COSMOS

DevOps for Complex Cyber-physical Systems

Project Number 957254

D3.1 Report of practices collected through interviews and questionnaires

Version 3.2
5 July 2021
Final

EC Distribution

University of Sannio and Zurich University of Applied Sciences

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

PROJECT PARTNER CONTACT INFORMATION

Aicas James Hunt Emmy-Noether-Strasse 9 76131 Karlsruhe Germany Tel: +49 721 663 968 0 E-mail: jjh@aicas.com	**Delft University of Technology** Annibale Panichella Van Mourik Broekmanweg 6 2628 XE Delft Netherlands Tel: +31 15 27 89306 E-mail: a.panichella@tudelft.nl
Intelligentia Davide De Pasquale Via Del Pomerio 7 82100 Benevento Italy Tel: +39 0824 1774728 E-mail: davide.depasquale@intelligentia.it	**GMV Skysoft** José Neves Alameda dos Oceanos Nº 115 1990-392 Lisbon Portugal Tel. +351 21 382 93 66 E-mail: jose.neves@gmv.com
Q-media Petr Novobilsky Pocernicka 272/96 108 00 Prague Czech Republic Tel: +420 296 411 480 E-mail: pno@qma.cz	**Siemens** Birthe Boehm Guenther-Scharowsky-Strasse 1 91058 Erlangen Germany Tel: +49 69 797 932850 E-mail: birthe.boehm@siemens.com
Siemens Healthineers David Malgiaritta Siemensstrasse 3 91301 Forchheim Germany Tel: +49 9191 180 E-mail: david.malgiaritta@siemens-healthineers.com	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of Sannio Massimiliano Di Penta Palazzo ex Poste, Via Traiano I-82100 Benevento Italy Tel: +39 0824 305536 E-mail: dipenta@unisannio.it	**University of Luxembourg** Domenico Bianculli 29 Avenue J. F. Kennedy L-1855 Luxembourg Luxembourg Tel: +352 46 66 44 5328 E-mail: domenico.bianculli@uni.lu
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	**Zurich University of Applied Sciences** Sebastiano Panichella Gertrudstrasse 15 8401 Winterthur Switzerland Tel: +41 58 934 41 56 E-mail: panc@zhaw.ch

DOCUMENT CONTROL

Version	Status	Date
1.0	Draft with guidelines and questions for interviews	22 February 2021
2.0	Modified structures based on R&D comments	16 March 2021
3.0	Deliverable ready for partner reviews	28 June 2021
3.1	Updates following first review	1 July 2021
3.2	Further updates following second review	5 July 2021

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This deliverable provides an analysis of the CI/CD practices collected by interviewing experts from nine different companies. The interview structure and protocol are first described, followed by the main results that have been gathered and summarised including the CI/CD practices, as well as challenges and barriers in applying CI/CD in a CPS context. Included in the appendices are the detailed questionnaire that was used, as well as the transcripts of the interviews with each company representative.

1. INTRODUCTION

1.1 WORK PACKAGE OVERVIEW

The overall goal of WP3 is to distil a methodology for setting-up and maintaining a DevOps pipeline for CPS, based on stakeholders' needs and on the technical approaches conducted in WP4-WP6. More specifically, we:

- Gather information about the current practice of CI/CD for CPS, as well as of challenges DevOps are encountering in this context;
- Distill a general methodology to be followed when setting-up and maintaining a DevOps pipeline for CPS;
- Identify a catalog of good and bad practices to follow/avoid when adopting DevOps for CPS, and provide a support to automatically detect bad practices and support DevOps in their resolution;
- Optimize the CI/CD pipeline, by means of prioritizing builds and tests through statistical techniques, and optimizing the use of simulators and HiL, while coping with security and privacy issues.

1.2 TASK OVERVIEW

The first step towards setting up a methodology for CPS development consists of gathering information about practices, challenges, and barriers related to CI/CD adoption for CPS. Our aim would be to identify (i) what industrial stakeholders have, so far, considered to be good principles and bad practices in CPS development, (ii) what kinds of benefits stakeholders wish to achieve when setting up a CI/CD pipeline for CPS; (iii) what are the challenges and barriers they are facing, that impede the adoption of CI/CD in their context. The analysis will be performed by combining semi-structured interviews with survey questionnaires (designed based on the results of the interview).

The output of the task will be a report on bad practices, challenges, and barriers in the adoption of CI/CD for CPS development.

1.3 PURPOSE OF THIS DELIVERABLE

This deliverable is structured in two main sections:

- The structure of the interviews;
- A summary of the CI/CD practices, as well as challenges and barriers in applying CI/CD in a CPS context, gathered through the conducted interviews.

This deliverable poses the basis for the next deliverable D3.2, which, using grounded theory, aims at eliciting a model of CI/CD practices in the context of CPS. Such practices will then be validated through a survey.

2. INTERVIEW'S STRUCTURE AND QUESTIONS

In this section we report on the structure of the interviews together with questions that have been asked. It is important to remark that, in defining the overall structure of the interview, we have started by looking at structures already being used in previous

literature by Hilton et al. [1] and Zampetti et al. [2]. Such studies are also considered as a knowledge baseline for what concerns the application of CI/CD in general (but not in the context of CPS).

More in detail, the interview is composed of two main sections. In the first one, we collect demographic information about our participants in terms of development experiences, domain of the company and their role in the company. After that, we start by considering different aspects of a CI/CD pipeline focusing the attention of properties and characteristics that are peculiar in CPS domain. For instance, we perform a deeper investigation about the inclusion of hardware components as well as of simulators. The full questionnaire used for the interviews is provided in Appendix A of this document.

Finally, in terms of how to conduct the interview, it is important to state that it is not mandatory to ask all the questions reported in the following. The questions are intended as guidance for the interviewer while conducting the interview to ensure topics are properly investigated in case some aspects are left uncovered by the interviewee.

3. INTERVIEW PROTOCOL AND INSTRUMENTS

Potential participants to the interviews (both internal to the COSMOS consortium and external) have been contacted beforehand via email, explaining the general purpose of the interview, so that they have enough elements to be prepared for it. A meeting time was then agreed.

Interviews have been conducted online through WebEx™ or Microsoft Teams™ and recorded. The interview has been conducted by one or two people (interviewers). Before starting, one of the interviewers reminded the general goal of the interview, asking the participant to refer to actual/intended CI/CD pipelines in the context of CPS. Also, the interviewer asked for consent to start the registration. Each interview lasted about 45 min. During the interview, the interviewer(s) followed the interview structure; however, they deviated from the structure based on the answers collected. For example, they skipped interview portions that did not apply to the current case, or analysed more in-depth topics that were particularly relevant for a given case.

After the interview was concluded, the registration was downloaded and transcribed in a textual document by one of the interviewers. The transcripts are provided in Appendix B of this document.

The summarized content of these transcripts constitutes the main content of this deliverable, and poses the basis for the definition of the taxonomy, which will be reported in D3.2.

4. ANALYSIS OF THE INTERVIEWS' RESULTS

In the following, we report the main results we collected in the interview. First, we report an overview of the interviewed companies (whose names are anonymised in this report). Then, we describe the CI/CD pipeline they have in place, detailing: (i) the technology being used; and (ii) the pipeline structure. After that, we perform an in-depth analysis of peculiar aspects of the pipeline related to: (i) verification & validation; (ii) deployment; (iii) presence of Artificial Intelligence components; (iv) usage of

simulators and HiL (Hardware-in-the-Loop); and (v) pipeline configuration. Finally, we conclude by outlining challenges and barriers in setting up a CI/CD pipeline for CPS.

4.1 COMPANIES' OVERVIEW

Table 1 lists the nine companies we have interviewed to identify the practices being used in different CPS domains while setting and using a CI/CD pipeline. As reported in Table 1, the companies vary in terms of size; indeed, we have:

- 2 micro-sized companies (C8 and C9), i.e., with less than 10 employees in total;
- 2 small companies (C1 and C3), i.e., between 10 and 20 employees;
- 1 medium-size company (C4), i.e., between 20 and 100 employees;
- 4 large companies (C2, C5, C6 and C7), i.e., more than 100 employees.

Looking at Table 1, it is possible to note that the nine companies vary also in terms of CPS domains. Moreover, while six out of nine companies have only a single CPS domain in which they are working, C1, C2 and C7 cover different domains.

Finally, the last column in Table 1 reports the mean developer numbers of developers in the team to which the interviewee belongs. Specifically, most of the interviewees work in small teams made up of less than ten developers, with C1 having no separation in terms of teams (i.e., "the company is the team"), while C6 and C7 counting 40 and 200 developers in the team respectively.

Table 1 Companies Description in terms of size, domain, team size and role within COSMOS

Company	Size	Domain	Teams
C1	Small	Acoustic sensors, Energy devices for industrial environments and smart cities	No separation
C2	Large	Avionics, Aerospace, Defense (the company covers other domains too)	Team composed of 4 people
C3	Small	Aerospace (the company covers other domains too)	Team composed of 3 people
C4	Medium	Robotics	Team composed of 5-10 people
C5	Large	Railways	Team composed of 50 people
C6	Large	Healthcare	Team composed of 200 people
C7	Large	Automotive and Industrial Automation (the company covers other domains too)	Team composed of 40 people
C8	Micro	Railways	Team composed of 4-5 people
C9	Micro	Software based on identification technologies (e.g., RFID)	Team composed of 3 people

It is important to highlight that, for two out of nine companies, we have interviewed more than one person actively involved in the CI/CD pipeline setting. Specifically, we

interviewed 12 people among nine companies. Among them, we have 4 project managers, 3 team leaders, 2 software/hardware integrators, 2 software engineer and 1 DevOps Architect. Furthermore, 7 out of 12 interviewees are directly/indirectly involved in the CI/CD process setting while the remaining ones simply use the CI/CD pipeline being set.

4.2 CI/CD PIPELINE TECHNOLOGIES USED

Table 2 reports the set of tools and frameworks used by the companies involved in the semi-structured interviews focusing on their CPS development process. The attention has been focused on the (i) main programming languages, (ii) versioning systems, (iii) CI/CD frameworks, (iv) build automation frameworks, and (v) whether or not the company uses Docker containers.

Since the focus is on CPS, seven out of nine interviewees rely on C and/or C++ for the development process. The other two companies, C7 and C9, use different programming paradigm due to their operation domain. Specifically, C7 relies on Real-Time Java since its focus is providing software for embedded operating systems, such as QNX and VxWorks. Instead, C9 mainly develops software relying on identification technologies such as RFID (Radio Frequency Identification). Furthermore, by looking at Table 2, it is evident that others programming languages are used mainly for front-end development and user interfaces development. For C2, there is a difference between the programming languages adopted for development purposes (C and C++) and the ones used for testing and configuration purposes (Python). Finally, for C5, there is the adoption of domain-specific languages (DSLs) depending on the hardware devices with which the software has to interact.

By looking at the versioning systems being used, six out of nine companies rely on Git, while C3 and C6 use different systems mainly due to the operational domain or the environment. Finally, for C5, there is not an explicit mention of the versioning system being adopted within the company.

Table 2 Tools and Frameworks used for CPS development

Company	Programming Languages	Versioning Systems	CI/CD framework	Build Automation Framework	Use of containers
C1	C and C++ (development); Python for testing and configuration	Git and GitLab		Make	Not for CPS
C2	C and C++ for back-end development; Java for front-end development	Git	Jenkins	Mainly Maven	Not for real-time operating systems
C3	ANSI C-99	SVN	Jenkins	Make	No
C4	C++; Python for user interfaces	Git	GitLab		Yes
C5	C and DSL specific to the hardware devices (e.g., PLC)		Jenkins		No
C6	C# and C++	Microsoft	Microsoft		No

Company	Programming Languages	Versioning Systems	CI/CD framework	Build Automation Framework	Use of containers
		TFS	TFS		
C7	Real-Time Java (RTJ)	Git	GitLab and Jenkins		No
C8	C and C++	Git and GitLab		Make	No
C9	C#; Java, Kotlin and Dart for mobile development	Git and GitLab			Yes

The fourth column in Table 2 reports the CI/CD framework adopted in the interviewed companies. Specifically, three companies adopt Jenkins, C4 relies on GitLab, C7 uses both Jenkins and GitLab, and C6 due to the environment relies on Microsoft technologies. Finally, it is important to remark that C1, C8, and C9 do not have yet a CI/CD pipeline for the CPS domain.

Unfortunately, five out of nine companies do not explicitly mention any framework adopted for build automation, while the other three, based on the programming language mainly rely on make files, while C2 mainly uses Maven, even if there are specific cases in which the company still uses Ant.

