (see Table 8), the interviewed organizations mentioned a wide set of actions to deal with the above challenge. One possibility is to prioritize and select only a subset of test cases in the test suite to be executed (used also by O₁, O₂, and O₇, and considered a useful action by O₄ and O₈). A different approach, highlighted by O₂, 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 O₄, O₅, O₆, and O₇, while O₈ only felt it as useful. It is also possible to run the whole build process only within nightly builds, even if this may be controversial since it defeats the CI/CD purpose [13]. However, this is considered acceptable for O₁, as its pipeline is limited in scope, *i.e.*, used only for V&V purposes. Also O₂, O₅, O₆, and O₇ rely on nightly builds to execute time-intensive tasks, while adopting incremental builds during working hours (O₂, O₅, and O₁₀). The latter is also used by O₇ and O₈, while O₄ and O₆ consider the mitigation useful even if they have never adopted it.

A different challenge, experienced by O₉, that can also apply to conventional systems, while it is more critical for CPSS, is related to the build time variability (PC₂), due to the adopted infrastructure “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 to the overall pipeline configuration, in the absence of clear coding standards or guidelines, the adoption of code style checking tools becomes problematic, if not unfeasible (PC₃). In

Table 8. Relations between challenges/barriers and mitigation strategies as seen from the semi-structured interviews and the member-checking survey. In bold there are the organizations that do not rely on the mitigation while considering it a useful solution.

Challenge/Barrier	Mitigation	Organizations
B ₂ : Limited hw/sw resources	Test Prioritization	O ₁ , O ₂ , O₃ , O₄ , O ₅ , O ₆ , O ₇ , O₈ , O₉ , O ₁₀
	Incremental Builds	O ₁ , O ₂ , O ₄ , O ₇ , O₈ , O ₉
B ₅ : Domain hinders HiL	Rely on sim./mock-up	O ₁ , O ₂ , O ₈ , O ₉ , O₁₀
PC ₁ : Long build	Test Prioritization	O ₁ , O ₂ , O₄ , O ₇ , O₈
	Adopt Parallelization	O ₂ , O ₄ , O ₅ , O ₆ , O ₇ , O₈
	Nightly Builds	O ₁ , O ₂ , O ₄ , O ₅ , O ₆ , O ₇ , O ₈
	Incremental Builds	O ₂ , O₄ , O ₅ , O₆ , O ₇ , O ₈
PC ₉ : Continuous installation	Containerization	O₃ , O ₄ , O ₅ , O₇ , O₈
PC ₁₃ : Sim. limited func.	Combine sim. and HiL	O ₄ , O ₇ , O ₈ , O ₉ , O ₁₀
PC ₁₆ : Sim. coupled with env.	Combine sim. and HiL	O ₂ , O ₃ , O ₄ , O ₈ , O ₉
PC ₁₇ : Sim. accessibility	Timeout	O ₁ , O ₅ , O₇ , O₁₀
PC ₂₁ : HiL costs and scalability	Combine sim. and HiL	O₁ , O ₂ , O₃ , O ₄ , O₅ , O ₇ , O₈ , O ₉
	Green-build rule	O ₂ , O ₃ , O ₄ , O ₇
PC ₂₆ : No resources' control	Fix the code	O ₄ , O ₆ , O ₇ , O ₉ , O ₁₀
	Fix pipeline config.	O ₆ , O ₇ , O ₉
PC ₂₇ : Network issues	Retry	O ₂ , O ₄ , O ₅ , O ₆ , O ₇ , O ₁₀

this scenario, approaches for coding style inference may be desirable [61, 83]. Similar considerations apply to bug-finding tools, sometimes inapplicable to CPSs for automating code review, as experienced by O₇: “*we need expertise on the developers' side for determining whether or not a train is behaving in the expected way.*” The latter is strictly related to PRC₂ where, in the presence of safety-critical systems, like the ones in the aerospace and railways domains, it is very difficult to find skilled experts in the domain from both the hardware and software viewpoints.

The lack of access to production code (as experienced by O₁) limits the ability to properly set static analysis or testing tools (PC₄) – “*One big challenge is that we need to guarantee the protection of the source code: How to test a component without having its production code?*” The latter is a specialization of the restricted access to information due to security aspects impediment found by Mårtensson *et al.* [52]. On the same line, there is a challenge (PC₅) related to the extent to which technology restrictions, or restrictions coming from the application domain, may impact the pipeline setting. For instance, O₂ mentioned that “*the Windows situation does not help us with dockerization*”, and at the same time, they are having trouble in properly configuring the CI/CD pipeline for CPS since “*[they] need 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 results in the last challenge related to the impossibility to reuse previously built artifacts (PC₆) in the integration branch (*i.e.*, O₂ mentioned: “*It's a huge pain that we do not reuse artifacts*”), mainly due to constraints imposed by the domain.

1030 **Thoroughness.** This category groups six challenges related to (i) ensuring the overall accuracy
 1031 and completeness of the CI/CD pipeline (PC₇, PC₈, PC₉) scattered across eight organizations, and (ii)
 1032 closing the DevOps loop by gathering data from the hardware, *i.e.*, PC₁₀, PC₁₁ and PC₁₂ experienced
 1033 by three out of 10 organizations.

1034 O₁ faces a challenge related to having a development environment detached from the execution
 1035 environment (PC₇). Another challenge (PC₈ experienced by O₂ and O₆) occurs in the presence
 1036 of incremental deployment, which makes it difficult to detect and isolate deployment errors.
 1037 Furthermore, O₆ reported how this even makes it necessary to reconfigure the entire pipeline — “*you*
 1038 *deploy blocks, if there is an error in one of the blocks detecting it and reconfigure and reset the pipeline*
 1039 *is a problem.*” Finally, continuous installation (PC₉) cannot be achieved due to the late deployment
 1040 strategy (PC₁₇). This is because changes to the environment impact the pipeline configuration,
 1041 which needs to be adapted every time. For what concerns continuous installation problems, O₂,
 1042 O₃, O₄, O₅, O₇ and O₈ have encountered them, with O₄ pointing out that using containerization
 1043 it is possible to facilitate the switching between software versions to deploy, meaning that it will
 1044 be possible to handle the variability of the environment in terms of dependencies. As shown in
 1045 Table 8, containerization is also used by O₅, while O₃, O₇ and O₈ consider it a viable solution.

1046 Moving on to the need for closing the DevOps loop, the interviews indicated three different
 1047 challenges hindering the acquisition of data from the physical environment (or hardware device).
 1048 Working in a CPS context implies having a tight interaction with multiple hardware devices, *i.e.*,
 1049 sensors and actuators, in which gathering data from them could be problematic due to the presence
 1050 of many external environmental factors that must be taken into account, as well as the need
 1051 for having invasive measurement instruments directly in the field. Specifically, O₅ stressed the
 1052 introduction of performance degradation (PC₁₀) due to invasive measurement instruments: “*The*
 1053 *challenge is that monitoring becomes invasive with respect to the system performance.*”, as well as
 1054 the presence of noise in the collected data (PC₁₂): “*There are architectural ways to deal with that*
 1055 *so that if some sensor does not update on time, you still can make a relatively informed decision. But*
 1056 *even then, you have to make sure that the drift is not over a certain size because then you cannot*
 1057 *make reasonable decisions anymore.*” O₄ and O₉ highlighted the presence of uncontrollable factors
 1058 in a CPS execution environment, making it challenging to close the DevOps loop. For instance, O₄
 1059 reported: “*Differently from other software applications, there is data that we cannot control such as*
 1060 *the presence of something on the floor that the robot is not able to perceive so it will fail. You have to*
 1061 *analyze the video data and this is very hard.*”

1062 **Simulators.** This category groups five challenges related to simulators’ issues and limitations
 1063 stressed more in the CPS domain due to the high environment complexity [74], which very often
 1064 results in having scenarios that cannot be emulated, such as in the presence of many external
 1065 environmental factors to be controlled. Specifically, the need to develop them in-house or the lack
 1066 of specific skills may lead to simulators that are limited in their functionality (PC₁₃). For instance,
 1067 O₈ stated: “*we prefer to spend time in testing on real hardware instead of spending time in developing*
 1068 *complex simulators*”, while O₄ reported: “*Walking is not so easy to simulate, so we need a real walking*
 1069 *robot for spotting bugs.*” As shown in Table 8, it is a common habit to adopt a pipeline that relies on
 1070 both simulators and HiL in different build stages to overcome the above challenge. A clear example
 1071 of this happens in O₇, where there is a build process made up of three different build stages, each
 1072 one adopting a specific execution environment (see Section 4.2).

1073 A lack of knowledge about the device/system to simulate can lead to wrong assumptions, affecting
 1074 the simulator’s correctness (PC₁₄) as experienced within O₁₀: “*This happens more at the beginning*
 1075 *of a project when you are not too familiar with the device and you make assumptions on how it works.*”
 1076 These problems might have an impact on the whole CI/CD pipeline setting and trustworthiness,
 1077
 1078

1079 because it is possible to have deviations of the monitored system behavior between the real hardware
1080 and simulators.

1081 As experienced by O₅, the limited capability to simulate real-time properties (PC₁₅) hinders the
1082 applicability of simulators or at least raises the need for further tests on HiL. The latter is also
1083 confirmed by O₉: *“for what concerns the simulation for the RFID we think that the simulation will not*
1084 *give us any benefits due to their unpredictable behavior.”*

1085 Likewise PC₁₃, the high level of interaction between different components (PC₁₆) forces orga-
1086 nizations to directly test feature interaction by using real devices, instead of simulating them.
1087 Indeed, when using simulators for CPSs it is important to remark that they have to interact with a
1088 too complex environment that must be simulated as well. As an example, O₆ mentions problems
1089 faced when simulating a car behavior *“for the CAN data, what do you want to wish to happen here?*
1090 *If you are driving around something you need to know how fast the wheels are turning, as well as*
1091 *what the engine revolutions are together with other sensitive data you might pick up over the canvas.*
1092 *There are a lot of details that are very application dependent.”* Also in this case, as shown in Table 8,
1093 organizations rely on pipeline configurations including different execution environments, *i.e.*, five
1094 out of eight organizations facing the challenge declare that this is a useful mitigation strategy (O₂,
1095 O₃, O₄, O₈, O₉).

