Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 142 (2026) 734–739

www.elsevier.com/locate/procedia

36th CIRP Design Conference (CIRP Design 2026)

Continuous Integration and Deployment in Manufacturing Process Design: A Use Case for Safe and Secure Human-Robot Interaction

Bernd Hader[a,*], Baris Tekin[a], Sebastian Schlund[a]

[a]*TU Wien, Institute of Management Science, Theresianumgasse 27, 1040 Vienna, Austria*

* Corresponding author. *E-mail address:* bernd.hader@tuwien.ac.at

Abstract

Collaborative robots (cobots) are one possible solution for efficiently automating the trend towards lot size 1 in production. Cobots enable humans and robots to work together without a protective fence. In addition, they can also be reprogrammed relatively easily by non-experts, due to their simple user interfaces. Nevertheless, reprogramming of a robot on the shop floor always results in a certain amount of downtime. One possible solution to reduce or even eliminate this downtime is provided by the continuous integration and deployment approach (CI/CD) from software development. The goal of CI/CD is to accelerate software development, to be able to react quickly to changes and to make software development and deployment as efficient as possible. This is implemented by continuously adapting the software, provided as short-cycle software updates. Therefore, automation is also an important aspect of CI/CD, such as automating the testing, the deployment, etc. In the domain of safety-critical industrial systems, CI/CD is currently still in its infancy, especially on the shop floor level. This paper presents a novel use case that illustrates the use of CI/CD for the continuous adaptation of cobot applications on the shop floor, thus reducing the downtime and increasing the production efficiency. However, the use of this technology in safety-critical systems also opens up new attack vectors. Hence, it is also shown how test automation and the open-source tool Jenkins can be used for automatic safety testing and monitoring of cobots to reduce potential safety and security risks. This paper therefore contributes to the research in the field of manufacturing process design by presenting a new application for improving the efficiency, safety and security of frequently changing collaborative human-robot interaction processes.

Keywords: CI/CD; Collaborative Robots; Safety-critical Software Updates; DevOps

1. Introduction

In industry, there is an increasing focus on digitizing production to enhance flexibility and efficiency [1]. An important aspect of this development is the use of collaborative robots (cobots) to enable a partially automated efficient production of small batch sizes together with humans [2]. The reprogramming or adaptation of robot programs can be done directly on the robot by using e.g., teaching by demonstration approaches. The different cobot manufacturers offer manufacturer-specific user interfaces for this purpose, which, as shown in [3], vary in simplicity depending on the manufacturer. The main disadvantages of the teaching by demonstration reprogramming approaches are robot downtime, manual testing and lack of traceability.

One way to overcome these disadvantages is to transfer the principles of Continuous Integration and Deployment (CI/CD) [4] from software development to the field of cobots. This allows frequently changing human-robot processes to be designed more efficiently, which in turn has an impact on the entire manufacturing process design. Therefore, this paper introduces both a conceptual framework and a concrete technical implementation that demonstrates how CI/CD pipelines facilitate continuous cobot program updates and enable the automated execution of safety and security tests on the shop floor.

10.1016/j.procir.2026.05.338

Chapter 2 explains the state of the art in programming cobots, continuous integration and deployment and safety testing of industrial robots using CI/CD tools. Based on this, in Chapter 3 the development and implementation of a novel CI/CD human-robot interaction assembly process in the Pilot Factory at TU Wien, is described. This use case implementation forms the basis for making efficient reprogramming of collaborative robot applications possible, including repeatable and automated safety and security testing of cobot applications. To illustrate this, Chapter 4 presents the implementation of two scenarios for automated, repeatable and traceable safety tests for cobots, based on the use case introduced in Chapter 3. Finally, Chapter 5 contains a summary, conclusions drawn and an outlook on further research topics in this area.