Finally, for what concerns the usage of containers in the CPS development process, only two out of nine companies (C4 and C9) adopt containerization. Specifically, C4 has a fully dockerized pipeline that uses both Debian packages and Docker, while C9, even if it does not have a CI/CD pipeline, creates Docker images that are manually deployed on the devices.

For what concerns C1, they use infrastructure tooling such as Docker and Kubernetes in other development processes but not for the SPL board development. The same occurs for C2 that uses Docker but not in the development process of the real-time operating system. C6, instead, has only one project running in a containerized environment with Docker. Their OS environment (i.e., Windows) does not help them with dockerization, as well as, their applications are not yet structured so that they may be easily dockerized. C7, instead, prefers to rely on virtual machines (VMs) instead of using Docker since VMs give more control over the resources being used. As an example, the interviewee mentions that it is possible to enforce the resources usage within a VM while the same is not possible within a container.

4.3 CI/CD PIPELINE STRUCTURE

Table 3 summarizes the tasks included in the CI/CD pipeline configured by the nine interviewed companies, focusing the attention on (i) static analysis, (ii) testing, focusing on both functional and non-functional tests, and (iii) deploy. Note that Table 3 only reports the information for the companies having declared to adopt a CI/CD pipeline for CPS development. Specifically, C1 aims at setting a CI/CD pipeline in the CPS context for compilation and deployment purposes (i.e., *"We would like to have it to automatically build software: check if it compiles and if it can build an image");* C8, also reports the lack of a CI/CD pipeline without explaining the reasons behind this

choice; and C9, instead, does not have a CI/CD pipeline for the CPS context mainly due to the low number of employees of the company together with a lack of knowledge about how the pipeline may interact with sensors and/or actuators. However, C9 has a CI/CD pipeline used for automatic testing and deployment of mobile applications. Finally, for what concerns C2, at the moment, they are configuring a CI/CD pipeline for the development of the real-time operating system. However, the pipeline for the avionics domain is more a MLOps pipeline instead of a traditional DevOps pipeline.

Table 3 Tasks involved the CI/CD pipeline for the CPS context in the interviewed companies

Company	Static Analysis	Testing	Deploy
C3	Yes	Yes, focusing more on functional testing at unit level	No
C4	Yes	Yes, focusing more on functional testing at unit and integration level	Yes
C5	Not sure	Yes, focusing more on functional testing at unit and integration level	Yes, but only with simulators
C6	Yes	Yes, both functional and non-functional	Yes
C7	No	No	Yes

Going deeper on the data reported in Table 3, C3 has a V&V pipeline for the aerospace domain and does not interact with the development of the software that has to be deployed over the satellites. For this reason, their pipeline mainly involves static code analysis tools for checking the adherence of the code with the MISRA[1] rules and unit testing.

For what concerns C4, their pipeline involves static code analysers, testing for functional requirements at both unit and integration level, and automated deployment. A similar configuration is adopted by C6 with only one difference: they have different CI/CD pipelines mainly used for different purposes so they are also able to automatize the execution of non-functional tests within the pipeline.

For what concerns C5, the interviewee mentions that he is not sure about the usage of any automated static code analysis tool, while for sure the company is adopting a code review process aimed at manually applying static code analysis. Furthermore, their CI/CD pipeline has functional testing for both the unit and integration level while they are not automatizing the execution of non-functional testing probably due to the lack of resources required for testing.

Finally, C7 is trying to improve the CI/CD pipeline that at the moment is mainly used for supporting the software updates on the devices on the fly. The interviewee highlights that their pipeline is mainly for deployment purposes and detached from the software development process.

[1] C/C++ coding standards developed by the Motor Industry Software Reliability Association (MISRA)- https://www.misra.org.uk

4.4 VERIFICATION & VALIDATION

For what concerns verification and validation activities done for developing CPS, we have investigated four different aspects:

- Usage of static code analysis tools for continuously checking the code quality of the system under development (e.g., maintainability issues, violations to coding guidelines or vulnerabilities);
- Functional testing at different levels (e.g., unit, integration or system testing);
- Testing non-functional requirements (e.g., performance, scalability, security);
- Testing strategies.

In the following, we will summarize the practices being adopted by our interviewees for each aspect under analysis.

4.4.1 Usage of static code analysis tools

- Two out of nine interviewees (C1 and C5) do not use any static code analysis tool, even if by adopting a Pull Request (PR) development process together with code reviews, there are reviewers trying to maintain the repository with similar coding styles. In both cases, there is a need for defining a specific set of coding guidelines that will be checked automatically.
- Four out of nine interviewees (C2, C3, C6 and C8) rely on static code analysis tools for spotting possible violations to certification standards. However, C6 customizes the rules enabled while running the static analysis by also considering specific rules for detecting technical debt in the code. It is important to highlight that, for what concerns C8, the tool used will generate compilation errors in presence of any violations in the produced code.
- C3 and C6 also use static code analysis tools for testing non-functional requirements such as security requirements. Furthermore, a different interviewee (i.e., C8) is planning to introduce static analysis for checking some of their non-functional requirements dealing with safety and security.
- One interviewee (i.e., C7) is not very satisfied with the adoption of static code analysis tools. While they are able to use them for verifying the “memory safety” they are planning to improve their usage for other purposes such as evaluating the inter-thread communication for avoiding deadlocks. As part of their future plan, there is also the inclusion of formal analysis tools that may help in spotting deadlocks within the system.
- The remaining two interviewees (C4 and C9) do not rely at all on static code analysis tools neither mention the need for including them in their CI/CD pipelines.

4.4.2 Functional Testing

- Four out of nine interviewees (C2, C4, C5, and C8) do both unit and integration testing within the CI/CD pipeline. However, C8 does not execute integration tests at every change, while they have continuous unit testing.

- C1 has both unit and integration testing mainly done manually for what concerns the software application and their interfaces with the hardware devices. The latter is mainly dictated from the application domain in which it is required to execute the tests in a controlled environment.

- C9 has automated unit and integration testing without using a CI/CD pipeline. However, there are some integration tests requiring hardware devices that, at the moment, are not automatized.

- Finally, the last interviewee (C3) due to the pipeline goal (verification pipeline and not a development pipeline), has only the unit testing within the pipeline.

4.4.3 Testing non-functional requirements

- C2 does not have the execution of non-functional tests within the pipeline due to the fact that their execution requires a huge amount of time that may result in slow builds. Furthermore, some of their non-functional requirements cannot be tested automatically. The same occurs for C5 for which we do not know the reasons that hinder them for automatizing the execution of the non-functional tests within the pipeline.

- C7 has non-functional tests executed automatically for both security and performance requirements. However, due to their expensiveness in terms of execution time, they are not executed at every change, while on nightly or weekly builds. The same behaviour is reported by C8, who, instead, executes install and performance tests.

- For what concerns the robotic domain (i.e., C4), there are no specific rules to be met for certification purposes, meaning that there are no compliance tests. The same applies to C6, mainly working on medical applications, in which it is required to follow a specific development process guaranteeing that the developed code follows the regulatory standard.

- C4 has both reliability and security testing mainly done manually with the tester supervising the test execution.

- In C6, there are different DevOps teams, each one aimed at checking a specific non-functional requirement. For instance, they mention the presence of a Sec DevOps team applying security testing manually without relying on a CI/CD pipeline.

- C1 executes performance testing not at every change due to the testing execution time and does not have specific security and safety requirements to be tested due to the application domain. Some constraints due to the application domain are highlighted also by C3, where due to the high costs needed for buying the tools

to be used (directly required by the customers), the execution of non-functional tests is left to the customers directly. The latter refers to performance, reliability and safety requirements.

- Finally, C9 has both performance and reliability testing while not considering security testing and also compliance testing because of the application domain (RFID). Moreover, their tests are not executed within the CI/CD pipeline since they do not have yet a CI/CD pipeline in place for the CPS development.

4.4.4 Testing strategies

- Two out of nine interviewees (C2 and C5) encourage developers to do unit testing in private builds before pushing their code on the main branch.

- The procedure used for testing purposes may depend on the safety integrity level (SIL) of the software under development. Two out of nine interviewees (C2 and C3) mention that the SIL dictates who can execute testing activities. Increasing the SIL results in the need for having a different team mainly aimed at testing the product that has to be different from the team who has developed it.

- All the interviewees mention manually written test cases mainly starting from the requirements.

- We did not find any evidence about the usage of mutation testing techniques.

- Finally, there are cases in which the companies select the number and type of test cases to be involved in a specific build based on impact analysis, i.e., by applying basic, change-based test case selection strategies. The latter implies that the CI/CD configuration varies based on the type of change, as well as, on the files being touched. In other words, in order to deal with the build execution time, the interviewed companies rely on incremental builds.

4.5 DEPLOYMENT

Based on the application domain of the companies, the deployment may be fully automated within the CI/CD pipeline or else is done manually. Specifically, C5 mentions that their deployment is fully automated for what concerns the deployment on the virtual environment that relies on a virtual train, while deployment is done manually on either the real train or the hardware test track. C8 is on the same line, indeed, they have both manual and automated deployment where manual deployment mainly used for deployment on the real hardware.

As concerns the robotic domain, C4 mentions relying on a partially automated deployment on the robot that aims at simplifying the switching between different software versions deployed over the robot.

C9, instead, highlights the difference between their deployment in the context of mobile development and CPS. For the former, the deployment is fully automated within the CI/CD pipeline, while for the CPS context, since they do not have yet a CI/CD pipeline they are able to containerize their application but the deployment on the device is mainly manual.

On the contrary, C1 and C6 have a fully automated deployment on their devices.

For what concerns C7, they intentionally do not rely on Docker technology for deployment while preferring the Virtual Machines solution that mimics better their microservices architecture.

Finally, two out of nine interviewees do not have deployment involved in the development process (C2 and C3).

4.6 INCLUSION OF AI CAPABILITIES WITHIN THE CPS DOMAIN

MLOps is an emerging field that is attracting attention from both academia and industry. It is mainly related to having a CI/CD pipeline dealing with software based on AI/ML algorithms. Since that software for CPS may be safety-critical, we do not expect to have many companies having AIOPS in place in our study. As expected, only C2 has more an MLOps pipeline than a classic CI/CD pipeline. However, their pipeline is not used for determining which ML model to run on the real device, or to properly calibrate the hyper-parameters of the ML model based on the field data acquired on previous execution. Their typical usage scenario is the following: based on a set of baseline results in terms of optimal route for the flight, they run the algorithm and compare the actual results with the expected ones trying to refine the model based on the comparison results.

C4, instead, for the robots, has some ML components for inspection capabilities but also for reinforcement learning for walking capabilities of the robot. However, the learning algorithm is not being tested within the CI/CD pipeline.

Finally, we also have one interviewee who uses AI/ML components mainly for monitoring purposes. Specifically, C6 uses a learning-based approach for root cause analysis with the aim of aiding developers with isolating the errors in the code with low effort and time.

4.7 HARDWARE-IN-THE-LOOP AND SIMULATORS

Table 4 reports information about the characteristics of the simulators and hardware devices which our interviewed companies rely on. Specifically, we report the type of simulators being used together with whether or not the simulators are self-written by the company but also the circumstances in which the companies rely on simulators and when, instead, relying on real hardware devices is mandatory. Finally, when available, we also report the type of discrepancies being observed by the companies in terms of outcome received while testing on simulators or on real devices.

Table 4 Simulators and real devices used by the interviewed companies

Company	Simulator characteristics	Hardware in the Loop	Notes
C1	No Simulators	Microcontrollers and devices generating input signals	No need to optimize the HiL usage
C2	Only in-house simulators for certification issues	HiL after everything works as expected on simulators	Discrepancies between HiL and simulators outcomes depend on the setting of environment

Company	Simulator characteristics	Hardware in the Loop	Notes
			parameters
C3	Simulators dictated from the customers	Not applicable due to domain-specific constraints	Limited number of simulators available
C4	Simulators for easy to simulate robot activities	Real robot for not easy to simulate scenarios (e.g., locomotion)	
C5	In-house virtual train running on a PC	Hardware test track and real train after simulation	There are scenarios that cannot be tested in a simulated environment
C6	In-house simulators	HiL used only on the integration branch	Simulators are very limited in functionalities
C7	Simulators relying on virtual machines not widely used due to their costs and complexities	HiL each time when it is possible	Discrepancies between HiL and simulators due to real-time constraints and differences in the environments
C8	In house simulators rarely used due to the complexity of the system to simulate	HiL used for complex scenarios	It is better to spend time on testing on real hardware instead of implementing and developing complex simulators
C9	In house simulators for simulating tunnels with antennas	HiL used for scenarios that cannot be tested with simulators	Discrepancies between HiL and simulators due to differences in the environment (e.g., speed differences or tags being not readable)

Focusing the attention on the adoption of simulators, C1 is the only company that does not include simulators in the development process. However, the interviewee points out the desire of having simulators especially for the microcontroller and the input signals coming from acoustic sensors.