1096 If an organization has to test third-party software, as in the case of O₁, there may be the need
1097 to run the simulated environment on a remote machine which may turn out problematic to be
1098 properly integrated into a local pipeline (PC₁₇), due to network security restrictions. Such a scenario
1099 typically occurs in the development of safety-critical systems (which very often are CPSs), because
1100 the software needs to be tested by somebody different from the development organization. To deal
1101 with this problem, O₁ mentions the usage of “timeout” within the pipeline. As shown in Table 8,
1102 O₁ and O₅ handle external simulator unavailability through timeouts, while O₇ and O₁₀ consider
1103 this useful yet they do not use it. O₁ also mentions they often *“request some customization at the*
1104 *customer side of their simulators. Sometimes it is accepted, most of the times not.”*

1105 **HiL.** This category groups four challenges related to issues and limitations of using HiL in the
1106 CI/CD pipeline. As shown in Table 7, three out of four challenges in this category are experienced
1107 by multiple organizations, while PC₁₈ is organization-dependent. Specifically, O₁₀ faces problems
1108 with checking hardware availability before running tests (PC₁₈): *“One of the biggest problems, when*
1109 *any particular hardware is involved, is that the hardware may either not be available, or it may be*
1110 *switched off”.*

1111 From a different perspective, as experienced by O₇, O₈, and O₉, deployment on HiL may be
1112 challenging (PC₁₉). Specifically in O₈ *“remote installation cannot be used with real systems”*, while
1113 in O₉ *“The other challenge is related to having a fully automated deployment over the customers’*
1114 *server in which it is possible to have full control on what is going on and try to identify, as soon as*
1115 *possible, failures/errors occurring during the deployment.”*

1116 Testing on HiL (PC₂₀) is considered very demanding to achieve. O₂ reports: *“If you translate test*
1117 *strategies to the hardware it is very demanding.”* and this is mostly a consequence of limited human
1118 resources being available. However, there are cases where testing on HiL is constrained by the high
1119 cost and lack of scalability (PC₂₁) of the hardware devices/systems *“This costs and does not scale”*
1120 *for O₂, or “it is very costly to test on trains” for O₇.*

1121 As shown in Table 8, the study participants identified two possible strategies to deal with these
1122 cost and scalability problems: (i) relying on a mixed pipeline where continuous builds run on
1123 simulators and some periodic builds on HiL (used by O₂, O₄, and O₉, and considered useful by O₁,
1124 O₃, O₅, and O₈), or (ii) adopting the green build rule when transitioning between simulators and
1125 HiL [15], as highlighted by O₇: *“Only when the tests in the virtual train are green can we move to*
1126
1127

1128 *the next step.*”, and also used by O₂, O₃ and O₄. The alternative would be, as pointed out by the O₅
 1129 survey respondent, *“working with virtual devices instead of real hardware devices.”*

1130 **Flaky behavior.** This category accounts for seven different root causes that may lead to non-
 1131 determinism in the build execution used for CPS development. Flakiness related to non-determinism
 1132 during test execution [89] has been largely studied [16, 46, 48, 58, 90] and approaches to detect
 1133 and cope with it have been proposed [47, 49, 59, 65, 87]. While similar to conventional software,
 1134 dependency installation within the pipeline (PC₂₂) may result in pipelines having a flaky behavior,
 1135 e.g., for O₄ *“ROS uses GitHub repositories for dependency resolution so when GitHub or the repositories*
 1136 *are down our build jobs will fail due to the impossibility of resolving dependencies”*, or else little
 1137 control over external resources (PC₂₆), e.g., *“the most important root cause we experienced is related*
 1138 *to the load on the server-side”*, the root causes behind flaky behavior in CPSs may be different from
 1139 conventional software. Specifically, a CI/CD pipeline for CPSs can suffer from flakiness due to:

- 1140
- 1141
- 1142 • The complex interacting environment (PC₂₃), i.e., CPSs are systems of systems with tight
 1143 interactions among different components, e.g., for O₂ *“the complexity of [the] subsystems*
 1144 *whose features interact across many indirections may lead to non-deterministic behaviors”*;
- 1145 • HiL unavailability (PC₂₄), where without a proper check of the availability of hardware,
 1146 the build outcome might fail intermittently since the pipeline was not able to properly
 1147 communicate with the device, i.e., O₁₀ reported: *“We experienced flakiness in terms of non-*
 1148 *deterministic behavior mainly due to hardware not being available”*. In this specific scenario, it
 1149 is important to properly discriminate between intermittent failures caused by communication
 1150 issues with the HiL from failures due to wrongly implemented functionality;
- 1151 • Presence of noise in the measurements (PC₂₅) when using HiL i.e., difficulty in removing the
 1152 effect of external environmental factors from the data read from the sensors, as experienced
 1153 by O₅ and O₁₀. Specifically, for O₁₀ *“Other times the charge level that you read out would go a*
 1154 *little bit higher or there is noise in the measurements”*, while for O₅ *“you need to understand*
 1155 *what your sensors are sensing and what the acceptable range of inputs are”*;
- 1156 • Network issues (PC₂₇) where, for instance, glitches in the network lead to a connections
 1157 being lost as reported by O₁₀, stressed more in the CPS domain where you need to control
 1158 among the communication occurring across a huge number of different hardware devices
 1159 operating in a complex environment;
- 1160 • Simulators not coping with timing issues (PC₂₈), e.g., O₁₀ stated: *“the last problem is related to*
 1161 *multi-threaded programming”*.
- 1162
- 1163

1164 For what concerns flakiness mitigation, as highlighted in Table 8, when the problem is related to
 1165 the lack of control over resources (PC₂₆), the solutions adopted are (i) to change and fix the pipeline
 1166 configuration, i.e., O₇ stated: *“The misbehavior is reported back to the integration team responsible for*
 1167 *the Jenkins configuration to find a solution.”*), as well as (ii) to fix the root cause of the flaky behavior
 1168 within the code: *“to not experience it anymore in the system”* from O₂. When the root cause of the
 1169 flaky behavior is in the networking (PC₂₇), the organizations leverage the “usual” retries (O₂, O₄,
 1170 O₅, O₇, O₁₀): e.g., *“of course we have some retry for network issues”* for O₄, or *“For what concerns*
 1171 *flaky connections, you have to be concerned about missed messages and retries”* for O₅, O₆, instead
 1172 only considers it a viable solution. Furthermore, the respondent belonging to O₂ mentioned as an
 1173 alternative solution the *“introduction of quarantine builds together with an appropriate process of*
 1174 *how to deal with these tests”*.

1175
 1176

4.3 RQ₂: How relevant are the identified CI/CD challenges/barriers and their mitigation for practitioners involved in CPS development?

This research question describes the results of the evaluation of the findings in RQ₁ made through an external survey leveraging practitioners that have not been involved in the semi-structured interviews. Note that we have only validated the barriers and the pipeline-related challenges together with their associated mitigation strategies.

As regards the five barriers encountered when trying to configure a CI/CD pipeline for CPS development, by looking at the results in Figure 3, we found that among the participants who answered each question, the limited number of human and software/hardware resources together with the presence of complex non-functional requirements to be checked within the pipeline are the ones felt as more relevant (> 72%). Furthermore, while 30 out of 55 respondents still consider as relevant the barriers dealing with security aspects hindering the inclusion of HiL in the CI/CD process, 31% do not consider such barriers as a real impediment. All the three mitigation previously identified were considered relevant by the survey participants. Specifically, the adoption of test case prioritization techniques is predominant (31 out of 55 respondents), followed by the usage of simulators or mock-ups (28), and the usage of incremental builds (17).

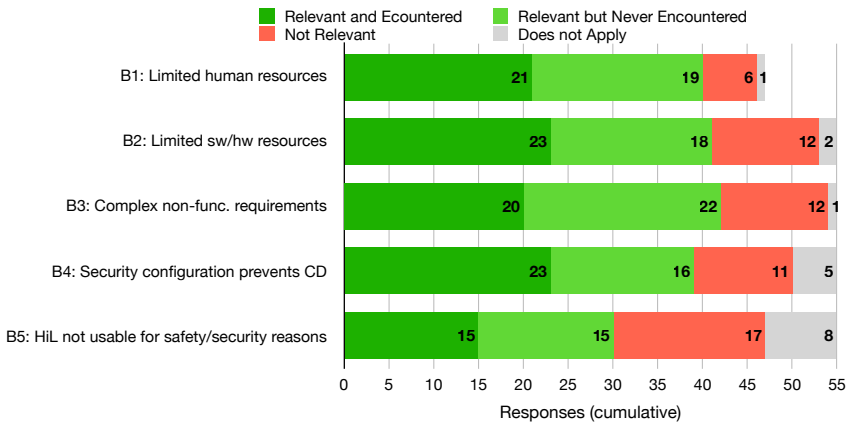


Fig. 3. Results of barriers perception

Figure 4 shows the results of the survey in terms of the six challenges belonging to the Pipeline Properties category. Unsurprisingly, 47 out of 55 respondents consider the long build execution time as a relevant challenge. Also, while from the semi-structured interviews the remaining five challenges were experienced by one or at most two different organizations, the survey indicates how some of such challenges are felt as relevant by more than 69% of our participants. These are (i) the need to properly estimate the build time before timing out the CI/CD process, (ii) the difficulty in properly configuring static code analysis tools, and (iii) the presence of a CI/CD configuration highly coupled with the environment. Regarding the impossibility of having access to the production code, if we do not consider the seven participants reporting that this challenge cannot apply to their context, ≈ 31% of the respondents do not consider it as a relevant challenge.

Moving onto the mitigation strategies, more than half of our respondents (30) rely on test case prioritization techniques, 28 rely on parallelization, 26 rely on nightly builds for time-intensive tasks, while 18 consider useful the adoption of incremental builds during normal working hours. Finally, one participant reported a new mitigation strategy dealing with long builds where “we simulate faster than in the reality where possible”, however, the same participant also points out

the drawback of this mitigation, *i.e.*, having different build outcomes when using simulators and HiL—“*this can introduce subtle timing differences in the test results*”.

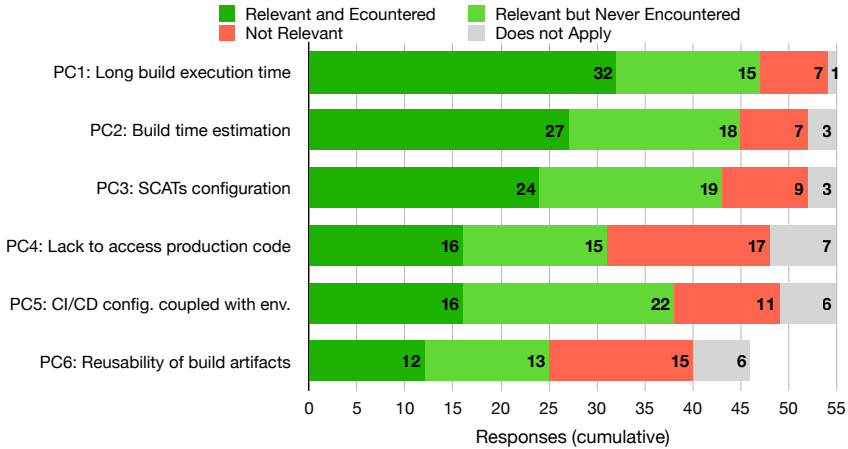


Fig. 4. Results of pipeline challenges perception: Pipeline Properties

For what concerns pipeline-related challenges in the Thoroughness category dealing with ensuring the overall accuracy and completeness of the CI/CD pipeline (see PC₇, PC₈ and PC₉ in Figure 5), differently from the RQ₁ results, more than half of our survey respondents consider the presence of a development environment detached from the execution environment as a real impediment to set up a CI/CD process for CPSs. This is also true for the difficulties in detecting deployment-related errors (39 respondents). The above differences stress the impossibility to have a “standardized” CI/CD configuration that can be applied to almost all the CPS domains. While all participants considered the adoption of containerization a viable solution to overcome these challenges, one new mitigation strategy comes up from a survey participant which, for PC₈, suggests the possibility of developing and adopting static analysis tools able to analyze (and detect errors from) deployment scripts.