2. State of the Art

2.1. Programming Collaborative Industrial Robots

Collaborative robots offer the possibility of interacting with humans without protective fences, which is made possible by additional collision detection sensors in the robot arms. Furthermore, they can be reprogrammed relatively easily directly on the robot using teach pendants, which enables (partial) automation even for small batch sizes. Current hurdles identified in the literature regarding the reprogramming of industrial robots and cobots are a) the downtime during adoption [5], b) code redundancy when using multiple robots [5], c) the varying intuitiveness of the different user interfaces of the manufacturers [3], d) the multitude of manufacturer-specific programming languages [3, 5], e) the unprofitability of reprogramming for small batches [6] and, f) the legally required repeated risk assessment [7]. Based on the issues identified, it can be concluded that there is a need for new solutions that make the (re)programming of industrial (collaborative) robots more uniform, transparent, intuitive, efficient and economically viable.

2.2. Safety Testing of Industrial Robot Applications

When developing and integrating industrial robots in the European Union, the two standards DIN EN ISO 10218-1:2021 and DIN EN ISO 10218-2:2021 must be considered regarding safety requirements. For collaborative robots there is additionally the ISO/TS 15066:2016, which provides further principles for the implementation of collaborative industrial applications, such as guidelines for evaluating and measuring the collision force. As described in DIN EN ISO 10218-2:2021, Section H, various means of verification and validation must be applied, such as visual inspection, observation during operation, testing of safety-related application software, and/or documentation. The DIN EN ISO 10218-2:2021 also lists verification and validation methods for each protective measure. The necessary risk assessment for the respective industrial robot implementation always serves as the basis for this. The standards refer to the DIN EN ISO 12100:2010, which describes an iterative process for risk assessment and mitigation for machinery. In accordance with the overarching European Machinery Directive [8], any changes to industrial robot applications, regardless of whether hardware- or software-related, must take into account new potential risks and assess and mitigate their effects, meaning that the risk assessment process must be partially repeated [7].

2.3. Continuous Integration and Deployment (CI/CD)

In software development, the V-Model was developed in the 1970s based on the waterfall model [9]. The waterfall model is a linear process model consisting of the five phases: Requirements, Design, Implementation, Verification and Maintenance [10]. The V-Model contrasts each of the five phases with a corresponding test phase [9]. The biggest disadvantage of the V-Model is that late changes can only be incorporated with considerable effort [9, 10].

In order to make software development more flexible, agile methods were introduced in the early 2000s [11]. Based on the principles of the Manifesto for Agile Software Development [11], agile development models such as Scrum [12] emerged, in which working software is developed in modules in 2-4-week sprints [12].

Software that is continuously expanded must be repeatedly built, tested and deployed. In order to increase efficiency and avoid repetitive tasks, the approach of continuous integration and deployment (CI/CD) gained attention at the end of the 2000s. CI/CD aims to automate software integration, testing, delivery and deployment. The three terms continuous integration, continuous delivery and continuous deployment build on each other (see Fig. 1) and are defined as follows [13]:

- Continuous Integration: Agile methods such as Scrum form the basis for continuous integration. Due to the short iterations when using agile methods, there is a constant merging process. CI therefore encompasses planning changes, programming, the build process, testing, and the integration into the product. In the most extreme case continuous integration goes so far that the changes made by a developer are automatically integrated (so-called merged) into the product immediately after the developer releases the code.
- Continuous Delivery: Continuous delivery requires continuous integration and is the process of testing the code after integration. If all tests are successful, the software is released for deployment.
- Continuous Deployment: Continuous deployment is the last step of the CI/CD process. The new software version, which was released in the continuous delivery stage, is automatically transferred to the live environment.

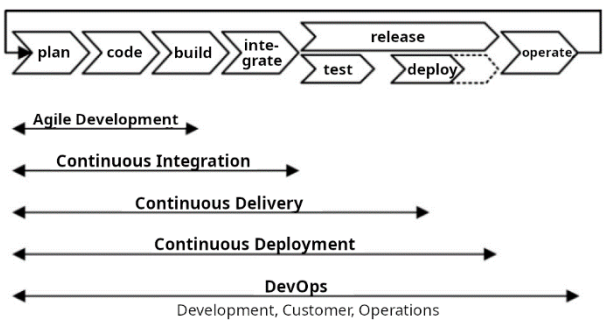


Fig. 1. Connection between agile methods, CI/CD and DevOps [13].