C3 is the only company that does not develop the simulators in which they rely on probably due to the safety integrity level of the domain in which the company works. The remaining companies develop their simulators in-house, and this introduces challenges and limitations. Specifically, C4, C6, C8 and C9 use simulators only for easy to simulate functionality. Concerning the discrepancies between the outcomes from the simulators compared with the real devices, C2 points out that the main root cause for these discrepancies resides in the complexity of the simulators, which must simulate complex hardware features.

Only C3 does not rely on testing activities involving the real hardware due to domain-specific constraints. Specifically, in the aerospace domain for using the real device it is important to have a controlled environment (i.e., clean room) and it is not possible to interact with the devices in the clean room. On the other side, C1 only relies on HiL probably due to the low costs of the devices and to their availability ("*We do not have problems with scares resources*").

C2 operates in the avionic domain, where the cost of the hardware is very high. For this reason, the company mainly adopts simulators and operates on the real devices only when there is a high level of trustability in terms of correct behaviour and absence of crashes.

Alternatively, C4 and C9 point out that in their domains there are scenarios that cannot be tested relying on simulators, so HiL becomes mandatory. As an example, the interviewee from C4 reports: "*Everything, except locomotion can be 100% tested*". Moving the attention on C5, it is possible to state that the company moves from using only HiL to rely on simulators due to the high costs needed for booking real trains and the efforts needed in terms of people required for testing on real trains. Similar to C4 and C9, also in this case, there are scenarios that cannot be executed and tested on a simulated environment. On the same line, C6 highlights that simulator cannot be used for non-functional testing, for which the real hardware seems to be mandatory.

C7, instead, rarely uses simulators due to their complexity for being compiled and their expensiveness in terms of costs and effort. Finally, C8 clearly mentions that the complexity of their applications hinders them from spending time in developing complex simulators resulting in a huge adoption of HiL testing. Of course, the latter means that for complex scenarios it is not possible to rely on simulators while it is better to use the real hardware directly.

4.8 CI/CD PIPELINE CONFIGURATION

This section describes the CI/CD pipeline configuration adopted within each company involved in the semi-structured interview. Specifically, the focus is on the build triggering strategies, constraints related to the build execution time, changes to the configuration, and flaky behaviour being observed together with strategies adopted for addressing the flakiness.

- C1 does not yet have a CI/CD pipeline for the CPS context.
- C2 has an MLOps pipeline with the following characteristics:
    - configuration does not change very frequently;
    - configuration does not have nightly builds, meaning that also slow builds are continuously built at every change;
    - build execution time is approximately equal to 1 hour;
    - lacks automation in terms of closing the loop by automatically analysing the results coming from the field.
- C3 has a fixed CI/CD pipeline that:
    - rarely executes nightly builds;
    - has some build execution time constraints. As an example, if the build execution time overrun a specific threshold it is likely that the simulator has encountered some problems such as memory leaks that have nothing to do with the software under test.

- C4 has a CI/CD pipeline that:
    - is triggered at every change;
    - has a fixed configuration since there is not the need to deploy on different environments due to the fact that they are the vendor for both the hardware and the software attached to it;
    - has a configuration that is dependent on the branch. The latter means that each branch is a proper customized CI/CD pipeline;
    - shows flaky behaviour due to the fact that in simulated environments, it is very hard to guarantee that the resources being used are always the same, and more importantly, due to the load on the server-side. The flaky behaviour is mainly addressed with retries, or in the presence of resources' issues, the maintainers of the pipeline try to find and remove the root cause of the flakiness.
- C5 has a CI/CD pipeline with:
    - a fixed configuration: minor changes to the configuration are needed as a consequence of the software evolution;
    - flaky behaviour due to external tools that may behave differently (e.g., virtual machines), and load on the servers. The flakiness is mainly addressed by trying to find and remove the root cause of the flaky behaviour.
- C6 has a very complex DevOps process since they have a branch for each component together with an integration branch. Furthermore, they also have customized builds for non-functional requirements, such as for doing performance testing. The CI/CD pipeline within each branch has:
    - incremental builds running at every change in which it is required to select the type of tests to be included based on the type of changes and the elements being touched;
    - nightly builds relying on simulators that execute the whole test suite;
    - nightly builds for the integration branch relying on real devices that execute the whole test suites once having integrated and compiled all the components;

    Flaky behaviour may arise due to network issues as well as the complexity of the interactions among the sub-systems that may lead to non-deterministic behaviour. Retry is the most obvious action for dealing with flakiness, even if there are cases where it is important to address the root cause of the flaky behaviour within the code directly.

- C7 has a CI/CD pipeline which execution time has dictated the adoption of the incremental build-triggering strategy. The flaky behaviour observed is related to the presence of flaky connections about missed messages and retries.

- Finally, C8 and C9 do not have a CI/CD pipeline for the CPS context.

4.9 CI/CD CHALLENGES AND BARRIERS FOR CPS

Table 5 summarizes the main challenges and barriers the interviewed companies are facing when putting in place a CI/CD pipeline for Cyber-Physical Systems. As the table shows, a recurrent challenge is the need to automate the deployment towards HiL, which often occurs late in the development process, and for sure never in continuous builds. This pairs with the unavailability of simulators suitable for testing certain properties (e.g., real-time properties, but also certain complex execution scenarios), and, very often, with the need to define automated oracle checks from CPS outputs.

In particular:

- In C1, the tests are executed manually, and the output (a sound) is checked by a human.

- In C2, the interaction with hardware boards represents the biggest challenge. Moreover, simulators may be overly imprecise, making it difficult to discriminate errors originating from the simulators from actual errors detected in the pipeline.

- In C3, there is a lack of integration between the development environment and the (simulated) execution environment. Also, often the production code is not available in the V&V environment, making analysis and testing activities challenging.

- In C4, the biggest challenge is to "close the loop", i.e., to gather execution data from the HiL and produce useful feedback for developers. In particular, there is a limited observability of the execution environment, making it difficult to identify the problems' root causes.

- In C5, a major problem is in need to cope with different HiL (i.e., trains) and, consequently, different simulators. Also, there is limited software reusability among those different environments.

- In C6, the main challenge is the long deployment cycle. This is not only accentuated by deployment difficulties, but, above all, by the expensive and not-yet-optimized test suites. Another problem is that development branches are component driven, and integration comes late in the process, therefore integration errors are discovered late.

- In C7, the main CPS-specific challenge is the need to fully automate the test execution, and above all the oracle check. Moreover, there is the need to integrate full traceability management in the DevOps pipeline.

- In C8, test automation is the main challenge to be faced in order to setup a fully-fledged CI/CD pipeline. Furthermore, currently testing is mainly performed on real-hardware, and a CI/CD pipeline capable of enacting continuous builds will only be possible with the availability of simulators that, due to the very complex scenarios to be contemplated, are not available and are quite difficult to develop.

- Finally, in C9, specific challenges are due to intrinsic limitations of simulated environments, which will not allow the testing of some properties (e.g., distance-related properties) as well as real-time requirements. At the same time, the interfacing of the pipeline with real components is challenging, and achieving a complete CD process (with HiL) is something still on the company's agenda.

Table 5 Challenges and Barriers as reported by the interviewees

Company	Challenges	Barriers
C1	• Test automation with HiL • Transform acoustic output analysis into automated signal processing • Use of simulated environments	• Lack of automated check for the oracle • Lack of simulated environments
C2	• Discriminate between simulator-related errors and pipeline errors • Deployment	• Interaction with the hardware
C3	• Integrate development and execution environment	• DevOps culture • Production code unavailability • HiL not feasible
C4	• Testing on the HiL • Gathering information back from the HiL, and learning from feedback	• Limited observability
C5	• Configure many different architectural variants (hardware components) • Automated deployment on HiL	• Use of different HiL with limited software reusability among them
C6	• Slow cycle time • Feedback to DevOps not fast enough • Tests take too long to execute	• Integration occurs late in the process • Deployment on HiL occurs late in the process
C7	• Integrating requirement traceability in the DevOps process • Testing based on formal specifications	• Test case outcome (oracle check) has to be determined manually
C8	• Testing automation	• Lack of suitable simulators especially for complex scenarios
C9	• Interfacing with HiL • Testing complex scenarios with simulation • Testing real-time requirements	• Deployment on HiL • Test execution requires manual intervention

5. CONCLUSION

This document reported the CI/CD practices collected by interviewing experts from nine companies. We described the interview structure and protocol, and then we summarized the main results gathered.

Results highlighted that the main barriers related to setting up a pipeline for CPS are related to:

- Need for manual intervention in the tests, for their execution and above all for the oracle check;
- Lack of suitable simulators, or, where available, simulators are not able to cope with all the scenarios required during development;
- Difficulty to automatically deploy software on the HiL, which often delays fault discovery or delivery.

Also, challenges companies are facing, often originating from those barriers, include:

- Shorten release cycles;
- Fasten the feedback cycle to DevOps, by also reducing build (and in particular test) time where possible;
- Ensure full traceability throughout the development process, also in the CI/CD pipeline;
- Improve the usage of simulators, also for complex scenarios;
- Fully automate test execution, also on HiL;
- Fully automate deployment.

Our next steps will be to (i) use grounded theory to devise, starting from the collected information a model of CI/CD practices for CPS, and (ii) validate the model through an extensive survey, conducted both within the COSMOS consortium and outside it.

6. REFERENCES

[1] M. Hilton, N. Nelson, T. Tunnell, D. Marinov and D. Dig, "Trade-offs in continuous integration: assurance, security, and flexibility," *Proceedings of the 2017 11th Joint Meeting on Foundations of Software, {ESEC/FSE},* pp. 197--207, 2017.

[2] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. C. Gall and M. Di Penta, "An empirical characterization of bad practices in continuous integration," *Empirical Software Engineering,* vol. 25, no. 2, pp. 1095--1135, 2020.

APPENDIX A - INTERVIEW STRUCTURE

DEMOGRAPHIC INFORMATION

- What is the approximate number of employees of your company? What about the size of your team?
- What is the operational domain of the team where you are employed (e.g., autonomous driving cars, robotics, drones, avionics, satellite, etc.)?
- What are the programming languages being used in your CPS?
- Does your team rely on a CI/CD pipeline for developing CPSs? If yes, when and why did your team introduce CI (i.e., approximate number of years)? If not, are you planning to introduce it?
- Do you know whether in your company different teams rely on different CI/CD for CPS? If yes, can you please give us a contact of a developer working on a different team who may be available for the interview?

BACKGROUND INFORMATION

- What is your academic background?
- How many years of development experience do you have?
- How many years of experience do you have in CPSs?

ROLE IN THE COMPANY

- How long have you been in this company?
- What is your role in the company/team?
- What is your role (if any) in the CI/CD process?

CI/CD PIPELINE STRUCTURE

- What is in your opinion the biggest problem that needs to be addressed while considering CI/CD pipelines for CPSs? Can you rank them somehow?
- What about benefits and challenges in adopting/moving to a CI/CD pipeline strictly related to the CPS context?
- Note: The above two questions are exploratory. They will be asked again in the end in order to confirm or contradict once having run out the whole interview.

PHASES AND STEPS

- Which are the phases of your CI/CD process (e.g., compilation, testing, verification, deployment, etc.)?
- Are there some differences in the CI/CD pipeline being used in software domains different from CPSs? Are there some specific reasons behind the above differences?
    - Are there some properties/characteristics that your pipeline has to satisfy based on the fact that you are developing CPSs? Examples of properties may be accessibility of hardware resources as well as simulators, security issues, performance, or build execution time.

TOOLS

- What are the technologies used in your CI/CD pipelines? Specifically, for:
    - versioning
    - build automation (e.g., Maven, Gradle)
    - CI automation server (e.g., Jenkins, GitLab)
    - Other tools (e.g., dockerization, Kubernetes, etc.)
- Are you customizing/integrating the steps in the CI/CD process by using shell scripts? If yes, why?