Moving to the three challenges related to closing the DevOps loop by gathering data from the hardware, *i.e.*, real environment, by looking at the bottom part of Figure 5 it is possible to state that more than 65% of the respondents consider them as relevant, with the presence of uncontrollable factors to account for having the highest percentage ($\approx 71\%$). While from RQ₁ we did not find any mitigation strategy for these challenges, we obtained some feedback from eight survey respondents. First of all, it could be possible to continuously analyze the logs also after the operation has started. At the same time, one survey respondent points out the possibility to make the monitoring less impactful on performance by “*disabl[ing] invasive logging methods*”. For what concerns the presence of uncontrollable factors, one respondent pointed out how using continuous testing allows to “*better overcome the problem of the uncontrollable factors in real life systems and usually diminish the future costs and improve efficiency*”. There are also mitigation strategies dealing with the overall CI/CD process. Specifically, one respondent mentions the possibility to use parallel DataOps observability pipeline, *i.e.*, “*we use ELK, but it is still under debate/migration*”. While a different respondent highlights as possible mitigation the presence of “*cross-functional teams which bring in more collaborations and ideas*”.

Figure

inclusion of simulators in the CI/CD process. As can be seen from the figure, at least 36 out of 55

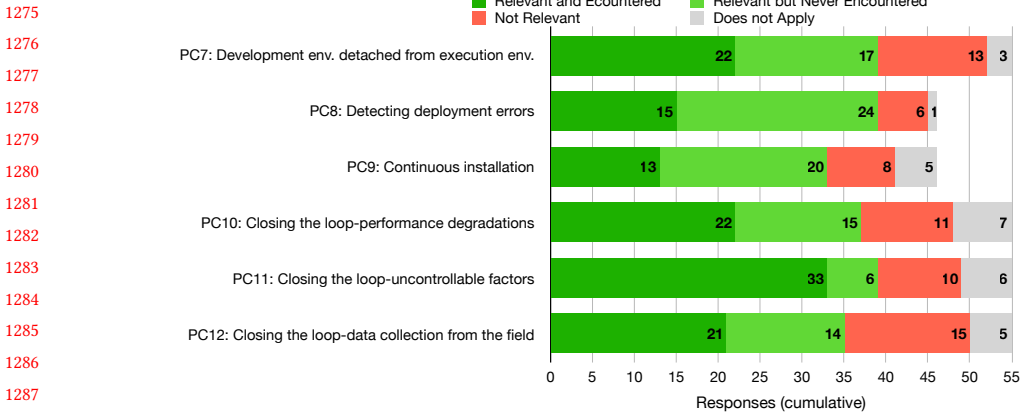


Fig. 5. Results of pipeline challenges perception: Thoroughness

respondents considered such challenges relevant. The only exception is the challenge of dealing with the impossibility of accessing the third-party simulators adopted in the pipeline (PC₁₇). In this case, 13 respondents mention that it does not apply to their context, meaning that the simulators are mainly self-developed within the organization they belong to. In terms of mitigation, instead, (i) 28 respondents use a CI/CD process made up of both simulators and HiL for overcoming the presence of limited functionality, (ii) 24 use both simulator and HiL for overcoming the complexity due to a tight interaction among different components and the environment, and (iii) among the 31 respondents struggling with the simulators’ accessibility, 14 adopt the “timeout” feature.

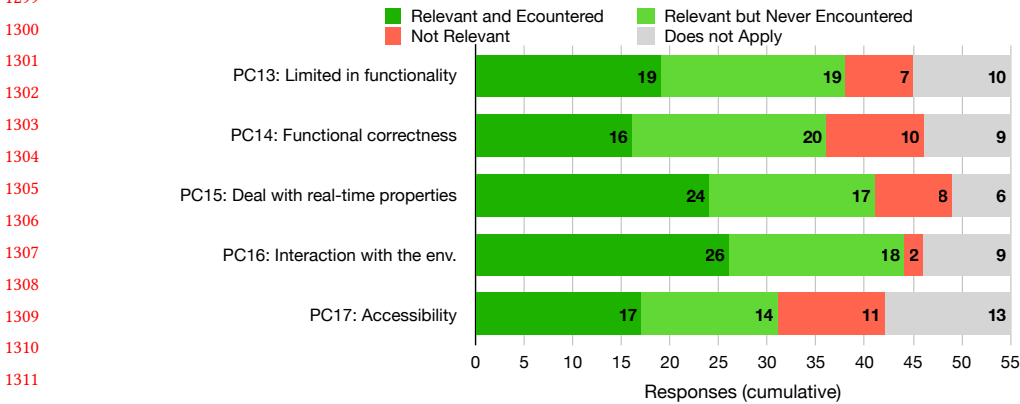


Fig. 6. Results of pipeline challenges perception: Simulators

As regards the four challenges dealing with the inclusion of HiL in the CI/CD process, as shown in Figure 7, the costs and scalability challenge is the predominant one (49 out of 55 respondents), followed by the need to check for HiL availability (41), and the complexity for automating both deployment and testing activities on HiL (42 and 40 respondents for PC₁₉ and PC₂₀ respectively). No new mitigation strategy comes up from the survey results. However, 28 respondents confirm that the adoption of simulators and HiL in different build stages can help to deal with costs and scalability issues, while 19 adopt the “green-build” rule, *i.e.*, HiL can only be considered when the CI/CD process relying on simulators has a green status.

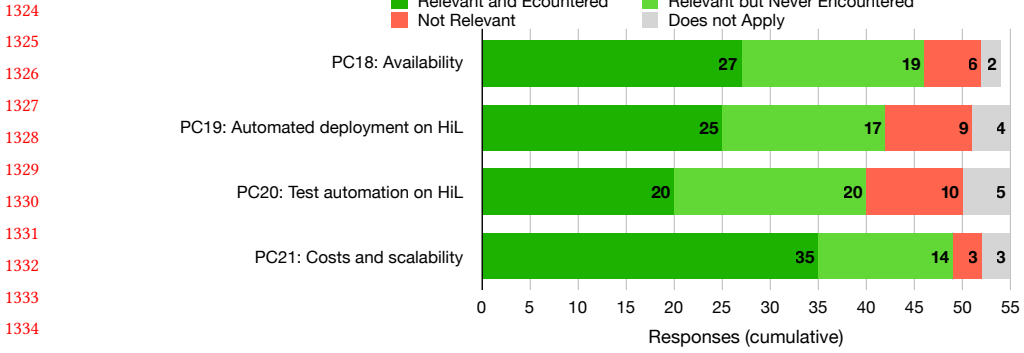


Fig. 7. Results of pipeline challenges perception: HiL

The last category of pipeline-related challenges being validated through the external survey considers the root cause for flaky behavior experienced in the CI/CD process. As shown in Figure 8, for each challenge we have that more than half of our respondents consider it relevant for CPSs. Moreover, 49 out of 55 respondents consider challenging to deal with HiL availability and simulators not coping with timing issues. Unsurprisingly, the two challenges being not specific to the CPS development, *i.e.*, PC₂₂ and PC₂₆, are the ones where several respondents (13 and 12 respectively) mentioned that it is not relevant. Fixing the pipeline configuration is the most frequent mitigation strategy, as indicated by 31 respondents, while no further mitigation strategies are suggested.

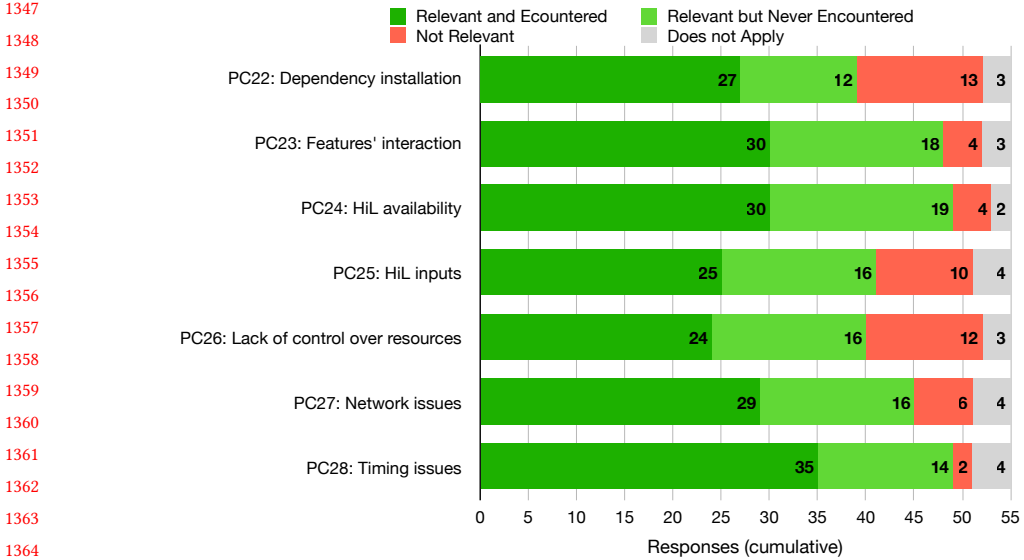


Fig. 8. Results of pipeline challenges perception: Flaky Behavior

Finally, by looking at the 15 answers to the open-ended question aimed at eliciting other challenges that we did not encounter in the semi-structured interviews, we gathered the following, additional challenges:

- (1) *Guaranteeing the supply chain security* (three respondents);

- 1373 (2) *The impossibility to use simulated environments* unless the quality of specific data types is
1374 ensured (one respondent);
- 1375 (3) *The need to have field tests included in the DevOps cycle even with a lower frequency* as both
1376 simulators and HiL can only cover a fraction of what usually happens during real field testing
1377 activities (one respondent);
- 1378 (4) *The difficulty to implement a “quick-retry” feature* in the CI/CD process, to selectively rollback
1379 at specific stages mainly because this is highly dependent on the infrastructure language
1380 (one respondent); and
- 1381 (5) *The difficulty to reduce the build execution time when dealing with HiL due to the need for*
1382 *checking the HiL availability, i.e., “very long hardware boot times”* (one respondent).
1383

1384 5 DISCUSSION AND IMPLICATIONS

1385

1386 This section summarizes the main findings and implications of our study. We divide the section
1387 into implications for (i) developers, (ii) educators, and (iii) researchers.

1388 5.1 Implications for developers

1389

1390 We start by discussing what, based on insights learned from this study, developers must consider
1391 when trying to set up and evolve a CI/CD pipeline for CPS development.

1392 **Simulators are necessary to achieve continuous builds on CI/CD pipelines.** Performing
1393 CI/CD on real hardware is often unfeasible, for different reasons. The automated deployment may
1394 be complicated, or the hardware may not be available onsite. Also, organizations doing V&V tasks
1395 only may have limited/no access to hardware, simulators, or even to the production code. Therefore,
1396 simulation is often the only choice available. However, having a reliable simulator is challenging
1397 for many CPS developers. While in some cases simulators come from hardware producers, in other
1398 circumstances the only option is to develop them in-house. This requires the allocation of suitable
1399 skills and efforts in the development process. Failing to do so would have severe consequences
1400 on the ability to setup not only CI/CD, but even simple test automation without relying on the
1401 hardware directly, when this is possible.