DevOps (see Fig. 1) covers all the elements and phases of CI/CD and goes one step further by also including the operation of the product/service and topics like culture [13]. Through a close cooperation between the development team and the operation team, errors and problems with the product can be resolved and implemented even faster [13].

In addition, there is a further development of DevOps called DevSecOps, which also integrates security into the entire process [14]. The goal of DevSecOps is to develop and improve software in short cycles and to pay particular attention to secure implementation in order to minimize cyber security risks [14].

Automation is essential for CI/CD in software development, which typically relies on two key components:

1. A version control tool for software code, such as Git [15], which enables collaboration and makes software changes traceable.
2. The central component of automation in CI/CD is the CI/CD server, which is responsible for automated deployment, automated testing, etc. Widely used state-of-the-art tools are Jenkins, Gitlab CI/CD, Travis CI, etc.

2.4. CI/CD in the Domain of Industrial Robots

A literature review was conducted to identify relevant scientific contributions dealing with (collaborative) industrial robots and the CI/CD approach. The research area is still relatively new, and very few relevant contributions could be identified in this field.

In the work of Kangas [16], a UR5e collaborative robot from Universal Robots was used for a test process for the quality assurance of keys. The robot was controlled with the help of Python scripts and the CI/CD tool Jenkins was used as test automation tool. Therefore, the focus was not on the continuous adaptation and testing of robot application code, but rather on the automation of quality tests with the help of a cobot and Jenkins.

Schlingloff and Hoppe [17] describe a cloud-based testing approach for cyber-physical systems. Cloud-based testing has the advantage that computationally intensive simulations can be distributed across a large number of standardised computing resources. This allows a large number of test scenarios to be tested multiple times, which would only be possible to a limited extent with a conventional computer. For this purpose, they have implemented a five-stage CI/CD pipeline based on Gitlab, ROS, Gazebo, Python and the Robot Framework to test the functionality of cyber-physical applications in a standardised manner in the cloud.

In contrast, Winkler et al. [18, 19] describe a flexible Testing Automation Framework that uses Jira, Specflow, Jenkins and the Gherkin language for formulating test cases, in the context of behaviour-driven development (BDD), for testing functional requirements of production systems. The use of the Gherkin language, with its specific keywords, allows test cases to be written in a simple and understandable way, also for non-technical persons. These test cases are then converted into executable source code for the scenario to be tested.

It is striking that there are initial studies dealing with the topic of CI/CD in the context of industrial robotics. However, these works focus on specific individual components, such as test automation by using natural languages, the use of cobots for test automation, etc. None of the studies found consider the use of the CI/CD approach for the continuous modification of human-robot applications to improve the efficiency of frequently changing robot-assisted assembly processes directly on the shop floor. Based on this identified gap, this work addresses the following research question: How can a CI/CD approach be designed to enable continuous, safe, and efficient updates of cobot programs on the shop floor?

3. A Use Case for CI/CD in Human-Robot Interaction

The use case which has been investigated and extended with the CI/CD approach is a collaborative human-robot assembly use case, located in the Pilot Factory Industry 4.0 at TU Wien. As shown in Fig. 2, the application is the mounting of a ski binding on a ski. The cobot, a Universal Robot UR 3 [20], picks the individual parts of the ski binding and places them directly on the ski. The worker checks the correct position, corrects it if necessary and screws the parts together. Since the ski models as well as the ski bindings are continuously expanded and the skis are specially mounted in small lot sizes according to customer requirements, there is a wide range of different combinations, which is constantly being expanded. Accordingly, new programs must be continuously implemented.