VERIFICATION & VALIDATION

- What are the tools mainly used for static analysis?
    - How did you configure static code analysis tools being used to verify the overall software quality? Other than focusing on code styles, are there coding standards that have to be met in order to have a product that may be certified?
    - What about the usage of static code analysis tools aimed at identifying security issues, as well as vulnerabilities and possible memory leaks? Are there other factors you usually consider while integrating static code analysis tools?
    - What about the usage of static code analysis tools for identifying possible bugs inside your CPS code?
    - Other types of static analyzers?

- What type of testing frameworks and testing activities are mainly involved in your CI/CD? For instance, do you rely on unit and integration testing?
    - Are unit, integration, and system-level test cases manually written? Why? What about functional testing?
    - Are you using some specific testing frameworks for non-functional requirements?
    - Do you apply field testing? What about simulation testing? What are the challenges/barriers in applying this?
    - A key aspect in CPSs may be related to certifications and standards. Do you consider any kind of compliance testing? What are the challenges/barriers in applying this? How much of your testing is done for certification?
    - Does the pipeline test for safety, privacy and security, as well as reliability and availability requirements?
    - Finally, since using different testing activities will increase the value of your final product by increasing its cost: do you include any kind of prioritization for selecting a subset of test cases to be run?
- Do you use Model-Driven-Testing and/or Test-Driven-Development? Why? If you develop by following TDD, who is responsible for writing test cases?
- Do you have a precise expectation of how your code should behave? For instance, in terms of the order in which certain events should occur, or the time they take or on how the data being processed by your code looks at various points?
    - If you have such properties of your code written down, do you already have a tool that can show you when/why/how your code adhered to them?
- Do you have a way to execute code under various inputs from its input space?
- Do you use system models to support test automation?
- How do you validate your system models?
- How do you identify critical scenarios (scenarios leading to failures) for system testing?
- How are your performance requirements expressed?
- How are the results of performance analysis presented to engineers?
- Do you have a way to determine root causes behind performance problems? If yes, how are root causes reported to engineers?

DEPLOYMENT

- Does the CI/CD pipeline include Deployment? Does your CI/CD process depend on the target environment? Discuss challenges related to delivery on HiL/CPS.
- Do you have processes and tools to observe differences on CPS behaviors between the development and the deployment environment?
- Do you perform any runtime monitoring of the system? How does it work?

INCLUSION OF AI CAPABILITIES

- Do you have AI capabilities included/integrated in the development of CPS? Specifically, does your code base use ML components or AI components? What types of CPS actions are controlled/decided by ML components or AI components?
- For which development tasks do you rely more on AI/ML components? Are you using supervised or unsupervised learning approaches?
- What are the strategies used to determine and manage data needed for MLs?
- What about the configuration of the models being adopted? Have you experienced cases where there was a need to retrain the model on a new/different set of data points? What about parameters tuning?
- What about the monitoring of the deployed model? Which data are you collecting to determine whether or not there may exist errors during real-world execution?
- Are you using specific testing strategies aimed at identifying bugs in ML applications since that MLs bugs may differ from traditional software bugs?

INCLUSION OF HARDWARE COMPONENTS AND/OR SIMULATORS

- Do you attach real hardware components (e.g., sensors and actuators) to the CI/CD pipeline? Do you use final versions of the hardware devices or do you rely on simplified HW testbeds?
- Do you make use of any simulator in the pipeline? Why do you rely on simulators instead of considering real hardware devices? What types of simulators?
    - If you use simulators, how did you decide to trust a given simulator? Did you assess the validity and accuracy of simulated operations?

- Can you please give us your thoughts about the possible tradeoffs you consider when choosing between using real HW devices or simulators? E.g., do you prioritize them depending on the features and/or changes applied to the CPS? Do you perform testing directly on the hardware?

CI/CD PIPELINE CONFIGURATION

- Did you customize the CI/CD pipeline based on the project's characteristics? Why?
- Do you schedule different build types (e.g., do you rely on nightly builds)?
- Do you have any performance (Memory and CPU) requirements? From where does performance requirements come from (e.g., usage of sensors data, etc.)?
- Can you provide us information about build triggering policies being adopted in the CI/CD pipeline for CPS?
    - Build triggering? Any change? Only specific changes? Daily builds? Nightly builds? Rebuild everything each time? Incremental build?

CI/CD PRACTICES: CHALLENGES AND EFFORTS NEEDED

- What is the average build execution time? Has the build execution time changed over time? If yes, do you know why?
- What is your maximum acceptable CI build execution time? What are possible barriers in CPSs that prevent you to have the desirable execution time?
- Have you ever experienced that the growing build execution time was due to some bugs in the build tools being adopted? Can you explain more about that specific situation and give us information on how you have fixed the problem?
- In terms of build execution time, how do you deal with balancing testing execution speed and certainty especially considering the safety characteristics of CPS in real word scenarios?
- What are the major costs when configuring the pipeline?
    - Do you have to cope with scarce HiL simulators resources and their limiter realism? What about accessibility of the above resources?
    - Since that it is important to increase security together with the ability to access and modify the CI system once needed: What do you think about having only a few people with the privileges of modifying the CI/CD configuration?

  - What are the challenges in assembling/configuring the pipeline (build, pipeline configuration scripts)? What customization did you apply to traditional CI/CD configuration steps?
  - What about the cost of testing features on the pipeline?
  - What do you think is the most important bottleneck in the way of writing/executing/automating tests in CPSs?

- How often do you have to modify your CI configuration?
  - What are the reasons for the modifications (e.g., dependencies installation or change versions for dependencies)?
  - Would it be useful to have a (semi) automatic recommender suggesting how to adapt the pipeline configuration?

- Have you experienced flakiness in the CI/CD process? What are the main reasons behind the flaky behavior? Is the flakiness in the hardware being included in the pipeline such as sensors (e.g., on the quality of data) and actuators? How did you deal with the flakiness from the CPS?
  - Where does the flaky behavior manifest (e.g., testing failure)? Have you experienced problems due to lack of observability of the flakiness? Any specific method in place to deal with flakiness problems? Moreover, have you experienced problems about the distance between failure and fault of the flaky behavior?
  - What about the coordination and communication between software and hardware components? Did you experience any problems due to lack of a proper documentation or lack of documentation at all?

- Do you monitor your CI/CD process (e.g., by relying on previous build history)? In other words, do you collect detailed logs during operation for debugging and monitoring?
  - How difficult is it for you to determine/locate the reason for the failure starting from the build log?

- Have you experienced specific problems while setting up a CI/CD pipeline for CPS systems (e.g., the inclusion of hardware components)?

- What is in your opinion the biggest problem that needs to be addressed while considering CI/CD pipelines for CPSs? Can you rank them somehow?

- What about benefits and challenges in adopting/moving to a CI/CD pipeline strictly related to the CPS context?

- Do you have needs about auto-configuring the CI/CD pipeline to address portability, compatibility, deployability and dependencies management requirements?

APPENDIX B - INTERVIEW TRANSCRIPTS

DEMOGRAPHICS INFORMATION

- C1: The company is made up of 15 people and there is no separation in teams, all the members of the company work across all the projects inside the company. The team is the company. About the area: Developers focusing on hardware components together with developers focusing on the real software application that has to be executed interacting and interfacing with the hardware components. We are developing and creating prototypes for hardware components/products. Hardware experts are onboard. Useful for sensors' integration in CPSs; help having knowledge about the hardware components, how they work and how it is possible to communicate with them. The main domain is related to developing acoustic sensors (SPL board involved in COSMOS) and energy devices involved in industrial environments and/or smart cities. Programming languages: Python for calibration and testing, C/C++ for microcontrollers. The code written in C/C++ is the one we need to analyze and test for guaranteeing code quality. We do not use any DSL for microcontrollers.

- C2: The company counts around 110/120 employees but overall, it counts more than 2000 employees. The team is made up of 4 different people. C2 overall covers different domains (multi-domains company): medicine, space, aeronautics, defense, IT itself like banking, insurance and so for. For what concerns my domain I am involved in both aeronautics and defense. Programming Language for CPS: mainly Java, C and C++. In defense we also have a little bit of Ada. Java is used for front-end development together with JavaScript. C and C++ is used for the back-end.

- C3: The company has now 16 employees, with three people in the team. In terms of domain, for CPS we work in the aerospace domain, specifically on-board software for satellites. We mainly use ANSI C-99 (C as programming language) following the MISRA rules 2012.

- C4: Total number of employees is 60 with between 5 and 10 developers per team. The application domain is robotics so focusing more on an autonomous robot. Each team is of course involved in both the development of the hardware and of the software components. 80% C++ and 20% Python mainly for user interfaces. We also have customized scripts to interact with the robots that are mainly written in Python.

- C5: Recently we got another company, I do not know the size of the company but for sure it is a large company (very large). I am included in two teams with shared resources with the aim of applying the same development approach independently from the project and the customer. We have around 50 employees in each team. For what concerns the domain: we deliver software for trains. We call it: Train Control Management System (TCMS). We interact with several hundred or thousand computers and/or ships on the train. TCMS is the central point controlling other different systems running on the train. We have different programming languages, each one specific to a device since we have different

hardware components for different situations. Basically, most of the logic is PLC and C.

- C6: Our R&D is about 200 developers for the project we are in. We have around according to Google 30k employees in the overall C6. The application domain of course is healthcare. Programming languages: C# and C++.

- C7: My entire team is around 40 people. We have basically three groups: one working on virtual machines, one working on web services because we provide DevOps solutions for embedded systems misto in Java and OSGi, and finally we have a small team working on customer delivery with the goal of adapting our tools to the customers' needs. Domain of operation: Our development is under Linux but we provide systems for embedded operating systems such as QNX and VxWorks and we provide support for the systems as well. So, we have provided systems for arm for Intel for MIPS. Our main area is automotive and industrial automation but we also have customers in medical devices, aerospace and military. Programming Languages: Real time Java (i) provides real time scheduling above the ten Java priorities; (i) provides APIs for accessing those priorities both as real time threads and events handlers so that you can structure your code in terms of events, either timed events or untimed events. Then we have a deterministic garbage collector, so we do not have anything like scope memory to get real-time guarantees for programs written in vwm.

- C8: We are a small company involving eight people; however, we have some subcontractors for specific activities. For what concerns COSMOS, we have around four-five people been involved. Operation domain is railway transportation and subway. Programming languages are mainly C and C++. We do not rely on Model Driven Development usually used for large projects. SCADE is very expensive even if it has a good business model to sell.

- C9: Our company counts less than 10 employees and my team is made up of three developers. We develop software relying on identification technologies such as RFID and Bluetooth low energy. We mainly rely on Java together with C#, Kotlin and Dart.

BACKGROUND INFORMATION

- C1: Master degree in electrical engineer and computer science. I have been working at C1 from six years.

- C2: Five-year degree in Software Engineering together with a post Master in Software Engineering. I have 20 years of experience in the job market. All the 20 years of experience are about development in aeronautics and defense domains. Until 2010 I was mainly involved in development tasks, then I started managing some projects doing less on development. Nowadays, I mainly manage projects.

- C3:

  - Master Degree in Software Engineering. From 2008 I started with development tasks, focusing on development in the aerospace domain from 2013.

  - Master Degree in Software Engineering. From 2010 I started with development tasks, focusing on development in the aerospace domain from 2016.

  - Master Degree in Software Engineering. From 2011 I started with development tasks, focusing on development in the aerospace domain from 2013/2014 even if not continuously.

- C4: Bachelor in Mechanical Engineering and a Master Degree in Robotics. 7 years of experience in the robotics domain.

- C5: I have 20 years of work experience but only 10 years of experience in the CPS domain. I have a bachelor degree in computer science (obtained in 2010). For the last two years I have been working with CI/CD.

- C6:

  - Degree in Computer Science. CPS development since 2003 with C6 and about years of development since 1990.

  - Bachelor degree in computer science with a focus in software systems. Development experience for 14 years with 10 years in CPS. I started with building technologies more in the security area. The journey started there.

- C7: I joined the company 20 years ago. I have a bachelor in Physics and Computer Science and a master in Computer Science. I have a PhD in Computer Science. I spent the first two years of my career working on CAD/CAM, then I have worked on designing wafers (for scale integration tools). After that, I worked on systems based on LISP for real time data analysis for Doppler radars.

- C8: I started my professional career around 25 years ago working on a cyber-physical system supervisioning the train communication. During these 25 years I served different roles: software designer, software developer, project manager and team leader.

- C9: I have a master degree in Engineering for Computer Science, I have three years of development experience in the CPS context.

ROLE IN THE COMPANY

- C1: I mainly work as software and hardware integrators.

- C2: Project Managers. For what concerns the role in CI/CD pipeline, I am in contact with the development team for the configuration and setting of the pipeline (kind of indirect relationship).