1402 **Balancing the use of simulators and HiL in the pipeline.** Deploying and running CPSs on
1403 HiL at every change could be troublesome, expensive, and may result in slow feedback. At the same
1404 time, for the reasons mentioned before, it is unlikely that developers could fully trust a quality
1405 assessment performed solely on simulators. Therefore, it is highly desirable to configure staged
1406 builds relying on different execution environments, namely (i) continuous builds on simulators,
1407 aimed at providing fast feedback to developers (*e.g.*, about the outcome of static checks, or possible
1408 integration issues discovered by tests); and (ii) periodic (*e.g.*, nightly) builds on HiL, to verify
1409 whether the assumptions made on simulators are still valid, checking properties that often cannot
1410 be verified on simulators (*e.g.*, response time properties), testing the system in scenarios that cannot
1411 be easily simulated, or verifying the compatibility of the software against hardware variants not
1412 fully reproduced by the simulators.

1413 **Late delivery is the crux of CPS development.** For the reasons explained above, CPS software
1414 tends to reach target production hardware very late in the development. This has several negative
1415 side effects, including the late discovery of defects that could not be identified through simulation,
1416 but also having a system that reaches the end user very late. Allocating sufficient effort, resources,
1417 and competences to enable automated delivery is therefore highly desirable.

1418 **Having hardware experts onboard may be a plus.** Based on what we learned from this study,
1419 it is clear how CPS development may highly benefit from the availability of both software and
1420 hardware experts so that it is much easier to self-develop simulators whose behavior is as much
1421

1422 as possible like the one of the real device. This would help to reduce the differences that might
1423 be observed in terms of build outcome, *e.g.*, the number and type of failing tests, when running
1424 the process on simulators and HiL. On the one hand, the presence of hardware experts in a team,
1425 when available, has been found useful by our interviewees. On the other hand, a context in which
1426 both hardware and software evolve makes tasks such as change impact analysis more challenging
1427 to handle, *e.g.*, to determine whether and to what extent a hardware change would impact some
1428 software components, or some software evolution would hinder the integration of certain pieces of
1429 hardware.

1430

1431 5.2 Implications for educators

1432 In the following, we discuss what, based on insights learned from this study, would be expected for
1433 what concerns the creation (or enhancement) of curricula related to CPS development.

1434 **Blended curricula with hardware and software competencies.** Any effective DevOps
1435 organizational setting or management of a CI/CD pipeline likely requires software engineering
1436 expertise other than what is currently taught in regular graduate-level courses, *e.g.*, knowledge
1437 about the hardware, software-hardware interplay, and domain standards expertise. On the one
1438 hand, university curricula shall strive to include such aspects in their teaching. On the other hand,
1439 practitioners should more actively engage in standardizing CPS application lifecycle management
1440 practices, patterns, and tools to enable the aforementioned educational augmentation exercise.

1441 **Specialized courses on simulator development.** In a context for which CPS specific curricula
1442 are highly desirable, one competence assumes paramount importance, and this is the development
1443 of simulators. The latter requires combining knowledge from physics, automated control (*e.g.*,
1444 system dynamics, discrete systems), and virtual reality (many simulators leverage 3D or even
1445 virtual reality environments, similar to those used in video games).

1446 **Teaching CI/CD in complex, heterogeneous environments.** CI/CD is oftentimes taught
1447 in the context of conventional system development. To favor the adoption of CI/CD for complex
1448 systems, and in particular for CPSs, courses on CI/CD should touch on topics related to (i) coping
1449 with complex hardware or simulators attached to the pipeline, and (ii) pondering fast builds with
1450 the need for testing a CPS on multiple devices (or simulators), where this is appropriate. Also, while
1451 conventional CI/CD literature advocates “building at every change” [13], CPS developers need to
1452 face with reality, and therefore such a common wisdom need to be revisited. Similarly, we found
1453 that for large and complex CPSs “retest all” does not work, and therefore incremental builds are a
1454 widely adopted practice.

1455 **Software architectures for CPSs.** CPSs heavily interact with HiL interfaces (and sometimes
1456 multiple HiL, having different characteristics and varying APIs) and, during the development
1457 process, with simulators. The latter may be updated or even replaced by better ones. From an
1458 educational perspective, it is desirable that courses related to software architectures properly treat
1459 such scenarios, discussing the proper architectural choices or design choices allowing an easy (even
1460 at run-time) replacement of different kinds of HiL and simulators in the software systems. Software
1461 components of CPSs may need to be deployed on, or interact with, multiple types of devices (*e.g.*,
1462 a control software may be deployed on different car models). This requires that developers must
1463 have suitable knowledge of product line engineering and follow related practices when designing
1464 CPSs. Furthermore, it is desirable to teach prospective CPS developers about how to design a CPS
1465 architecture to make a system scalable, but also secure, and easy to be monitored and tested.

1466

1467 5.3 Implications for researchers

1468 Implications for researchers aim at developing approaches and tools to support developers in setting
1469 up, maintaining, and using CI/CD pipelines for CPSs.

1470

1471 **The target environment of CPS is multifaceted and diversified, making CI/CD pipelines**
1472 **complex and expensive.** Often a CPS may target multiple devices, as well as both HiL and
1473 simulators. This may entail a build matrix against which the pipeline must be run, *i.e.*, the matrix
1474 describes different combinations of parameters (*e.g.*, simulator models, HiL instances, other settings)
1475 for the build. That being said, it is possible that, while some changes may entail different behavior
1476 on different matrix instances, other changes do not, and therefore running the build on all possible
1477 configurations would be a waste of resources. On the one hand, this stimulates research towards
1478 approaches aimed at recommending the creation of a suitable build matrix system based on similar
1479 systems, and in general systems targeting similar devices. Also, these kinds of recommenders
1480 should be able to point out the need for maintaining build matrices by learning “from the crowd”,
1481 *e.g.*, the need to prune out obsolete environments and add new ones. On the other hand, proper
1482 approaches should be developed to trigger builds on different matrix instances based on the changes
1483 performed.

1484 **Coping with multiple root causes for flaky behavior.** The complex technological stack,
1485 the behavior of simulators and HiL, their (sometimes uncontrollable) unavailability or lack of
1486 accessibility, and the mechanisms used to collect test outputs (*e.g.*, sensors or video cameras)
1487 require not only to better monitor all possible elements causing flakiness, but also to combine
1488 and enhance various mitigation approaches, including checking the status of HiL/simulators, and
1489 leveraging the “usual” retries. As indicated by the participants, flaky behaviors in CPSs are often
1490 due to the complex interacting environment (*e.g.*, lack of complete control on the hardware status)
1491 rather than on the order with which the tests are executed. Hence, flaky test detectors that target
1492 flaky tests considering their ordering [24, 88] are not effective for environment-dependent flakiness.
1493 CPS-specific detectors could be inspired to those for undetermined specifications [87], or based on
1494 machine learning models [59] but trained on CPSs data and encompassing CPS-specific features,
1495 *e.g.*, changes to simulators or HiL configurations, as well as their build logs. To improve CI/CD
1496 infrastructures for CPS, it may be useful to develop recommenders, integrated into the pipeline,
1497 able to support developers in the identification of flakiness behavior, and identify its root causes.

1498 **Challenges in automated test execution.** We found that one of the reasons that impede
1499 full CI/CD automation for CPS is the difficulty to automate test execution, especially when the
1500 system is deployed on the hardware. That is, the system receives inputs from sensors and interacts
1501 with actuators. Full test automation requires (i) tools, such as scenario generators or record replay
1502 tools able to seed inputs to the CPS, and (ii) the capability of CI/CD infrastructure to support the
1503 execution of such tools. Very often these tools are GUI-oriented and not particularly well-suited to
1504 be integrated in a CI/CD pipeline.

1505 **Challenges in automated oracle creation.** The CPS execution environment (*e.g.*, simulators
1506 or HiL) drastically complicates the definition and automatic check of oracles. The latter requires
1507 to ponder several factors: (i) the test scenario (or requirement to assess); (ii) the accepted level
1508 of realism in simulations; (iii) the readiness level or maturity of the hardware proxies used in
1509 the pipeline; and (iv) the output sources, *e.g.*, based on actual sensors’ data or mocking/synthetic
1510 data. Besides, the oracles consist of value ranges (*e.g.*, time intervals) instead of scalars, or they
1511 may be signals that need to be properly processed, as highlighted in existing studies on testing
1512 for CPS [6, 53]. It may be important to account for non-functional properties, including timing
1513 ones [80]. Also, to cope with inputs originating from sensors or even from a multimedia recording
1514 of the CPS execution (as pointed out by O_3 and O_4), it is desirable to develop approaches for pattern
1515 recognition [28, 38, 72].

1516 **Need for specific fault models.** Looking more broadly at configuring V&V phases within
1517 the CPS pipeline, respondents would like to early discover some defects through static analysis.
1518 This requires a clear fault modeling in the CPS context (as the ones for autonomous cars [26] and
1519

1520 unmanned vehicles [81]), but also to develop CPS-specific linters, which can be integrated into the
1521 CI/CD pipelines to allow early detection of build failures, hence avoid to perform expensive testing
1522 activities, and hence long builds, which constitute a major problem for CPS developers according
1523 to our study results. Moreover, CPS-specific fault models can be useful for other purposes, not only
1524 to facilitate root-cause analysis [26], but also to create domain-specific mutation testing strategies,
1525 as it has been done in other cases such as deep learning [36] or mobile development [19, 76].
1526

1527

1528

1528 6 THREATS TO VALIDITY

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

1551

1552

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

1566

1567

1568

Threats to *construct validity* concern the relationship between theory and observations. The interview participants might have misinterpreted our questions, or they might have reported their personal (and biased) views of the phenomenon. While this is typical for interview-based studies [17, 33], we mitigated the threat by using semi-structured interviews and following up with clarifications every time we realized this was needed.

There could be threats to construct validity related to how survey respondents interpreted the survey questions and provided their answers. We have mitigated this threat by providing a self-explanatory description of the challenges, barriers, and mitigation strategies. Also, we left them the possibility to provide open comments to also point out cases of misunderstanding. However, based on the provided answers, we had no evidence of cases where respondents had difficulties in understanding the posed questions. In addition, for the external survey, we leveraged demographic information to filter out responses where the information provided made it evident that a participant had not the required knowledge.

Threats to *internal validity* concern confounding factors that could have influenced our results. To limit subjectivity in our coding, we employed multiple coders, computed inter-rater reliability, and used follow-up discussions not only to resolve cases of inconsistent coding, but to review any single coding. We elicited codes and relations only based on explicit occurrences of words in the transcripts. However, we could not exclude imprecision due to our interpretation of the participants' answers.

Another threat could be the low representativeness of the respondents in the semi-structured interviews and, to some extent, in the external survey. In the first case, participants were obtained through personal contacts, as we need people available to participate in a relatively long interview. However, such participants cover a relatively diversified set of domains (8). As for the survey, the use of snowballing and especially the use of *Prolific* allowed us to mitigate a possible bias due to the direct personal contacts.

Threats to *reliability validity* relate to the extent to which results can be reproduced. To achieve this goal, we (i) have described the data collection and analysis process in detail, and (ii) provide in our replication package the detailed outcome of the coding phases.