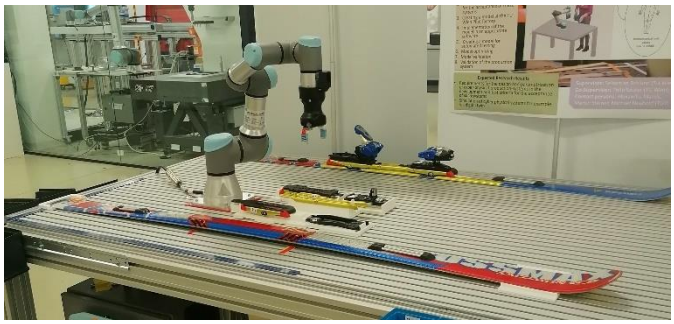


Fig. 2. Collaborative ski binding assembly use case at TU Wien.

The original process was structured in such a way that the integrator has to make each adjustment directly on the robot, using teaching by demonstration. This approach has the major disadvantage that the assembly process has to be interrupted for each adjustment. Depending on the adjustments required and the skills of the employee, this reconfiguration results in a certain downtime of the robot each time - for reprogramming the robot and manual testing. Furthermore, it is also difficult or impossible to determine when, why and by whom changes were made.

3.1. CI/CD Concept for Continuous Modification of Cobot Programs on the Shop Floor

To keep the downtime to a minimum, a new solution for the updating process, using the CI/CD approach from software development, was developed and implemented (see Fig. 3). Each time a new order is received the integrator creates a new program. At the same time, the operator can execute old orders (therefore the downtime of the cobot is reduced). After the integrator has adapted the program, the updated version is up-

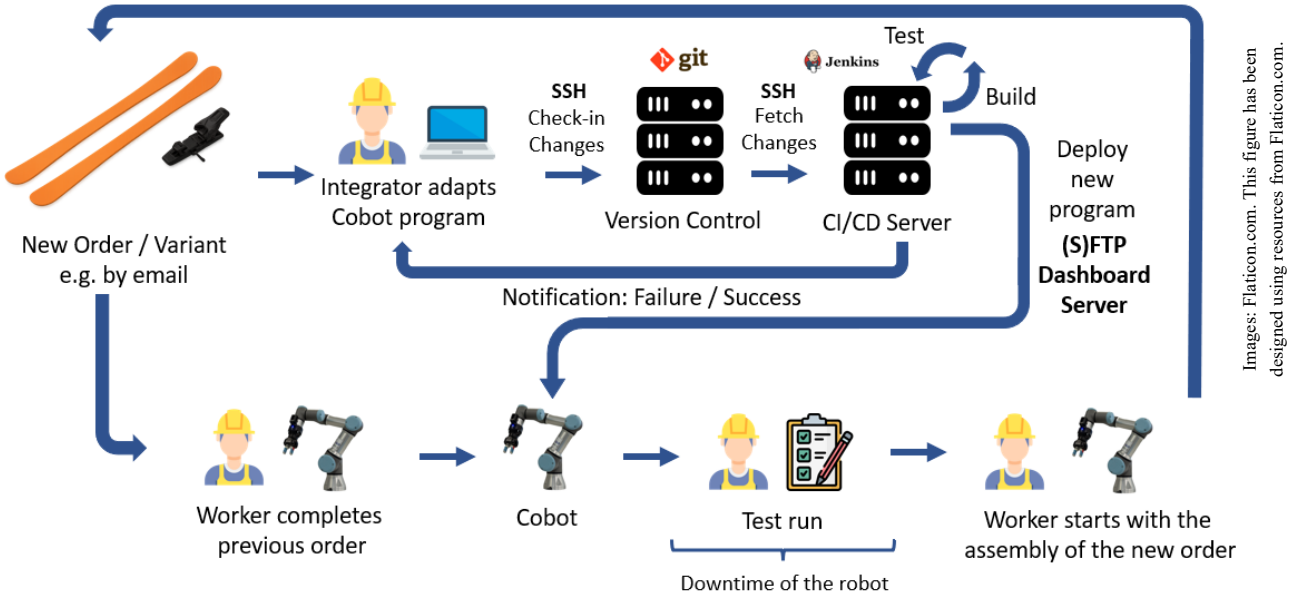


Fig. 3. Technical implementation of the continuous integration and deployment (CI/CD) ski assembly use case at the pilot factory of TU Wien.