- C3: Research and Development Manager together with an Engineer responsible for cloud applications and a Manager for the IT solutions for industrial applications. The responsible for cloud applications is mainly involved in setting up the CI/CD pipeline. However, there are no strict roles meaning that a developer who needs to customize a CI/CD pipeline by simply using Yaml files can customize it directly. The setting of the pipeline in terms of infrastructure and architecture has been realized by the responsible for cloud applications.

- C4: I am leading the software quality process inside the company. I am also part of the DevOps team which is in charge of configuring and setting the pipeline but also improving it.

- C5: Officially, I am a software integrator involved: monitor all the builds, identify any issues, configure and enhance it with the CI/CD. On the device level I do some integration activity requiring coding.

- C6:

    - Since 2003. DevOps Architect for three years. When the R&D desired a better approach for the whole thing. I was responsible for establishing the first complete pipeline until deployment. "Lead Architect at the system level"

    - For 10 years. I am team leader of the so-called DevOps enablers: our mission is translating the technologies we develop here also with respect to the pipeline to the developers but also generating an impact on services meaning saving money. I am also a product owner. "Team Architect".

- C7: I am both CEO and CTO of the company. I am also the spec leader for the updates of the real-time specification for Java (RTSJ).

- C8: I am the team leader for the project.

- C9: I am a software engineer working on both the mobile development and the software development dealing with sensors and/or actuators. I have set up the CI/CD pipeline being used in mobile development and I am also one of its users.

CI/CD PIPELINE STRUCTURE

- C1: We do not have a CI/CD pipeline for the development of the SPL board. We would like to have it to automatically build software when we have new commits and new releases to check if it compiles and it can build an image.

- C2: For the demonstrator in COSMOS, we still do not have a CI/CD pipeline in place. We have CI/CD pipelines using Jenkins and Maven/Ant to deploy some statistical applications and some applications used as support. We have one pipeline that is right now in a continuous improvement (still changing and adapting) stage used for the development of a real time operating system. This pipeline includes the HiL and gives us the possibility to collect feedback/evidence that may help us in obtaining the certifications. I have to

highlight that the demonstrator we are developing in the avionics domain for COSMOS is more an MLOps than DevOps. It is more an MLOps pipeline than a normal CI/CD pipeline.

- C3: We have a partially implemented pipeline for the aerospace domain and we have started working on it around 10 months ago. The CI/CD pipeline has to be triggered manually by the developer who wants to start the execution of the tests, or else we rely on nightly builds. We are only involved in the execution of verification tasks so we rarely rely on nightly builds. The latter is due to the fact that in the aerospace domain, developers and testers have to be different. We do not interact with the CI/CD pipeline of the clients developing the aerospace software. For what we know, the CI/CD pipeline of the clients starts at each change. We do not have a complete DevOps process; our pipeline for now is only used for automatizing the test execution.

- C4: We have a CI/CD pipeline that involves the deployment of our code to the customers. All is shipped through Debian packages.

- C5: Yes, we have a CI/CD pipeline in place that was introduced two years ago. The choice behind the adoption has been motivated by the obvious benefits obtainable using a pipeline but also for every developer and testers to use its own computer it will require a lot of time for compiling the whole project. So, if you build it in the cloud, you have a more powerful computer, it is more configurable and it will go faster. Now, all the members of the team work more together. We continuously monitor the build process and in presence of a failure all developers are stopped until the build becomes green again. People are aware that the problem has to be directly fixed before continuing their normal activities. It costs a lot of money having something broken.

- C6: We have a CI/CD pipeline that has been introduced many years ago. CI in pure software levels is around 10 years but in terms of applying CI/CD for CPS we started around 4 years ago.

- C7: One area on which we are working on is improving our pipeline. At the moment we are able to support updating software on devices on the fly. What we are doing is adding DevOps to that system so that development can be directly brought down to a device. I think that with Java you can emulate things on the desktop and then do a lot of development there, but there are also timing issues and other things that you may want to test on devices. So, we use the Bndtools that helps upload our cloud product and download it onto devices. At the moment it is still kind of an infancy we are still working on improving. In summary, the pipeline is mainly for deployment purposes but is somewhat detached from what is the development and testing phases. We are working on making this connection right now.

- C8: Actually, we do not have a CI/CD pipeline for this project. We do not have a server in which tests are executed every time somebody does a change.

- C9: For what concerns mobile development we already have a CI/CD pipeline for testing and automatic deployment of our apps on the play stores. For what

concerns the development involving sensors and/or actuators we are planning to set up a CI/CD pipeline by the end of the year. The main reason behind not having a CI/CD pipeline for the CPS context is that we now are under-dimensioned (we do not have enough developers) who can solve this task.

PHASES/STEPS

- C2: The pipeline we are using are more CD than CI since we use them for automatizing the deployment.

- C3: The pipeline mainly involves the compilation and the execution of the unit testing due to the fact that those two phases are the ones that are much easier to automatize.

- C4: We have a branch for each feature that needs to be implemented and/or improved and we use PRs to merge the work in the stable release branch. Before merging there are several steps that need to be done such as static code analysis to give feedback to the original developers in terms of whether or not their code improves/degrades the overall code quality of the system but we are not enforcing anything at the moment. Then we run unit tests that can also involve Ros (simulators). If the test passes and the code is peer reviewed, it is possible to merge on the mainline. We also rely on nightly builds (that is not so expensive in terms of execution time) that run tests on already packaged components. So, the deployment does not happen at each PR but only during the night. After that we have a proper release process to ship the product to the customers.

- C5: In general, we have different environments we compile and build. We have a set of different configurations for each device and in the end, we deploy everything together, and we run it on a virtual machine (virtual train, software running on a PC that should behave like it does on a real train). After that, we execute some automatic tests. At the moment, we are only testing only based functionalities. We have not included deeper testing usually done with real trains due to resources limitation, even if most of these tests can also be easily automatized.

- C6: We have metrics, builds that instrument indeed we have different types of instrumentations, test coverage analysis. The nightly builds have automated deploy without reusing existing artifacts while building all of them from scratch in a clean environment. We have three different PCs (physical systems that are equivalent to the real hardware in the CT Scanner but not connected to anything around it, so it is a standalone system which has a simulator running on it) used for deployment and every night there is a full deployment on one of these PCs. So, the test execution during nightly builds is executed in a real product environment. The same procedure is used on the integration branch where the components of our 70 teams are integrated in a single joined point. Based on this integration branch, we automatically deploy on a real CT Scanner every night on which we will run three different levels of testing: we had some kind of firmware testing that needed the environment, we had subsystem testing and we had complete system testing. The integration branch is the point where you get new features on a system level. That means that as soon as it is integrated and

tests are green on that level, the system test will consume such a release package and do the formal tests based on that.

- C7: The pipeline is mainly for deployment purposes. We also have tests that we run on our embedded systems by relying on Jenkins but those are internal test machines that are not quite bound to the deployment part. The latter justify the reasons why we want to combine the two parts at the moment. We foresee in the pipeline running performance and security testing as well.

TOOLS

- C1: We use GitLab and git for versioning. Several branches for maintaining and developing different features. We rely on makefiles for building the code, generating the image and uploading it to our servers. We use infrastructure tooling such as Docker and Kubernetes in contexts different from the one involved in COSMOS (SPL board development). At the moment we do not have any static code analysis tool to which we rely on.

- C2: We use Jenkins together with Maven, a little bit of Ant and for versioning we rely on Git. We use Docker but not in the development process of the real time operating system (SPL board development). It is useful in a couple of projects to facilitate the deployments on the clients. The main problem is that the process works in a one-way direction. In other words, the pipeline is used for deployment but we are not able to obtain feedback from the pipeline once the application has been deployed on the clients. For deployment we use out-of-the-box tools while for testing we typically create our own test scripts. We mainly rely on Cantata for what concerns static analysis and testing, while use Doors for mapping the requirements on the code.

- C3: Jenkins and GitLab, git, in the aerospace domain we use SVN and makefiles, Ant and Maven as build frameworks, SonarQube. Rarely, we include shell scripts in the CI/CD configuration.

- C4: Git in combination with Gitlab. We rely on the CI/CD infrastructure provided by Gitlab. We also have a PostgreSQL Database that contains information about static metrics used for code quality and code coverage information from the unit test execution. For Deployment we use Debian packages and Docker. The pipeline is fully dockerized.

- C5: We have fog solutions on our servers. We have Jenkins as the ground with several plugins.

- C6: Microsoft Technologies: all is TFS. We are trying to switch to git for versioning. In addition, we leverage WDS (Windows Deployment Service) for automating windows installation that is not very fun without this service. We have one project running in a containerized environment with Docker (infrastructure automation) but this is not something we have on a broad base. Again, the Windows situation does not help us with dockerization and our application strategies are not yet towards isolated aspects to be dockerized or containerized.

- C7: We use GitLab and Jenkins for continuous testing in our development but this is mostly for our host app. We just use virtual machines for deployment because they are actually better than containers in a certain sense: for instance, you have more control over the resources being used: we have the ability to enforce our resources usage inside the virtual machine while you do not have quite the same extent with a container.

- C8: We rely on GitLab but we have a manual versioning that means that there is documentation and software that represent two different parts that must correspond to each other. All our software is on GitLab and we use it to prepare releases once a number of requirements have been implemented. As build automation we mainly rely on make files.

- C9: We use GitLab for versioning.

VERIFICATION & VALIDATION

- C1: At the moment we do not have any static code analysis tool to which we rely on. It has to come together with the definition of proper strategies to be used for orchestrating the development process (like code reviews and following coding guidelines for code quality). Expectations: Rely on static analysis tools for helping the development process in finding bugs and potential failures on the systems to guarantee quality. Also, to decrease development time. At the moment, when we release, we have to test each single feature manually of the device: sometimes we can forget to test something that has a problem, while other times we can test something not in the right way to make a problem arise if present. In these cases, it will be very tricky to guarantee the quality of the product when doing software releases. Mostly quality control and reduce development time and effort. Coding standards: we try to maintain each repository with common coding styles but we do not have strict coding guidelines. We have a single maintainer for each repository at least for what concerns the master branch. PR based development process. Testing: from time to time, we do performance testing since it takes some time to execute performance tests on the devices. When releasing we try to test all the interfaces/commands to the devices to guarantee that the main functionalities are still working (kind of regression testing on the integration). Certification standards: not about the code but we have specific certifications to follow for what concerns the acoustic signals that we are actually able to test even if manually. Security and Privacy concerns: The code itself is protected. The updates are done with encrypted images. So, the only security concern is on whether the code can be extracted from our encrypted images. No security analysis and test. Level of automation for the test: The testing for the software is all manually done. For the hardware devices, instead, everything is automatic: testing for power supply, calibration, communication interfaces, electrical part and acoustic part of the microphone. The commands and the interfaces with the software are tested manually. Even if there is a test suite the test cases are conducted manually even if they can be automatized. The main challenges for automatizing the test execution: good way to model the test itself and have an oracle that can compare with the actual behaviour. Also, the oracle should be modelled in order to have a proper level of automation for testing execution. The

most challenging part to be tested is related to the acoustic part that has to be tested in a controlled environment (almost done in our office where we have a lot of noise). Try to do it in a simulated environment. By assessing the acoustic signal manually what about the possibility of the task to be error-prone and subjective? Humans can be mistaken and could be influenced.

- C2: We use ASCATs for memory leaks and for complexity in terms of the level of nesting (maintainability and spot bugs). We mainly apply unit and integration testing, while system and non-functional testing is done outside the pipeline. It is highly recommended (not official or mandatory) that developers run unit testing in their private builds. Slow builds hinder the inclusion of running non-functional testing in the pipeline. We also have cases where the complexity of the non-functional testing hinders their automatization in the CI/CD pipeline. The latter mainly occurs for the real time operating system. As regards the development of the real time operating system for creating the tests we start from safety requirements. In other cases, the tests are written starting from general requirements. Testing strategies for certification purposes: yes, it depends on the level of the certification. if it is a design assurance level that is C or less (usually the norm), the development team (eve if it is not recommended) can perform the unit and integration testing, while if it is higher than C (B and A) than the testing is usually done by a team independent from the development team. Violations for certification requirements are checked with code coverage tools: memory leaks and level of nesting are examples of rules for certification. In case of A or B you also have to consider the presence of recursion. There are a different set of rules for different levels of assurance (A, B, C or less).