Finally, threats to *external validity* concern the generalizability of our findings. The interview-based study has been conducted involving 10 organizations developing CPS for 8 different domains. We are aware that the obtained findings may not generalize to different organizations and domains. Indeed, from the performed interviews, we found that CI/CD pipelines were extremely different from case to case. Therefore, as in other interview studies conducted within a limited set of organizations, and also considering the study topic, the generalizability is relatively limited. To mitigate this threat, when addressing RQ₂ we have validated the findings collected in RQ₁ through an external survey with practitioners different from the ones involved in our semi-structured interviews, and belonging to 9 domains. Still, it is possible that, also in this case, as Figure 2 shows, some domains are better covered than others, as well as some are still not covered at all.

7 CONCLUSION AND FUTURE WORK

In this paper, we investigated the adoption, usage, and evolution of CI/CD pipelines for CPS development, by focusing on challenges and barriers that DevOps teams face when setting up or evolving CI/CD processes for CPS development highlighting that the configuration is highly dependent on the domain. The study is based on interviews from 10 organizations developing CPSs in 8 different domains, followed by a member-checking survey within the same development teams, and an external validation survey involving 55 participants from 9 domains. By performing an open coding on the interview results, we have elicited a set of challenges/barriers, along with their mitigation strategies.

The obtained findings are a first step towards supporting DevOps teams in properly using and configuring CI/CD for CPSs. Also, they have implications on how to enhance education/training for CPS developers, and trigger future research. Based on that, future work aims at triangulating this study through other channels, e.g., in-field observations, and at investigating bad practices in applying and maintaining CI/CD for CPSs. In particular, our goal will be to automatically detect, by analyzing CI/CD pipeline configurations and run-time data, problematic situations (“smells”) that would require an intervention on the DevOps side, and, for what possible, automatically suggest repairs.

ACKNOWLEDGMENTS

We gratefully acknowledge the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS. Also, we would like to thank the study participants for taking their time to participate in the study and for providing insightful feedback.

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*. Springer Intern. Publishing, Cham, 194–207.
- [3] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. Which Commits Can Be CI Skipped? *IEEE Trans. Software Eng.* 47, 3 (2021), 448–463.
- [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)*. IEEE, 143–154.
- [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, Dusseldorf, Germany*. 326–338. https://doi.org/10.1007/978-3-030-56441-4_24
- [6] Aitor Arrieta, Shuai Wang, Ainhua 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*. 1411–1418.
- [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*. Springer, 5–23.
- [8] Sebastian Baltes and Paul Ralph. Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering* 27, 4 (2022), 1–31.
- [9] L. Chen. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software* 32, 2 (2015), 50–54.
- [10] Lianping Chen. Continuous Delivery: Overcoming adoption challenges. *Journal of Systems and Software* 128 (2017), 72 – 86.
- [11] 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*. 4–5. <https://doi.org/10.1109/ICSE-Companion.2019.00023>

- 1618 [12] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: An Open Urban
 1619 Driving Simulator. In *1st Annual Conference on Robot Learning, CoRL 2017 (Proceedings of Machine Learning Research)*,
 1620 Vol. 78. PMLR, 1–16.
- 1621 [13] Paul Duvall, Stephen M. Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing
 1622 Risk*. Addison-Wesley.
- 1623 [14] Paul M. Duvall. Continuous integration. Patterns and Antipatterns. *DZone refcard #84* (2010). <http://bit.ly/l8rfVS>
- 1624 [15] Paul M. Duvall. Continuous Delivery: Patterns and Antipatterns in the Software Life Cycle. *DZone refcard #145* (2011).
 1625 <https://dzone.com/refcardz/continuous-delivery-patterns>
- 1626 [16] Moritz Eck, Fabio Palomba, Marco Castelluccio, and Alberto Bacchelli. Understanding flaky tests: the developer’s
 1627 perspective. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the
 1628 Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. 830–840.
- 1629 [17] Omar Elazhary, Margaret-Anne D. Storey, Neil A. Ernst, and Elise Paradis. ADEPT: A Socio-Technical Theory of
 1630 Continuous Integration. In *43rd IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging
 1631 Results, ICSE (NIER) 2021, Madrid, Spain, May 25-28, 2021*. 26–30.
- 1632 [18] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A Ernst, and Margaret-Anne Storey. Uncovering the
 1633 benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering* (2021).
- 1634 [19] Camilo Escobar-Velásquez, Mario Linares-Vásquez, Gabriele Bavota, Michele Tufano, Kevin Moran, Massimiliano Di
 1635 Penta, Christopher Vendome, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Enabling Mutant Generation for Open-
 1636 and Closed-Source Android Apps. *IEEE Trans. Software Eng.* 48, 2 (2022), 186–208. [https://doi.org/10.1109/TSE.2020.
 1637 2982638](https://doi.org/10.1109/TSE.2020.2982638)
- 1638 [20] Erik Flores-García, Goo-Young Kim, Jinho Yang, Magnus Wiktorsson, and Sang Do Noh. Analyzing the Characteristics
 1639 of Digital Twin and Discrete Event Simulation in Cyber Physical Systems. In *Advances in Production Management
 1640 Systems. Towards Smart and Digital Manufacturing (IFIP Advances in Information and Communication Technology)*,
 1641 Vol. 592. Springer, 238–244.
- 1642 [21] Martin Fowler and Matthew Foemmel. Continuous integration. [https://web.archive.org/web/20200522100521/https://
 1643 martinfowler.com/articles/originalContinuousIntegration.html](https://web.archive.org/web/20200522100521/https://martinfowler.com/articles/originalContinuousIntegration.html). Accessed: 2021-11-21.
- 1644 [22] Heinz Gall. Functional safety IEC 61508 / IEC 61511 the impact to certification and the user. In *2008 IEEE/ACS
 1645 International Conference on Computer Systems and Applications*. 1027–1031.
- 1646 [23] Keheliya Gallaba and Shane McIntosh. Use and misuse of continuous integration features: An empirical study of
 1647 projects that (mis) use Travis CI. *IEEE Trans. Software Eng.* 46, 1 (2018), 33–50.
- 1648 [24] Alessio Gambi, Jonathan Bell, and Andreas Zeller. Practical test dependency detection. In *2018 IEEE 11th International
 1649 Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–11.
- 1650 [25] Alessio Gambi, Tri Huynh, and Gordon Fraser. Generating effective test cases for self-driving cars from police reports.
 1651 In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations
 1652 of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 257–267.
- 1653 [26] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, and Qi Alfred Chen. A comprehensive study of
 1654 autonomous vehicle bugs. In *ICSE ’20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27
 1655 June - 19 July, 2020*. 385–396.
- 1656 [27] Smitha Gautham, Athira Varma Jayakumar, Abhi Rajagopala, and Carl Elks. Realization of a Model-Based DevOps
 1657 Process for Industrial Safety Critical Cyber Physical Systems. In *2021 4th IEEE International Conference on Industrial
 1658 Cyber-Physical Systems (ICPS)*. 597–604.
- 1659 [28] Farshad Gholami, Niousha Attar, Hassan Haghighi, Mojtaba Vahidi Asl, Meysam Valueian, and Saina Mohamadyari. A
 1660 classifier-based test oracle for embedded software. In *2018 Real-Time and Embedded Systems and Technologies (RTEST)*.
 1661 104–111.
- 1662 [29] Carlos A. González, Mojtaba Varmazyar, Shiva Nejati, Lionel C. Briand, and Yago Isasi. Enabling Model Testing of
 1663 Cyber-Physical Systems. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering
 1664 Languages and Systems (MODELS ’18)*. Association for Computing Machinery, New York, NY, USA, 176–186.
- 1665 [30] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics* (1961), 148–170.
- 1666 [31] Robert M. Groves, Floyd J. Jr. Fowler, Mick P. Coupyer, James M. Lepkowski, Eleanor Singer, and Roger Tourangeau.
Survey Methodology, 2nd edition. Wiley.
- [32] Philipp Helle, Wladimir Schamai, and Carsten Strobel. Testing of Autonomous Systems - Challenges and Current
 State-of-the-Art. *INCOSE International Symposium* (2016), 571–584.
- [33] 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*.
- [34] 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*

- 1667 *Engineering (ASE)*.
- 1668 [35] David L. Hoover. Textual Variation, Text-Randomization, and Microanalysis.. In *DH. Alliance of Digital Humanities*
- 1669 *Organizations (ADHO)*, 223–225. <http://dblp.uni-trier.de/db/conf/dihu/dh2016.html#Hoover16a>
- 1670 [36] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. DeepCrime: mutation testing of deep learning systems
- 1671 based on real faults. In *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual*
- 1672 *Event, Denmark, July 11-17, 2021*. 67–78. <https://doi.org/10.1145/3460319.3464825>
- 1673 [37] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. Quality metrics and oracles for autonomous vehicles testing. In
- 1674 *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 194–204.
- 1675 [38] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. Quality Metrics and Oracles for Autonomous Vehicles Testing.
- 1676 In *14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, April 12-16,*
- 1677 *2021*. IEEE, 194–204. <https://doi.org/10.1109/ICST49551.2021.00030>
- 1678 [39] Suzette Johnson, Harry Koehneman, Diane LaFortune, Dean Leffingwell, Stephen Magill, Steve Mayner, Avigail Ofer,
- 1679 Robert Stroud, Anders Wallgren, and Robin Yeman. *Industrial DevOps - Applying DevOps and Continuous Delivery to*
- 1680 *Significant Cyber-Physical Systems*. National Academies Press. <https://itrevolution.com/book/industrial-devops/>
- 1681 [40] Nidhi Kalra and Susan Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous
- 1682 vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (12 2016), 182–193.
- 1683 [41] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research part 2: designing a survey. *ACM*
- 1684 *SIGSOFT Software Engineering Notes* 27, 1 (2002), 18–20.
- 1685 [42] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research: part 3: constructing a survey
- 1686 instrument. *ACM SIGSOFT Software Engineering Notes* 27, 2 (2002), 20–24.
- 1687 [43] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research part 4: questionnaire evaluation.
- 1688 *ACM SIGSOFT Software Engineering Notes* 27, 3 (2002), 20–23.
- 1689 [44] Barbara A. Kitchenham and Shari Lawrence Pfleeger. Principles of survey research: part 5: populations and samples.
- 1690 *ACM SIGSOFT Software Engineering Notes* 27, 5 (2002), 17–20.
- 1691 [45] Klaus Krippendorff. Reliability in Content Analysis: Some common Misconceptions and Recommendations. *Journal of*
- 1692 *the Royal Statistical Society. Series B (Methodological)* 30, 3 (2004), 411–433.
- 1693 [46] Wing Lam, Patrice Godefroid, Suman Nath, Anirudh Santhiar, and Suresh Thummalapenta. Root Causing Flaky Tests
- 1694 in a Large-Scale Industrial Setting. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing*
- 1695 *and Analysis (ISSTA 2019)*. ACM, 101–111.
- 1696 [47] Wing Lam, Stefan Winter, Angello Astorga, Victoria Stodden, and Darko Marinov. Understanding Reproducibility and
- 1697 Characteristics of Flaky Tests Through Test Reruns in Java Projects. In *31st IEEE International Symposium on Software*
- 1698 *Reliability Engineering, ISSRE, Coimbra, Portugal, October 12-15, 2020*. 403–413.
- 1699 [48] Wing Lam, Stefan Winter, Anjiang Wei, Tao Xie, Darko Marinov, and Jonathan Bell. A large-scale longitudinal study
- 1700 of flaky tests. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 202:1–202:29. <https://doi.org/10.1145/3428270>
- 1701 [49] Johannes Lampel, Sascha Just, Sven Apel, and Andreas Zeller. When life gives you oranges: detecting and diagnosing
- 1702 intermittent job failures at Mozilla. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering*
- 1703 *Conference and Symposium on the Foundations of Software Engineering*. 1381–1392.
- 1704 [50] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep
- 1705 drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics* 36, 1 (2019), 1–14.
- 1706 [51] Ivano Malavolta, Grace Lewis, Bradley Schmerl, Patricia Lago, and David Garlan. How Do You Architect Your
- 1707 Robots? State of the Practice and Guidelines for ROS-Based Systems. In *Proceedings of the ACM/IEEE 42nd International*
- 1708 *Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '20)*. ACM, New York, NY, USA, 31–40.
- 1709 [52] Torvald Mårtensson, Daniel Ståhl, and Jan Bosch. Continuous integration applied to software-intensive embedded
- 1710 systems—problems and experiences. In *International Conference on Product-Focused Software Process Improvement*.
- 1711 Springer, 448–457.
- 1712 [53] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-
- 1713 intensive cyber-physical models: An approach based on system identification. In *2020 IEEE/ACM 42nd International*
- 1714 *Conference on Software Engineering (ICSE)*. IEEE, 372–384.
- 1715 [54] Vuong Nguyen, Stefan Huber, and Alessio Gambi. SALVO: Automated Generation of Diversified Tests for Self-driving
- 1716 Cars from Existing Maps. In *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 128–135.
- 1717 [55] Helena Holmstrom Olsson, Hiva Alahyari, and Jan Bosch. Climbing the "Stairway to Heaven" – A Multiple-Case
- 1718 Study Exploring Barriers in the Transition from Agile Development Towards Continuous Deployment of Software. In
- 1719 *Proceedings of the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA '12)*. IEEE
- 1720 Computer Society, Washington, DC, USA, 392–399.
- 1721 [56] Heejong Park, Arvind Easwaran, and Sidharta Andalām. Challenges in Digital Twin Development for Cyber-Physical
- 1722 Production Systems. In *Cyber Physical Systems. Model-Based Design*. Springer International Publishing, Cham, 28–48.
- 1723
- 1724
- 1725