- C3: For ASCATs, in some contexts we rely on plugins for static analysis to check unreachable code, fan in and fan out for the components, complexity especially in the aerospace domain in which we have few resources on which we can rely on. We use SonarQube, which helps us in checking the fulfilment of the MISRA rules (for certification), the presence of duplicated code, and also for spotting possible errors. In the aerospace domain, among the requirements we receive we also have non-functional requirements expressed in terms of rules available in SonarQube. The latter is mainly for certification purposes. We also have cases, where the strategies to be applied for defining the test cases are explicitly expressed as a non-functional requirement (i.e., use MC/DC for deriving the test suite). Performance requirements (schedulability analysis) are mainly checked and evaluated directly by the clients due to the fact that some tools used have high cost in terms of license. Compliance testing: We need to highlight that the requirements for certification purposes are involved only when developing the on-board software while not for the verification tasks we often are involved in. Non-functional testing: safety yes, since that the on-board software of satellites has a level of criticality B meaning that you need to guarantee a certain level of robustness; reliability, yes since that on board software for satellites must have some functionalities, even if the reliability is usually tested during integration testing. Tests are mainly classified into two classes: nominal and robustness. In the latter case you usually inject an error in the software to check how the system behaves/reacts in presence of unexpected inputs (inputs having values out of the admissible range). Test Selection and Prioritization: This is a problem on which we are working right now. Some

strategies rely on genetic algorithms with the goal of optimizing the resources available for the testing execution environment. In other words, we are trying to optimize the resources we have in order to reduce the overall test execution time. Trade-offs between tests to be executed on the fly with test batches (test suites that have to be executed) while having only one environment available for testing.

- C4: Regression testing for sure by using bot simulation and real robots. If you want to sell a robot you do not need to have a certified robot, so we do not have testing for checking the adherence to specific standards for certification purposes. Non-functional testing for reliability: running the robot a long time with a guy supervising the test execution to understand when and why the robot starts to not work anymore. This is a primitive approach we use for reliability testing (simple measure for the moment). We have more protection against hackers that is not part of C certifications that we need to guarantee (this is more security). For what concerns safety requirements, we have one aimed at guaranteeing that once pressing the stop button the robot actually shouts down, and a different one is related to the fact that it is not possible that simultaneously different people have the control on the robot. Usually, safety requirements are written highlighting situations that could never happen.

- C5: We test how the component interacts with each other. We test specific train functionalities such as whether we should activate the train in this mode. Usually, we have a long sequence of events for each test that involves different devices and components so we are mainly doing integration testing. Private builds running unit tests. Our rule is: All developers must make a private build before pushing on the mainstream. For the moment we cannot configure the number and type of tests to be executed locally, so also in private builds the developer will run the whole test suite running within the CI/CD pipeline. This lack of configuration is mainly dictated from having a kind of sanity check. The test records are only used as an indication of the fact that we can use this version to test it officially. Static Code Analysis: I am not sure we have any static analysis automated. The current process is that we have code reviews that should follow principles that are not automated (something we need to improve). Interaction between Code reviews and CI/CD process: I am not aware of any interactions. The code review process is mainly used since we need expertise on the developers' side for determining whether or not a train is behaving in the expected way. This is not easy to automate. Requirements should be understood differently by different people and all may be correct. It's quite time consuming to onboard new developers. Testing strategies: we do not generate test cases automatically but the developer writes test cases starting from the requirements. We have non-functional tests that at the moment are executed manually (not automated). Around 20 general test cases for non-functional requirements. Compliance testing: always we have standards to follow, being used from the requirements, to the development, to the testing. The main challenge is: how do you read the standard? The standard is interpreted so the same requirement can be differently interpreted by different people (challenge for automation). Deviation between tests run on the virtual train and on the hardware track and real train are seen very frequently. Only when the tests in the virtual train are green can we move to the next step. We want to catch as many

defects as possible while running on the virtual train since it is very costly to test on trains.

- C6: We have a Sonar instance that will compute any kind of static code analysis together with identification of TD in the code. I am not sure in which kind of builds these static analysers are run but for sure each team in each development branch has at least one build checking for static properties of the code. How is SonarQube configured? We are not the receivers of this information the developers' team are deciding the configuration. For sure we need to follow medical application frameworks providing a base set of rules in terms of how to build applications and how to integrate them. We adjust and amend them with our own metrics. It's tailed and it is not standard. Also, security analysis is going on by relying on SonarQube. Security Analysis: for sure we use static analysis but we are not sure whether there are some kind of testing activities aimed at accomplishing the same task. Security Testing for the Sec DevOps team is something we do very late (near the finalization of the product) in the process and it is usually done manually and it is not continuous. Performance testing: for two years, we have specific performance test builds that test whether each component (some components) stays within the resource limits they are assigned to. These requirements come from the customers who while using the system will point out the expected time for receiving a specific functionality from the system. The results of the performance testing are usually compared over time in order to see any specific trend in the performance being measured (is it increasing/ is it decreasing). Tests are manually written. Compliance testing: no. We have processes that define whether our development is following the regulatory standards for developing medical applications. Using the above processes, we are sure that the developed code follows the regulatory standards in our specific domain.

- C7: Security testing is the most difficult because those are exactly things that you do not expect to happen. People try to get into the system by breaking the normal flow of control. So, it is straightforward to do the testing where you say you wanted to do this positive test. But if you say, I do not want somebody to be able to get into the system if they do not have the right password, for instance, that's a negative test and it is very hard to prove a negative. Those are mainly negative tests that are not easy to specify. For what concerns the execution of both performance and security tests that may be very expensive in terms of execution time, I think that you have to segregate the tests based on your changes. For instance, if you change a part of the system that does not really end up in the runtime system, you have different tests that you want to do then if it does. On the other hand, you want to keep a continuous set of performance tests: in real time systems it is very important to monitor performance at each change so that you will be kind of sure of what is the change that has produced a degradation in the performance of the real time system under development. To summarize, you won't be able to do all testing for every change, but it is important to have nightly or weekly testing executions that include the whole test suite executions. For what concerns static analysis, memory safety is something we are interested in, in the forms of both static analyses done by the compiler, but also with dynamic checks that are able to reinforce the analysis by the compiler. However, there are many more things we would like to do. For

instance, we would have ways of evaluating inter-thread communication making sure that you do not have any deadlocks in the system. That's a great place for static tools. However, other than having static analysis, we also want to have formal analysis. We are aware that there are some tools available for Java that you can use easily, but after experimenting with some of them (for instance JML), we realized that they are not in line with what we would like them to be. We have also done some experiments with dataflow analysis for deadlock detection, and it works in principle but getting a tool that is really robust is a lot of work. For what concerns how we generate our tests, we start from natural language requirements and we generate test cases manually from reading them. We know that JML (Java modelling language) has an interesting approach in which it is possible to use JML specifications that give you an oracle and then write tests based on that oracle. So, we know that it will be possible to generate tests by starting from JML specifications, however in our company too little work went into there so we know there is a promising approach but we did not experiment with it. Performance Analysis: we have various performance tests that we run regularly to track our performance as the system evolves. At the moment, we have various benchmarks focused on VMs, but we are extending them with customers-based testing in which we have critical parts of their systems that we use for testing as well.

- C8: At the moment we do not have any static code analysis tools being run for verifying software quality. We are planning to include these tools especially for safety and security requirements, not for checking the quality of the code. At the moment, since we do not have tools supporting us for this aspect, we have a set of requirements that must be met and a set of people that rely on proofs specified in the assessment to verify the safety requirements being met by each software module or subsystem. For checking whether the produced C and C++ code is certifiable or it is compliant with the railway standards and specifications we use a specific complex tool that can be configured and parameterized based on our needs derived from the specifications and standards. Specifically, in presence of C or C++ constructs that are not compliant the code would not compile. So, in the presence of violations we have compilation errors. The big challenge here is time in terms of development and design that results in increasing the cost. To reduce time, and consequently to reduce cost, our needs are trying to automatize both the design and the testing phase. For what concerns testing, tests are written manually starting from requirements and system specification but also their execution requires a manual effort. In case we have one hardware with three partitions each one providing some activities, with one of them being native while the others being under Linux, it is mandatory to test their dependencies in terms of how the three partitions cooperate among them but also how they operate with respect to system resources such as CPU time and memory size. Actually, this is about install tests and performance tests on a specific target platform. In terms of functional testing, we need to rely on simulators. We also run integration tests based on the cooperation of the components under test. Of course, integration testing is not run at every change. It is not necessary if the change impacts only the functionality that is limited in scope to a specific module without impacting the interfaces with other modules/components. In the latter case we also run integration testing.

- C9: At the moment we are not using any static code analysis tools for checking the overall software quality. We have RFID-readers connected to a network on which we execute our tests automatically. Some tests require simulators for what concerns the acquisition of data from sensors. In some cases, the test execution requires manual intervention. For instance, when we need to test a transfer of tags between different antennas, we cannot use automation. We have unit and integration testing. The unit testing is made up of different test suites each one aimed at testing a specific module inside the project. The integration testing is done automatically for those cases where it is possible to rely on simulators, or where there is no need to interact with hardware components, for the remaining cases the integration test is mainly done manually. We do not execute the whole set of test suites at each change. We do a kind of impact analysis to select what are the test cases that are impacted by the change that, consequently, need to be executed. So, in general, we do incremental testing and for each change we manually select the tests to be executed. For executing the whole set of unit test suites, we need around two hours. We do not have any security testing, while we rely on both performance and reliability testing. 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.

DEPLOYMENT

- C1: We have a board for doing the deployment and calibration of each SPL device. The board has needles to connect to certain points through the world, so it can control its pinout and communication and everything. The board does not need to have a computer attached to it. It is an autonomous CPS system that is used to deploy, calibrate and test each SPL device. The deployment is mostly all automated.

- C2: The monitoring of the system is done through the pipeline for what concerns the board. We get information from the pipeline about for instance scheduling and memory.

- C4: Use Docker in the context of the pipeline: when we package our software, we want to make sure that all the dependencies are correct and we do not rely on dependencies that are locally installed into our servers. In other words, it is used to create a clean environment. Moreover, we do not want a manual setup of our build server. We could do that with an image but we think that now Docker is the better solution for our purposes. We also use Docker partially on the robot so we can run it in a privileged mode. The latter allows us to switch between software versions quite easily, each one may choose the version of the software that has to be run over the robot.

- C5: The deployment is automatized within the CI/CD pipeline only for deployment in the virtual environment with the virtual train. The deployment on the real train or on the hardware test track is not automated at the moment even if we are working on making it automatic. Testing and deployment in the virtual environment will need around one hour and half (this has to be optimized by increasing the number of virtual machines). On the test track, it will take one

day with people involved in the testing and on a train a couple days where many people need to be involved.

- C7: What about the scenarios where you adopt VMs for DevOps in CPS? Our basic deployment method is: we have an OSGi system that we put on the platform and our OSGi has a notion of bundles, which are basically the original microservices. There is also a service registry so when a new service comes in, you register it so it is easy to start a new version of this service, run down the old one and switch over the new one during runtime. And then you have a versioning capability: you also have a lot of metadata about how this thing can be configured. These are things that Docker is just now realizing are necessary. Of course, one could say I can put in a Docker container and everything runs in there, but if you start communicating you might want to know how you are communicating, with whom you communicate and whether or not the communication is vulnerable. The above questions need to be answered in a truly microservices environment. However, there is a cost to pay while relying on VMs instead of Docker. One of the things we can do is pre-compile the machine, the bytecode for the machine that is running on, and dynamically link that code so we can get a lot of the performance benefits of relying on the native code. A very important part of this kind of system is that we have absolute memory safety. By looking at a recent post by both Google and Microsoft we found that around 70% of the security violations are due to failures of memory safety. So, by using a garbage collected environment, we can prevent those issues from occurring. However, this is not just because of garbage collection, but also because of the requirements we put on the system. For instance, you cannot get a pointer to something that you did not create or receive from somebody else. Moreover, there is a way to go across the end of array boundaries because they must be checked, indeed if you do not check them the garbage collector is worthless.

- C8: The installation of the software could be manually in our laboratory or could be remote meaning that we are able to transfer the binary image directly onto the system so the system is remotely built. Remote installation cannot be used with real systems.

- C9: The deployment is fully automated for what concerns mobile development. In the CPS context, instead, the deployment relies on Docker so we create docker images that are manually deployed to the servers.

INCLUSION OF AI CAPABILITIES WHILE DEVELOPING CPSS

- C2: The original demonstrator was aimed at retraining and also at model building, but more important was focused on finding the oracles: what is the optimal route, continuously improving the oracle with the knowledge-base. There is nothing related to choosing the proper model inside the pipeline, while the scope of the retraining is mainly the fine-tuning of the model itself without recalibration of the hyperparameters being used by the model. How to test it? The concept of testing was: we have a set of baselines expected results, we run the algorithm, we collect a number of additional results that we compare with the expected outcome and based on the comparison we refine the model

and the algorithm based on the obtained results. The comparison has the goal of classifying the results in terms of: is worst, is better, is far worst, is far better and so on.

- C4: We have some ML components for inspection capabilities and also some reinforcement learning for walking. We try to test it in the same way, mainly black box testing for testing the overall learning framework (model-base controllers). We do not test the learning algorithm. The reinforcement learning is aimed at making the robot able to walk so we test whether or not the robot walks and that's it.

- C6: The company delivers applications that contain AI components but, in our pipeline, there is no direct AI involved. For the analysis of errors, we are using a learning-based approach where we try to help with root cause analysis that matches errors patterns in the test and in the build execution to help developers to find where the error comes from. There is a model behind the learning model that helps with analysing errors/failures. Again, this is an infrastructure supported process and not a product.

- C7: We have support for executing AI models in embedded systems but our systems are pretty algorithmic. So, we do not have AI components in our products.

- C8: Since we are developing a safety critical system, our software does not integrate any AI/ML components.

- C9: We do not have AI/ML components in our systems.

INCLUSION OF HARDWARE COMPONENTS AND/OR SIMULATORS USAGE

- C1: We only work with HiL. At the moment we do not have any simulators involved even if it would be beneficial to include them. I see two points that would be nice to simulate: one could be the microcontroller itself, so we could try to render the firmware on it and try to do some testing over it (simulated version of microcontrollers). We never looked into the possibility of simulators provided by vendors. Another interesting part to simulate is the acoustic part helping in also removing noise from the surrounded environment. Since the output of the system is sound and the test should check the sound quality and property it is better to have it in a controlled environment that makes use of simulation. We do not have problems with scarce resources (we have around 10 products needed for testing purposes) since we have a small team and we work with a little number of changes.

- C2: In avionics we do not have simulators and HiL in the same pipeline mostly for certification issues. Based on the fact that in the avionics domain the cost of the hardware is very expensive, we do most of the work in simulated environments, and once we have enough trustability in terms of correct behaviour, absence of crashes, safety we run it on the HiL and try to see whether the behaviour is the same with the ones produced by the simulators. We typically have in-house simulators. The main difficulty in those simulators is related to simulating the HiL inside the simulators. This is the main root cause

(difference) of obtaining different results from the simulated environments and the actual environments with the HiL (for instance a wrong parameter being set in the simulators). In other systems, the difference between the outcome of the simulated environment and the environment with the HiL is in terms of network. The networking has to conform to specific rules for transmissions (it is not Internet) and this may be a root cause for different behaviour being observed. The networking in those systems has to be simulated and it is very complex to implement r664 rules.

- C3: Note that tests are executed on simulators (SVF). Actually, in the aerospace domain, for using a physical simulator you need to have a specific environment (clean room) that is usually out of scope for our tasks. It is not possible to interact with the elements that are inside the clean room. We cannot choose the simulators to be used. ESA has defined a framework for simulation (SMP2) aimed at hosting different simulators for different satellite models for the digital twin of the satellite. The simulators being used are mainly provided by the vendors of the hardware components we need to simulate.

- C4: very simple, we have nightly builds during which we execute regression testing by using simulation relying on walking and locomotion controllers that help us in understanding whether the robot walks, does not fall and meets some speed criteria. In simulation it is quite easy, now we are focusing more on gathering information back from the robot to understand what is going on. Walking is not so easy to simulate so we need a real walking robot for spotting bugs. So, there are scenarios where it is possible to use simulation and other ones in which it is required to use a real hardware device (robot). I think that everything, except locomotion, can be 100% tested in simulation.

- C5: In the past, we have delivered software to trains and the devices were not in a start-up, and these costs a lot of money to book a rail train and it involves hundreds of people. For these reasons, whatever we can do we first run the software in a simulated environment on PCs, and we have test tracks, hardware test tracks in labs (that's the second step). First step is related to using a virtual train running on a PC, the devices are run in some kind of containers and we have frameworks building and connecting the whole set of devices and components. We cannot support the whole set of functionalities of the trains in the virtualized environment. The second step is that we go to the hardware test track, where we have the whole set of devices and even some more that we do not have in the virtual environment. So, we can extend our test in that scope. And also, we have use cases where some test cases will pass in a virtual environment but not in real hardware (limitations). For some test cases, we are not allowed to rely on the virtual environment while we must consider hardware track or real train. Trustability of the virtual environment: we have a process in which we run the same tests in the virtual environment and on the real train. So, with the above process we can prove, by comparison, that the virtual environment is reliable.

- C6: Our simulator is completely self-written. We have the nightly builds in which we only have PCs with no additional hardware connected. The simulator we are using is software only and is completely self-written and it interacts with

the system on a network connectivity level (http protocol connectivity). Knowing the interfaces of each subcomponent we can simulate the interaction as reacting to network messages. Problem: the simulator is very limited in functionalities indeed for instance in presence of a firmware update using our simulator will take around 10 milliseconds while in reality the update of the firmware will last around five minutes. At the moment, simulators are not able to test for performance requirements and are mainly used for functional testing only. For non-functional testing you have to go directly to the system. To be more precise, we can only do performance testing within a specific subsystem relying on simulators. Other project: Digital Twin: they do hardware in the loop testing; we have certain types of hardware being connected such as the tables where they have something simulating the table having the same exact electrical and semantic interface and so you can directly connect the real table or the simulated table. You can decide which hardware has to be emulated and more specifically which features of the hardware have to be emulated. This is an area under development and so it is not widely used during our development process.

- C7: We tried to test against real hardware when we could. When we cannot, we have to rely on our testing on other systems and then employ our customer for the last mile. Specifically, we have a virtual machine with a reporting layer at the bottom. So basically, we have a good history of being able to take code that runs on one system, and run it on a different system without having to change it, because of our virtual machine technology. The only thing that really changes is the timing. So, once you make sure that the porting layer is correct, all the functional testing can be done on any platform and only the final timing tests have to be done on the final real system. For what concerns discrepancies being observed testing on real hardware or relying on virtual machines, one root cause is related to the fact that since we can compile code to the machine, the code can be different. Specifically, we do not use an emulator. We have a virtual machine. We have a version that runs on the desktop and a version that runs on the hardware. Based on this, the differences are only on timing or related to bugs in the actual virtual machine implementation. We need to specify that, it's not like if you write C code and then you have an emulator for your arm that you run on your desktop, and then see what happens when it is put on the real hardware. It's not like this at all. Data from sensors or data sent to actuators: We do not necessarily have the same data input that the end device has. For sure, we have to simulate data coming from real sensors in a real environment (for instance GPS data). We need an environment simulator that helps with the development. We do not use simulators, as much as we would like to, just because they are not so easy to compile and tend to be expensive. This requires a lot of work to adapt the entire system. For instance, for the CAN data, what do you want to wish to happen here, if you are driving around something you need to know how fast the wheels are turning, as well as, what the engine revolutions are together with other sensitive data you might pick up over the canvas. There are a lot of details that are very application dependent. At the moment, we work with recorded data and see how that works, and then we go in the fields for more extensive testing.

- C8: Preliminary tests on simulators can help us but we know that once deploying the code on the real system things can be very different. It could be difficult, demanding and expensive to have a one-to-one relationship between simulators and real systems. At the moment, we would like to simulate the communication and the quality of the communication channel and we would like to test reactions to this component in terms of quality of services and quality of signals. We use simulators only in specific cases because relying on simulators means developing software aimed at simulating that specific part of the system. Since at the moment we do not have complex simulators in place, we prefer to spend time in testing on real hardware instead of spending time in developing complex simulators. So, given the complexity of some features and given the needs and the budget of our company, it's much better for more complex scenarios to rely on the hardware in the loop and only use simulations when whatever needs to be simulated is very simple. And in such cases, we rely on simulators that we use by ourselves: we do not rely on third party very expensive simulators. For what concerns checking whether or not the system behaves as expected on real systems, we obtain from the customers real data, we collect data while running our tests on the real system and then we manually compare the outcome with the real data by visualizing them on a screen during execution. For many scenarios it is not possible to automate the verification of whether or not a system is behaving in the expected way (comparison between real data provided by the customer and actual data acquired during test execution). Of course, the latter is much truer when dealing with safety requirements. In many cases, the same safety requirement is being tested multiple times in different situations in order to be sure that the system behaves as expected. Actually, for the testing phase, we have a set of testers in front of a screen who monitor and check for the presence of any discrepancies about what is expected and what is instead observed while running the system. In real systems, for testing the communication among different components you need to have the train running in a real environment with real traffic, possibly without people. It's a kind of weekly testing session.

- C9: When creating a tunnel, i.e., a sequence of antennas on which it is possible to transfer tags, for verifying the content of the packets being transferred without opening them, we usually rely on simulators since starting from data observed in a real environment it is possible to verify the movement of the packets across different antennas as well as verify whether the algorithm is behaving as expected. In this scenario, the focus is more on the sequence of events being occurred without considering the quality of the reading. The simulators are developed by us based on our needs. A different example in which it is not possible to rely on simulators is aimed at testing a feature that estimates the distance of a tag by using the rssi (received signal strength indication). In this context, simulating the value of the rssi cannot be accurate since that the rssi received from a RFID tag varies based on the environment, such as the amount of metal and the distance of the operator. It is very complex to simulate this behaviour taking into account a huge number of factors that may influence the overall behaviour of the hardware component. Of course, in this context the test cases being executed manually are not repeatable, and the reproducibility of the test in this context is not required. Very often the outcome of the tests relying on simulators differs from the outcome of the same tests executed on the real

hardware in a real environment. One of the root causes for these differences is the fact that the simulators, of course, cannot simulate all possible factors that may influence the actual behaviour of the system in the real world. Some examples are: the belt travels at a higher speed than the one assumed by the simulator; the packet contains a huge number of tags with some of them being not readable. At the moment, the version of the simulators we are using is good enough but the simulator is still evolving.

CI/CD CONFIGURATION

- C2: The pipeline configuration (not 100% sure) is more or less fixed, with configurations adapted based on the languages, the rules to fulfil (how cantata has been configured) for certifications, and the type of tests to be executed within the pipeline. We do not rely on nightly builds, even the slow builds are continuously built. The build execution time is around 1 hour if we include the testing. We typically calibrate the pipeline duration on the time needed for executing the whole set of tests to be run. Each project has a specific build duration that is acceptable for the project. Nightly builds are usually applied in IT contexts like banking applications. For what concerns the level of automation, for the real time operating system everything is almost automated, with the exception for the analysis of the results that may be automated.

- C3: The pipeline changes based on the type of the project. One of the goals is trying to standardize the procedure. So, in this case, something aimed at configuring the pipeline automatically may help. We have some build execution time constraints (5 minutes for each test). For instance, if the build (test) execution time overcomes a specified threshold it is likely that the simulator has encountered other problems like a memory leak that have nothing to do with the software under tests. We have never encountered cases where the problems in terms of stalled builds (long running builds) are related to how the CI/CD pipeline has been set.

- C4: We only test the whole project each time there is a change in one of the 100 packages being included in the application. We just build changes. We do not necessarily change the configuration. We have a fixed configuration so also the environments for building do not change. We do not have to deploy over different environments since we have full control over the environment considering that we sell the robot. Specifically, we have different building blocks for different branches but each branch has a fixed configuration that we do not change over time. Flaky behaviour within the pipeline: yes, and there are two main reasons for that. First of all, our build servers are virtualized so it's kind of reproducible, except for timing used for running unit testing. However, for me this is more a deficiency of the test rather than issues with the infrastructure behind. In simulation testing, it is very hard to guarantee that the resources are always the same, introducing false negatives in the simulation. The most important root cause we experienced is related to the load on the server side. Dealing with flakiness: of course, we have some retry for network issues, Ros uses a GitHub repo for dependency resolution so when GitHub or the repo are down our build jobs will fail due to the impossibility of resolving

dependencies. In case of resources problems, we do not have retries but the pipeline maintainers are able to open issues aimed at solving the problem.

- C5: Resources for the hardware devices (hardware test tracks and test beds as real trains) represent an issue for us. We have a limited number of test tracks. We do not have to change the CI/CD configuration quite often, but software is changing and we need to adapt the configuration (minor changes). We do not have timing-related properties in terms of CI/CD pipeline configurations. Flakiness: we have seen flakiness during our testing executions. This is complex because most of the cases are related to external tools that may behave differently (e.g., virtual machines behaving differently). Another issue is related to the number of virtual machines running on the same server. For instance, there are cases where the system behaves differently based on the load on the server. The misbehaviours are reported back to the integration team responsible for the Jenkins configuration with the goal of finding a solution.