- 1716 [57] Shari Lawrence Pfleeger and Barbara A. Kitchenham. Principles of survey research: part 1: turning lemons into
 1717 lemonade. *ACM SIGSOFT Software Engineering Notes* 26, 6 (2001), 16–18.
- 1718 [58] Gustavo Pinto, Breno Miranda, Supun Dissanayake, Marcelo d’Amorim, Christoph Treude, and Antonia Bertolino.
 1719 What is the Vocabulary of Flaky Tests?. In *MSR ’20: 17th International Conference on Mining Software Repositories, Seoul,
 1720 Republic of Korea, 29-30 June, 2020*. 492–502.
- 1721 [59] Gustavo Pinto, Breno Miranda, Supun Dissanayake, Marcelo d’Amorim, Christoph Treude, and Antonia Bertolino.
 1722 What is the vocabulary of flaky tests?. In *Proceedings of the 17th International Conference on Mining Software Repositories*.
 1723 492–502.
- 1724 [60] Akond Rahman, Chris Parnin, and Laurie Williams. The Seven Sins: Security Smells in Infrastructure As Code Scripts.
 1725 In *Proceedings of the 41st International Conference on Software Engineering (ICSE ’19)*. IEEE Press, 164–175.
- 1726 [61] Steven P. Reiss. Automatic code stylizing. In *22nd IEEE/ACM International Conference on Automated Software Engineering
 1727 (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*. 74–83.
- 1728 [62] Vincenzo Riccio and Paolo Tonella. Model-based Exploration of the Frontier of Behaviours for Deep Learning System
 1729 Testing. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations
 1730 of Software Engineering (ESEC/FSE ’20)*. Association for Computing Machinery.
- 1731 [63] Terry Rowlands, Neal Waddell, and Bernard McKenna. Are We There Yet? A Technique to Determine Theoretical
 1732 Saturation. *J. Comput. Inf. Syst.* 56, 1 (2016), 40–47. <http://dblp.uni-trier.de/db/journals/jcis/jcis56.html#RowlandsWM16>
- 1733 [64] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. Continuous deployment
 1734 at Facebook and OANDA. In *Companion Proceedings of the 38th International Conference on Software Engineering (ICSE
 1735 Companion)*. 21–30.
- 1736 [65] August Shi, Jonathan Bell, and Darko Marinov. Mitigating the effects of flaky tests on mutation testing. In *Proceedings
 1737 of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA*. 112–122.
- 1738 [66] Hesham Shokry and Mike Hinchey. Model-based verification of embedded software. (2009).
- 1739 [67] Sebastian Sontges and Matthias Althoff. Computing the Drivable Area of Autonomous Road Vehicles in Dynamic
 1740 Road Scenes. *IEEE Trans. Intell. Transp. Syst.* 19, 6 (2018), 1855–1866.
- 1741 [68] Donna Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media.
- 1742 [69] Daniel Ståhl and Jan Bosch. Automated software integration flows in industry: a multiple-case study. In *Companion
 1743 Proceedings of the 36th International Conference on Software Engineering*. ACM, 54–63.
- 1744 [70] Daniel Ståhl and Jan Bosch. Modeling continuous integration practice differences in industry software development. *J.
 1745 Syst. Softw.* 87 (2014), 48–59.
- 1746 [71] Daniel Ståhl, Kristofer Hallén, and Jan Bosch. Continuous Integration and Delivery Traceability in Industry: Needs
 1747 and Practices. In *42th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016, Limassol,
 1748 Cyprus, August 31 - Sept. 2, 2016*. IEEE Computer Society, 68–72.
- 1749 [72] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving
 1750 systems. In *ICSE ’20: 42nd International Conference on Software Engineering, Seoul, South Korea*. 359–371.
- 1751 [73] Sirasak Tejpit, Imre Horváth, and Zoltán Rusák. The state of framework development for implementing reasoning
 1752 mechanisms in smart cyber-physical systems: A literature review. *Journal of Computational Design and Engineering* 6,
 1753 4 (04 2019), 527–541. [arXiv:https://academic.oup.com/jcde/article-pdf/6/4/527/33135124/jcde.2019.04.002.pdf](https://academic.oup.com/jcde/article-pdf/6/4/527/33135124/jcde.2019.04.002.pdf)
- 1754 [74] Martin Törngren and Ulf Sellgren. *Complexity Challenges in Development of Cyber-Physical Systems*. Springer
 1755 International Publishing, Cham, 478–503.
- 1756 [75] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar T. Devanbu, and Vladimir Filkov. Quality and productivity
 1757 outcomes relating to continuous integration in GitHub. In *ESEC/SIGSOFT FSE*. ACM, 805–816.
- 1758 [76] Mario Linares Vásquez, Gabriele Bavota, Michele Tufano, Kevin Moran, Massimiliano Di Penta, Christopher Vendome,
 1759 Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Enabling mutation testing for Android apps. In *Proceedings of the
 1760 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8,
 1761 2017*. 233–244. <https://doi.org/10.1145/3106237.3106275>
- 1762 [77] Carmine Vassallo, Sebastian Proksch, Harald Gall, and Massimiliano Di Penta. Automated Reporting of Anti-Patterns
 1763 and Decay in Continuous Integration. In *Proceedings of the 41st International Conference on Software Engineering, ICSE
 1764 2019, Montreal, Canada, May 25 - 31, 2019*. IEEE.
- 1765 [78] Carmine Vassallo, Sebastian Proksch, Anna Jancso, Harald C Gall, and Massimiliano Di Penta. Configuration smells in
 1766 continuous delivery pipelines: a linter and a six-month study on GitLab. In *Proceedings of the 28th ACM Joint Meeting
 1767 on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 327–337.
- 1768 [79] Carmine Vassallo, Fiorella Zampetti, Daniele Romano, Moritz Beller, Annibale Panichella, Massimiliano Di Penta, and
 1769 Andy Zaidman. Continuous Delivery Practices in a Large Financial Organization. In *32nd IEEE International Conference
 1770 on Software Maintenance and Evolution (ICSME)*. 41–50.
- 1771 [80] Chunhui Wang, Fabrizio Pastore, and Lionel C. Briand. Oracles for Testing Software Timeliness with Uncertainty.
 1772 *ACM Trans. Softw. Eng. Methodol.* 28, 1 (2019), 1:1–1:30. <https://doi.org/10.1145/3280987>

- 1765 [81] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei Sui. An exploratory study of autopilot software
1766 bugs in unmanned aerial vehicles. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and*
1767 *Symposium on the Foundations of Software Engineering*. 20–31.
- 1768 [82] Rajaa Vikhram Yohanandhan, Rajvikram Madurai Elavarasan, Premkumar Manoharan, and Lucian Mihet-Popa. Cyber-
1769 Physical Power System (CPPS): A Review on Modeling, Simulation, and Analysis With Cyber Security Applications.
IEEE Access 8 (2020), 151019–151064.
- 1770 [83] Fiorella Zampetti, Saghan Mudbhari, Venera Arnaoudova, Massimiliano Di Penta, Sebastiano Panichella, and Giuliano
1771 Antoniol. Using code reviews to automatically configure static analysis tools. *Empir. Softw. Eng.* 27, 1 (2022), 28.
1772 <https://doi.org/10.1007/s10664-021-10076-4>
- 1773 [84] Fiorella Zampetti, Damian A. Tamburri, Sebastiano Panichella, Annibale Panichella, Gerardo Canfora, and Massimiliano
1774 Di Penta. *Continuous Integration and Delivery Practices for Cyber-Physical Systems: An Interview-Based Study - Online*
Dataset. <https://doi.org/10.5281/zenodo.6337040>
- 1775 [85] Fiorella Zampetti, Carmine Vassallo, Sebastiano Panichella, Gerardo Canfora, Harald C. Gall, and Massimiliano Di
1776 Penta. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering* 25, 2
1777 (2020), 1095–1135.
- 1778 [86] Eleni Zapridou, Ezio Bartocci, and Panagiotis Katsaros. Runtime Verification of Autonomous Driving Systems in
1779 CARLA. In *Runtime Verification*. Springer International Publishing, Cham, 172–183.
- 1780 [87] Peilun Zhang, Yanjie Jiang, Anjiang Wei, Victoria Stodden, Darko Marinov, and August Shi. Domain-Specific Fixes for
1781 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. 50–61.
- 1782 [88] Sai Zhang, Darioush Jalali, Jochen Wuttke, Kıvanç Muşlu, Wing Lam, Michael D Ernst, and David Notkin. Empirically
1783 revisiting the test independence assumption. In *Proceedings of the 2014 International Symposium on Software Testing*
and Analysis. 385–396.
- 1784 [89] Celal Ziftci and Diego Cavalcanti. De-Flake Your Tests : Automatically Locating Root Causes of Flaky Tests in Code At
1785 Google. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 736–745.
- 1786 [90] Behrouz Zolfaghari, Reza M. Parizi, Gautam Srivastava, and Yoseph Hailemariam. Root causing, detecting, and fixing
1787 flaky tests: State of the art and future roadmap. *Softw. Pract. Exp.* 51, 5 (2021), 851–867. <https://doi.org/10.1002/spe.2929>
- 1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813

APPENDIX - DETAILS ABOUT THE INTERVIEW PARTICIPANTS

The interviews' transcripts together with the labeling procedure helped us in forming organization profiles, focusing more on how the interviewed companies set and maintain a pipeline for CPS development.