- C6: Rolling build is a sort of incremental build based on impact analysis while nightly builds are full builds. The key decision for rolling build is about which kind of tests to run. Flakiness: it's a huge topic, indeed it is not only the network but the complexity of our subsystems whose features interact across many indirections that may lead to non-deterministic behaviour. This always shows up in the pipeline and because we have strict rules you cannot progress in the integration pipeline without testing being green so red tests are announced, especially the flaky ones. The latter are mainly highlighted by considering specific metrics (such as number and age of tests) that are compared among subsequent build executions. It's quite a pain due to a multiplication effect: if in the integration pipeline, at the system level, you see ten flaky tests it can be than one team may experience around 30 flaky tests and if the root causes for the flakiness are different than it is problematic and effort prone to properly address them. Address flakiness: The actions are mainly determined by looking at the logs and the metrics being outputted from the build execution. Retry is the most obvious action. There are also cases where you need to reformulate the tests, but more interesting there are also cases where you try to address the code causing the flaky behaviour, in order to not experience it anymore in the system.

- C7: The overall build execution time made us organize the system in modules since the entire system testing takes much, much long. Just for the virtual machine, you may need between half an hour to an hour to fully compile. So it's a huge bill. Because it is not just the virtual machine. It's also our builder, which is a kind of a virtual machine creator: it basically takes a Java program and makes an executable out of it. Module won't take very long but the whole system takes very long. And that is just the start, then we have all the various tests. We have one thousand Java APIs that we have to test regularly, and OSGi adds more to that. So, it is a lot easier to test the modules, because they are small and have a very well-defined Java implementation in that system making the average build execution time in the order of ten minutes. We have a kind of incremental build: first we have the virtual machine creation, then we have the OSGi creation that goes on the platform, and then we test individual modules. We never do a whole build involving all the modules because that is not how we deploy; we deploy individual bundles to a platform. Flakiness can happen.

During testing you need to understand what your sensors are sensing and what the acceptable range of inputs are and emulate your system based on those sensors data. For what concerns flaky connections, you have to be concerned about missed messages and retries. In cyber physical systems you want to have some guarantees about data collection. There are architectural ways to deal with that, like with mailboxes, so that if some sensor does not update on time, you still can make a relatively informed decision. But even then, you have to make sure that the drift is not over a certain size, because then you cannot make reasonable decisions anymore. Monitoring of the system: we have the results of each run so we can look at and see the history. Jenkins provides a good facility for this. As far as deployed systems, we have things like logging and debugging interfaces and other things that we can use to monitor the system while running but this kind of application depends on what you want to monitor and why because it is always a performance versus utility issue. The challenge is that monitoring becomes invasive with respect to the system performance.

- C9: We have flakiness during test executions as a consequence of the environment in which we are working. Based on the type of the tests, we have cases where the oracle is done at a higher level of granularity so it is able to properly deal with flaky behaviour. However, we also have cases where it is required to re-run the test multiple times. Monitoring after deploy: we have in place a system that is able to monitor both our platform and the devices of our customers so that in presence of anomalies behaviour or in presence of errors, based on the gravity of the problem being encountered we are contacted directly by mail or telegram so we are aware of the presence of some misbehaviour. However, before starting to look at the problem being signalled, we wait for the problem being raised from our customer.

CHALLENGES AND BENEFITS

- C2: Challenges in CI/CD for CPS: Interaction with the hardware itself (the board). We had to take into consideration the BSPs. How to retrieve information to gather the results. In the beginning it was difficult to monitor the execution because we have to continuously improve the BSPs once deployed: deploy without errors and collect results with a minimum impact on the scheduling. Since it is a real time operating system: retrieving information has for sure an impact on the scheduling of the application. With regards to the simulation before deployment, the main problem was invalidating the results: in presence of an error, discriminating whether the error cause was in the pipeline (problems in the configuration) or in the system itself (simulator) was difficult. One of the main challenges is collecting feedback from the hardware once the application is deployed together with the assessment of the results being obtained. The detection of errors in the deployment is also a challenge, especially if the deployment is done incrementally: you deploy blocks, if there is an error in one of the blocks detecting it and reconfigure and reset the pipeline is a problem. You need to guarantee stable environments once deployed. The incremental deploy may introduce flakiness in the system since it is difficult for us to discriminate why a component/block is not deployed in a proper way due to the fact that with the same configuration sometimes it deploys correctly and some other times it fails. Another challenge in obtaining a highly automatized pipeline

is related to the complexity of integrating within the pipeline the execution of non-functional testing and system testing. Another open issue is selecting the subset of test cases that have to be included in the pipeline that at the moment is completely done manually. An approach that helps us in properly determining the type and number of tests to be included based on a specific change may help us in obtaining faster feedback. One of the benefits we obtained is reducing the number of human errors, and the automation of the process itself.

- C3: The development environment (SDVE Software Development and Validation Environment) is a virtual machine with a set of applications (instruments) that a developer and a tester may use during the development process. At the moment, the first technological problem is trying to integrate the SDVE with the execution environment (network problems due to security issues). So, the main challenge is trying to automate the process including both the compilation and the execution of the test cases. We are not experts in the CI/CD process so one challenge is also related to the culture and lack of knowledge. The clear benefit is related to the reduction of the execution time needed for running out the whole execution of the tests (mainly due to the automation). There are cases where the simulators are not accessible from the outside. One big challenge is that we need to guarantee the protection of the source code: How to test a component without having the production code of the component? An interesting possibility is to automatically configure the pipeline to reduce time and effort. Actually, the pipeline does not have to change/evolve based on the changes in the technologies being used (version for compilers and or programming languages) since that the aerospace domain follows the waterfall process: the tests are executed only after having completed the development task. So, everything is freezed up: no changes may occur later on in the process.

- C4: The challenges start when the Ops start to play a role. Testing is a big challenge especially due to the HiL. Gathering information back from the robot to close the DevOps feedback. We want to incorporate learning from the field back in the DevOps cycle. Closing the circle. The most challenging part to be fully automated: testing with HiL in the pipeline in terms of regression testing. We want to have a dedicated robot somewhere supervised with cameras and running the tests in it without human intervention. In the above scenario, we also face a challenge dealing with how we get data back from the robot. Differently from other software applications, there is data that we cannot control (we cannot see from the data) such as the presence of something on the floor that the robot is not able to perceive so it will fail. You have to analyse the video data and this is very hard. Benefits: level of automation, developers do not have to struggle with configuration issues so they only have to commit their code and that's it.

- C5: Challenges: Our solution is not cloud-based. We have a fog solution with our own server so we do not have redundancy so in the presence of network issues or server issues we are totally black and this is affecting everyone. We do not have all the computer resources we need. To make it more effective we want to deploy as many version machines as possible but today we are limited on that. Another challenge is related to configuring the architecture of the train that

is different between trains: We can almost never copy-paste software that has to run on different train architectures. Automatically deploy the software on the hardware test tracks: this is part of our next step, together with extending the test execution to find errors as soon as possible in the software (delay the defects discovery will cost a lot of money). The majority of the tests we are doing manually at the moment, should be automated. Testers and Developers are in separate teams in presence of new functionalities to be implemented both start together to implement and write test cases. Benefits: we can deliver faster with higher quality, as well as, verify changes faster compared to before. We had a weekly release build, but today relying on the pipeline we can do more than one release per day if we want to. Before, a release took almost one week.

- C6: Problems for setting a CI/CD pipeline for CPS. The biggest problem on a KPI level is cycle time. Three years ago, the cycle time was six weeks (deploy new stuff) while now we could do it every day. It is still not enough from a developer perspective because the feedback is not fast enough. So, we are at the point in which we have a lot of automated tests but if you look at a nightly run it always needs more than one night to complete. So, when developers come to work, jobs are still running. We have long builds mainly running over the night but at the same time we still have rolling continuous buildings but we spent a lot of money on infrastructure to get them going. This was another biggest challenge we needed to address: If you translate test strategies to hardware it is very demanding. We have a lot of real scanners, the same available to the customers and we also have parts of it that can be used for testing in isolation a single module/component. These costs and does not scale like for instance as Kubernetes. The main problems are not the builds themself but our test execution. The build lasts around 15 minutes while the testing infrastructure kills us in terms of test execution time. We have an initiative that pushes testability so we have workflow supported services: 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. So, for what concerns rolling builds we try to limit the amount of testing being executed in them to be as fast as possible. We do impact based testing to figure out the impact of the changes and select the tests to be executed based on the impact. (As minimal as possible). For this reason, we encourage small check in since small check-ins translate in small changes and then small sets of tests to be executed. We have 20 test machines in parallel for managing the overall test size especially for nightly builds. We want to achieve version vector subsystem integration. It's a huge pain that we do not reuse artefacts. We made huge improvements over there but we are not there yet. Another problem we have is related to the fact that deployment to real scanners comes late in the process. Everybody knows that they are developing for a CT but they are sitting in front of their PCs and not a scanner under their table where they test permanently. Our main issue is that if we do not have what I just said we will have big changes, a big number of changes down the line and then we deploy on the system. So, the main issue is that we will not be able to run the software on the system because the installation even does not work on the system, because the update/upgrade does not work, or because the system behaviour is not being considered in the early stages of development. We actually have the problem of showing the benefits of deployment on the system itself because the other ones say you never

get there, you tell us you want to give us feedback but you cannot even install. Yes, we cannot install because the software is not capable of being continuously installed. We cannot deploy at the moment because a change in the security configuration of the software prevented our standard isolation process. Biggest benefits: Unfortunately (or fortunately) testing the installation procedure. Historically the installation process was a simple PowerShell script with no tests. Continuous testing of what we have especially for what we cannot deeper test internally.

- C7: The main difficulty I see is the need for a well-integrated requirements management system. Systems like Doors are nice for one off projects but they do not really work well with revision control. The idea that your requirements are in a database is old-fashioned. If you look at what Eclipse can do now with dynamically analysing code to give you feedback for developers, there is no reason why you could not do that with requirements as well. For instance, you keep requirements in a text form, use something to identify them uniquely, and then automatically pull them into your tool and use that for tracing for testing. This would be a better way of managing requirements. For what concerns CPSs the traceability problem is more evident since we care more about requirements engineering, whereas in web-based development, nobody gives a damn. Looking at the whole free software, there is nobody who does requirements engineering because it is too hard. They only do where the government says you have to do this. So, requirements engineering is a general problem, but only for cyber physical systems, people care about it. Also, for the testing phase, it is very important to trace requirements to code to tests. Everybody does regression testing but only few people do requirement-based testing which is required for every certification regime out there. How do you envision requirement-traceability being integrated in a DevOps pipeline? Every time you make a change you must have a traceability to the requirement that drove that change. In other words, every time you make a change impacting the traceability you must have a requirement change that drives that change in the code. And in this context, by using test you might be sure that the applied change is appropriate, i.e., does not violate guidelines and standards written as requirements. Good formal testing tools are lacking together with requirement engineering and guaranteeing traceability between requirement, code changes and tests. Another challenge is that for the moment the oracle has to be determined manually even if it can be checked automatically.

- C8: Our main needs at the moment are automating the testing, as well as, having simulators that can help us also for complex scenarios.

- C9: The setting of a CI/CD pipeline in the mobile context has been very easy while we are scared about setting a CI/CD pipeline that has to deal with hardware components in terms of complexity and required time. The main challenge we see is related to how it is possible to interface the software with the hardware within the pipeline. For now, the integration testing phase related to the software that interfaces with hardware components is quite complex so we envision the need of using simulators but for what concerns the simulation for the RFID we think that the simulation will not give us any benefits due to their unpredictable behaviour. Another big barrier is related to the test case execution

that at the moment we are doing manually since that for both the environment setting and the oracle definition require manual intervention. A challenge we see is related to the need for adapting the execution of some test cases that require a specific configuration or can be executed only under specific circumstances. The other challenge is related to having a fully automated deployment over the customers' server in which it is possible to have full control on what is going on and try to identify as soon as possible failures/errors occurring during the deployment. Another challenge is related to automatically identifying the set of test cases that have to be run based on the type of change being applied. The pipeline has to be customized based on the type of changes being applied to the system. Obviously, another challenge is related to a lack of a deeper knowledge in the CI/CD context for CPS. The main benefits we expect are: reduce the number of manual-tasks that are more error prone together with increasing the release cycle of our products.