In the following, we detail 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, for each organization participating in the interviews, we describe—by leveraging the codes elicited during the interviews' transcripts analysis phase—the CPS development process and especially the adoption of CI/CD pipelines and, in general, of build automation³.

O₁ (Aerospace)

CONTEXT: O₁ is involved in verification and validation (V&V) tasks for aerospace software (i.e., on-board software for satellites), hence their CI/CD pipeline is only for V&V and not for development. This is because, due to the safety integrity level (SIL) of the CPS, the development and the V&V teams and pipelines must be kept distinct [22]. O₁ relies on conventional programming languages dictated by standards in the aerospace domain (“*We mainly use ANSI C-99 following the MISRA rules*”). This implies the need for certifying software (i.e., following the Motor Industry Software Reliability Association (MISRA) standards [1, 7]).

PIPELINE STATUS: O₁ 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*”). O₁ does not rely on build matrices with jobs related to different environment variants since “*the pipeline does not have to change/evolve based on the changes in the technologies being used (version for compilers and or programming languages), the aerospace domain follows the waterfall process. So everything is [frozen]: no changes may occur later on in the process.*”

AUTOMATED TASKS: 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., “*we also have non-functional requirements expressed in terms of rules available in SonarQube*”) mainly related to the presence of duplicated code, and (iii) identifying bugs as soon as they are introduced, and executing unit and robustness tests 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 for running test suites requiring a long time to complete. It is important to note that 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*”). Finally, O₁ 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).

HiL AND SIMULATORS: O₁ cannot involve HiL in the pipeline, as it would require a clean room not accessible from the outside. Instead, it relies on third-party simulators dictated by the customer. This

³The interested reader could find the code mind maps in the replication package [84].

1863 is because the customer follows “a framework for simulation aimed at hosting different simulators
1864 for different satellite models for the digital twin of the satellite”. Relying on third-party simulators
1865 helps in reducing the costs/efforts needed to develop the simulators from scratch, as well as, helps
1866 in guaranteeing the trustworthiness of the outcome being produced and provided to the customer.
1867 Of course, the level of trustworthiness increases for those cases where the simulator is provided by
1868 the same vendor of the hardware device that must be simulated.

1869

1870

1871 O₂ (Healthcare)

1872 CONTEXT: O₂ is a large organization involved in the healthcare domain: it provides Computed
1873 Tomography (CT) scanners for clinical use. As regards the development process, O₂ has a team for
1874 each component being developed —around 17 different teams working on 70 branches— together
1875 with an integration branch where all the other branches are integrated into a “single joined point”.
1876 Furthermore, each team adopts conventional programming languages, *i.e.*, mainly C# and C++.

1877 PIPELINE STATUS: O₂ already has a CI/CD pipeline in place for CPS development that has been
1878 introduced 4 years ago, and they are still improving it. Furthermore, based on its application
1879 domain, O₂ is constrained to “follow medical application frameworks providing a base set of rules
1880 in terms of how to build applications and how to integrate them”. The latter requires the adoption
1881 of processes aimed at verifying whether or not the overall development process adheres to the
1882 regulatory standards for developing medical applications.

1883 AUTOMATED TASKS: O₂ adopts both incremental and nightly builds. Of course, the tasks involved in
1884 the different types of builds, as well as, the execution environment involved in them vary. Specifically,
1885 nightly builds leverage HiL, and run three different types of testing, namely unit/component, sub-
1886 system, and system testing. To provide developers fast feedback about the impact of their changes,
1887 O₂ relies on incremental builds executing only a subset of the whole set of functional tests — by
1888 doing “impact based testing to figure out the impact of the changes and select the tests to be executed
1889 based on the impact.” To control the overall build execution time, O₂ encourages developers to push
1890 small changes leading to “small sets of tests to be executed.” Finally, both incremental and nightly
1891 builds run static code analysis tools mainly aimed at identifying maintainability and security flows
1892 in the code.

1893 There is a specific type of build aimed at checking performance requirements like “test whether
1894 each component (some components) stays within the resource limits they are assigned to”. The outcome
1895 of such a build is compared over time to identify and monitor possible performance degradation
1896 within the whole system. Moreover, O₂ has a specific DevOps team for checking the fulfillment of
1897 security requirements, even if this is not done continuously while only “near the finalization of the
1898 product”, and it is not automated.

1899 HiL AND SIMULATORS: As explained above, both the triggering strategies adopted by O₂ and the
1900 tasks being automatized within each type of build influence the choice between using simulators
1901 and/or HiL. Nightly builds have an automated deployment on a “real” CT Scanner “without reusing
1902 existing artifacts while building all of them from scratch in a clean environment”, for executing the
1903 whole test suite in a real production environment. Note that, when talking about “real” CT Scanner,
1904 O₂ refers to “physical systems that are equivalent to the real hardware in the CT Scanner but not
1905 connected to anything around it which has a simulator running on it”.

1906 For what concerns simulators, O₂ relies on self-developed simulators — there are suitable knowl-
1907 edge and skills to properly develop simulators, *i.e.*, O₂ develops both the software and the hardware.
1908 However, at the moment, O₂ does not use simulators (“mainly used for functional testing only”) for
1909 checking non-functional (*i.e.*, performance) requirements.

1910

1911

1912 **O₃ (Acoustic Sensors)**

1913 CONTEXT: O₃ is involved in CPS innovation for the industry, among others the development of the
 1914 SPL Noise Meter Board, *i.e.*, a low-cost, high quality, electronic sensing board capable of measuring
 1915 noise of the environment. It does not have any separation of roles between the members of the team
 1916 (“*The team is the company.*”), however, the team is composed of both software and hardware experts
 1917 who work together, simplifying the overall development process, in particular for those activities
 1918 requiring the integration and communication between software and hardware components (“*Useful*
 1919 *for sensors’ integration... [it] help[s] having knowledge about the hardware components, how they*
 1920 *work and how it is possible to communicate with them.*”)

1921 O₃ adopts a pull-request (PR) development process with one branch per feature (*i.e.*, “*Several*
 1922 *branches for maintaining and developing different features.*”) Furthermore, even if it does not have
 1923 strict guidelines in terms of coding standards, O₃ attempts to adopt similar coding styles within
 1924 each branch. Finally, it relies on conventional programming languages, *i.e.*, Python for testing and
 1925 C, C++ for micro-controllers development.

1926 PIPELINE STATUS: At the moment, O₃ does not have a CI/CD pipeline for CPS development.

1927 AUTOMATED TASKS: Even if O₃ does not have a CI/CD pipeline in place, the deployment is
 1928 fully-automated, while the testing is still manual mainly due to the impossibility of automating the
 1929 oracle specification, in particular for testing acoustic signals. Furthermore, even if O₃ does not have
 1930 certification constraints for the developed code, they need to cope with certification constraints
 1931 “*for the acoustic signals.*”

1932 HiL AND SIMULATORS: O₃ only uses real hardware devices, even if within the organization there is
 1933 the wish of including simulators in the process to test the acoustic signal (*i.e.*, the main outcome of
 1934 their product) in a controlled environment, *i.e.*, “*removing noise from the surrounded environment.*”

1936 **O₄ (Robotics)**

1937 CONTEXT: O₄ is involved in the development of autonomous robots, and is made up of several
 1938 development teams where each team accounts for both hardware and software developers. Fur-
 1939 thermore, it adopts a PR development process with one branch per feature (*i.e.*, “*We have a branch*
 1940 *for each feature that needs to be implemented and/or improved and we use PRs to merge the work in*
 1941 *the stable release branch.*”) Based on the application domain, it mainly adopts C++, together with
 1942 Python for users’ interfaces and for interacting with the hardware devices.

1943 PIPELINE STATUS: O₄ has a fully containerized (using Docker) pipeline for CPS development.
 1944 It relies on continuous and nightly builds, even if they are not used for running time-intensive
 1945 tasks, *i.e.*, “*(that is not so expensive in terms of execution time)*”, while for running regression testing
 1946 activities on already packaged components and for deployment to the customers. Furthermore, the
 1947 CI/CD configuration is pretty stable meaning that, even if each branch may rely on a customized
 1948 CI/CD process, the configuration does not have to change over time.

1949 AUTOMATED TASKS: Our interviewee mentions the execution of static code analysis tools to
 1950 inform developers about code quality degradation, and unit tests relying on simulators. Only when
 1951 a PR is peer-reviewed and there are no failures in the entailed CI/CD process, it is possible to merge
 1952 the change on the stable repository and enact a release process for shipping the product to the
 1953 customers. O₄ monitors the overall quality of the development process in terms of static analysis
 1954 metrics and code coverage from the unit test execution. The application domain does not introduce
 1955 certification constraints, *i.e.*, “*If you want to sell a robot you do not need to have a certified robot*”,
 1956 while it hinders the automation of non-functional testing within the pipeline. Specifically, our
 1957 interviewee mentions the manual execution of reliability and safety tests “*running the robot a long*
 1958
 1959
 1960

1961 *time with a guy supervising the test execution to understand when and why the robot starts to not*
1962 *work anymore”, or “guaranteeing that once pressing the stop button the robot actually shuts down”.*

1963 HiL AND SIMULATORS: O₄ relies on third-party simulators and HiL. One point raised by our
1964 interviewee is related to the partial usage of Docker on the hardware so that it is possible to run
1965 the robot in a privileged mode and switch between software versions quite easily: “*each one may*
1966 *choose the version of the software that has to be run over the robot.*”

1968 **O₅ (Automotive)**

1969 CONTEXT: O₅ is a large company operating in the automotive domain working on the software-
1970 focused driving platform. The development process is organized into three different teams each
1971 one with a specific goal: “*one working on virtual machines, one working on web services because we*
1972 *provide DevOps solutions for embedded systems, and finally, we have a small team working on customer*
1973 *delivery with the goal of adapting our tools to the customers’ needs.*” This is the only organization in
1974 our study relying on real-time languages, *i.e.*, real-time Java to cope with scheduling requirements
1975 of embedded systems.

1976 PIPELINE STATUS: O₅ already has a CI/CD pipeline in place mainly for deployment purposes (“*we*
1977 *are able to support updating software on devices on the fly*”), even if it is working on improving it —
1978 “*it is still kind of an infancy we are still working on improving*”.

1979 AUTOMATED TASKS: O₅ mainly uses the pipeline for deployment. However, differently from other
1980 organizations, O₅ relies on virtual machines instead of using containers for several reasons: (i)
1981 better control over the resources (*i.e.*, “*the ability to enforce our resource usage inside the virtual*
1982 *machine while you do not have quite the same extent with a container*”), (ii) versioning capability, *i.e.*,
1983 “*when a new service comes in, you register it so it is easy to start a new version of this service, run down*
1984 *the old one and switch over the new one during run-time*”, and (iii) memory safety guarantee, *i.e.*,
1985 “*by looking at a recent post by both Google and Microsoft we found that around 70% of the security*
1986 *violations are due to failures of memory safety. So by using a garbage-collected environment, we can*
1987 *prevent those issues from occurring.*” Going deeper into how the deployment process works, O₅ first
1988 creates the virtual machine, (*i.e.*, emulating the virtual environment), then the OSGi infrastructure,
1989 and finally it tests individual modules. The latter means that O₅ does not test all the developed
1990 modules together since it “*deploy[s] individual bundles to a platform.*”

1991 For what concerns the verification of non-functional requirements, O₅ performs security and
1992 performance testing, even if they are not included in the pipeline. Specifically, for real-time systems
1993 it is important to monitor the impact of each change on performance properties to be able to
1994 identify, as soon as possible, the change introducing performance degradation, *i.e.*, “*we have various*
1995 *performance tests that we run regularly to track our performance as the system evolves.*”

1996 HiL AND SIMULATORS: Since O₅ develops software for embedded entertainment in the automotive
1997 domain, the HiL is only available for a final validation on the customer’s side: “*then employ our*
1998 *customer for the last mile*”, so most of the work is done relying on virtual environments.

2000 **O₆ (Aerospace)**

2001 CONTEXT: Similar to O₁, O₆ operates in the aerospace domain, and is mainly involved in the
2002 development and refining of the routing algorithm for the Free Route Airspace (FRA). For what
2003 concerns the programming language being adopted, O₆ relies on conventional languages: “*C and*
2004 *C++ [are] used for the back-end.*”

2005 PIPELINE STATUS: O₆ already has a CI/CD pipeline mainly for deployment and testing purposes,
2006 that is under continuous improvements. Moreover, our interviewee mentions that the pipeline is
2007 more an MLOps than a simple DevOps pipeline.

2010 AUTOMATED TASKS: Among the phases being automated there are: (i) static code analysis for
 2011 identifying maintainability flows and spotting bugs as soon as they are introduced, (ii) unit testing,
 2012 (iii) integration testing, and (iv) deployment. Furthermore, the execution of non-functional testing
 2013 activities is mainly carried out manually and outside the pipeline, due to the high complexity of the
 2014 real-time operating system under development. Similarly to O_1 , it is required that the developed
 2015 code satisfies strict certification requirements that are mainly checked by relying on code coverage
 2016 tools. Differently from other organizations, O_6 does not rely on nightly builds, meaning that also
 2017 time-intensive tasks are executed at each change, *i.e.*, “*even the slow builds are continuously built.*”
 2018 O_6 recommends developers to use private builds before pushing their changes on the stable release
 2019 branch, at least for what concerns the execution of unit testing. Finally, the pipeline provides
 2020 a monitoring mechanism for what concerns aspects of the real-time operating system such as
 2021 scheduling and memory that “*gives us the possibility to collect feedback/evidence that may help us in*
 2022 *obtaining the certifications.*”

2023 HiL AND SIMULATORS: O_6 relies on both simulators and HiL, however it does “*not have simulators*
 2024 *and HiL in the same pipeline mostly for certification issues.*” Specifically, it is possible to rely on real
 2025 devices only when there is enough trustworthiness about the software in terms of correct behavior,
 2026 as well as the absence of crashes gained by relying on self-developed simulators.

2027 O_7 (Railways)

2029 CONTEXT: O_7 is involved in delivering software for railways, *i.e.*, Train Control Management System
 2030 (TCMS), and similarly to what is reported for the aerospace domain, due to the safety integrity
 2031 level of the software under development, developers and testers must be different (*i.e.*, “*Testers*
 2032 *and Developers are in separate teams in presence of new functionality to be implemented both start*
 2033 *together to implement and write test cases.*”)

2034 PIPELINE STATUS: O_7 already has a CI/CD pipeline in place for CPS development – “*introduced*
 2035 *two years ago*” – that, at the moment, is in a continuous improvement state since it does not
 2036 automatize the whole development process (*i.e.*, “*The deployment on the real train or on the hardware*
 2037 *test track is not automated at the moment even if we are working on making it automatic.*”) In terms of
 2038 programming language, the interviewee mentions the need of adapting the programming language
 2039 to the device on which the software has to be executed, however, they mainly rely on conventional
 2040 languages.

2041 Based on the application domain, O_7 adopts staged builds following the “green-build rule”. In the
 2042 first stage, the build process is executed on a virtual machine, *i.e.*, “[*a virtual train, software running*
 2043 *on a PC that should behave like it does on a real train*”. Once a change occurs on a specific component
 2044 the related build process is enacted and, in presence of a green status, all the components are
 2045 deployed together so that it is possible to enable the execution using the virtual train (“*the devices*
 2046 *are run in some kind of containers and we have frameworks building and connecting the whole set of*
 2047 *devices and components.*”). If the build process ends with a successful state, it is possible to move to
 2048 the next stage that relies on the hardware test track – *i.e.*, “*where we have the whole set of devices*
 2049 *and even some more that we do not have in the virtual train.*” Finally, if the build process for the
 2050 second stage ends with a green status, it is possible to run the last stage relying on a real train.
 2051 Note that, each device/component has a proper CI/CD configuration.

2052 AUTOMATED TASKS: O_7 uses the pipeline to automatically test basic functionality (*i.e.*, “*We test*
 2053 *specific train functionality such as whether we should activate the train in [a specific] mode*”), as well
 2054 as, the interaction between different components/devices (*i.e.*, “*we have a long sequence of events for*
 2055 *each test that involves different devices and components so we are mainly doing integration testing.*”)
 2056 The test suites used in different stages of the build process may be different since “*for some test*
 2057 *cases, we are not allowed to rely on the virtual environment while we must consider the hardware track*
 2058

2059 *or a real train.*” At the moment, O₇ has automated deployment within the pipeline only for the first
2060 stage, *i.e.*, relying on the virtual train (for which the overall build execution is “*around one hour and*
2061 *[a] half*”), while it is done manually for what concerns the other two stages: hardware test track
2062 and a real train. Testing against non-functional requirements is also done manually, because of the
2063 high variability and complexity of the environment.

2064 Going deeper into how developers interact with the CI/CD pipeline, O₇ enforces developers to
2065 run private builds before pushing their changes on the main stable repository. The private builds
2066 are aimed at executing the same test suite later executed on the CI/CD servers — “*for the moment we*
2067 *cannot configure the number and type of tests to be executed locally*”. Furthermore, the “green-build
2068 rule” is used for determining the development tasks: “*in presence of a failure all developers are*
2069 *stopped until the build becomes green again.*”

2070 HiL AND SIMULATORS: O₇ adopts both simulators and HiL in different stages of the build process,
2071 with the use of HiL occurring only in the last stage of the pipeline.

2072

2073 O₈ (Railways)

2074 CONTEXT: The application domain of O₈ is railways, and in particular the development of a specific
2075 component used for transmitting data between on-board and ground applications. O₈ uses C and
2076 C++ (*i.e.*, conventional programming languages), and it has strict constraints for what concerns
2077 the production code that has to satisfy strict certification requirements, as well as the compliance
2078 with the railway standards and specifications. The latter is mainly checked by relying on “*a specific*
2079 *complex tool that can be configured based on specifications and standards.*”

2080 PIPELINE STATUS: Due to the limited availability of human resources together with the complexity
2081 and the safety integrity level of the application domain, O₈ does not have a CI/CD pipeline in place
2082 for CPS development and it has, in general, little automation in the development process.

2083 AUTOMATED TASKS: Only the adherence to standards and specifications is automated, while
2084 functional “*tests are written manually starting from requirements and system specification[s] but also*
2085 *their execution requires a manual effort*”. This is also the case for non-functional and integration
2086 testing (*i.e.*, “*we have a set of testers in front of a screen who monitor and check for the presence of any*
2087 *discrepancies about what is expected and what is instead observed while running the system.*”) Due to
2088 the effort and time needed to manually verify the reliability of the software under test, functional
2089 tests are executed at every change, while integration tests are only executed when the change
2090 impacts the “*interfaces with other modules/components.*”

2091 HiL AND SIMULATORS: Due to the high cost of the hardware devices involved in this particular
2092 application domain, O₈ mainly relies on simulators that are self-developed (“*we do not rely on third*
2093 *party very expensive simulators*”). However, once per week, O₈ performs a testing session with a
2094 real “*train running in a real environment with real traffic [and] possibly without people.*”

2095

2096 O₉ (Identification Technology)

2097 CONTEXT: O₉ is involved in “*develop[ing] software relying on identification technologies such as*
2098 *RFID [(Radio Frequency IDentification),] Bluetooth low energy or bar codes*, other than mobile app
2099 development for which there is a CI/CD pipeline used for testing and automated deployment on
2100 the play stores. For what concerns the CPS development, O₉ relies on conventional programming
2101 languages such as C# and Java.

2102 PIPELINE STATUS: The limited availability of human resources, together with a lack of culture
2103 for setting a pipeline dealing with sensors and actuators, results in not having a CI/CD pipeline in
2104 place for CPS development.

2105 AUTOMATED TASKS: The testing phases are almost fully automated. Specifically, there are “*RFID-*
2106 *readers connected to a network*” on which it is possible to execute unit and integration testing

2107

activities automatically. For what concerns integration testing, 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, O₉ does not run the whole test suite at each change. Instead, they manually select some test cases based on impact analysis: “*select what are the test cases that are impacted by the change that, consequently, need to be executed*”. Other than having unit and integration testing activities, O₉ also executes, from time to time, performance testing.

For what concerns the deployment of CPS-related software, O₉ relies on Docker for creating images that are manually deployed onto the servers.

Finally, the development process also features a monitoring component for the internal development platform and customers’ devices, to notify about anomalies and errors, as soon as they occur.

HiL AND SIMULATORS: The development process adopted by O₉ relies on both (self-developed) simulators and HiL. Simulators are developed based on specific organization needs and use case scenarios, implying that they are limited in their functionality.

O₁₀ (Energy)

CONTEXT: O₁₀ is involved in the development of prototypes and proof of concepts for the energy domain. The development of prototypes rather than real products represents concrete facilitation, since there may be less stringent constraints in terms of pipeline setting and evolution.

PIPELINE STATUS: What is mentioned above 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. The pipeline configuration is pretty stable probably due to the development of prototyping solutions that do not need to be shipped to real environments.

AUTOMATED TASKS: Other than having a compilation phase, the CI/CD pipeline is aimed at executing unit and integration tests (“*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 for 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, O₁₀ does not rely on nightly builds. It only uses incremental builds so that each build execution time does not overcome the 10 minutes rule.

HiL AND SIMULATORS: Looking at the execution environment, O₁₀ 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, O₁₀, 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, O₁₀ uses Docker images for checking the correct behavior over the real devices, as well.