loaded using SSH (Secure Shell) into the versioning tool. In our example, Git was utilized as a version control system to manage and track changes in the software development process. Every time a change is made in the version control, the CI/CD server, the open-source tool Jenkins was used in scenario, is automatically triggered. Jenkins fetches the new program version and tests it automatically. This ultimately depends on which automatic tests have been implemented in Jenkins (e.g., security tests, functional tests, safety tests, etc.). Then there are two possibilities:

- If all the tests were successful the integrator receives a message about the success.
- If errors have occurred (e.g., functional errors, safety violations, security issues, etc.) and code changes are therefore necessary, the integrator receives a message detailing the problems. Once these issues have been updated by the integrator, the steps “checking the code into the version control”, “running tests on the CI/CD server”, and “informing the integrator” are repeated.

When the continuous integration process on the CI/CD server has been successful, the new program is ready for deployment. The CI/CD server deploys automatically the new program to the cobot using SFTP (Secure File Transfer Protocol). After completing the previous assembly process, the worker starts the new program on the teach pendant, conducts a test run and subsequently begins executing the updated ski assembly process.

Applying the CI/CD approach at the manufacturing level in human-robot interaction enables integrators to continuously create and update programs and automatically test and deploy them during assembly. This minimizes robot downtime, especially in production processes with small batch sizes and frequent program changes and increases production efficiency.

The novelty of the proposed concept lies in a) combining continuous software development and delivery methodologies with programming cobots on the shop floor, b) enabling efficient reprogramming of cobot applications, and c) the integrating automated safety and security testing for continuous safety assurance.

3.2 Technical Implementation

The technical implementation of the use case and the CI/CD environment mainly consists of the following four components (see Fig. 3):

1. **Updating the cobot program:** RoboDK [21] was used to adapt and create the robot programs via offline programming. RoboDK is a graphical simulation environment for creating programs for industrial robots. It also allows the integration of different manufacturer-specific robots and the conversion of the general RoboDK code into robot-specific code, e.g. code for cobots of Universal Robot, Kuka, etc. It is also possible to transfer code from RoboDK directly to the robot, but this means losing the advantages of automated testing, transparency, and traceability that are enabled by the implementation of the CI/CD approach as shown in Fig. 3.
2. **Version Control:** With the help of a version control software such as Git [15] or Apache Subversion (SVN) [22], the software code is stored in a central location. Changes are traceable, they can be reversed if necessary, and collaboration between multiple developers is made easier. Once a new program has been created, it is transferred to the Git/SVN repository via Secure Shell (SSH). In the example, Git was used as version control system and GitHub [23] as the central cloud repository.
3. **CI/CD Server:** The free open-source software Jenkins was used as the CI/CD server. Jenkins was deployed on a laptop using Docker. A CI/CD pipeline was then configured, which is triggered when changes are made in the GitHub repository. The pipeline retrieves the current code (the next assembly program for the cobot) via SSH and transfers it to the cobot. No test cases were integrated in the first step. This was done afterwards and is described in Chapter 4.

4. **Cobot:** A UR3 from Universal Robots [20] was used as collaborative robot in this scenario. Universal Robots offers several options for the teleoperation of their robots. Based on the classification of Cobas [24], interfaces for remote maintenance are required, to deploy application code remotely on the robot. There are two protocols in the case of Universal Robots available: SFTP and SSH, for the implementation of the use case SFTP was used.

4. Continuous Safety Testing of Cobots with CI/CD

Based on the legal requirements of the Machinery Directive [8], according to which the risk assessment must be repeated for all changes to industrial cobot applications, including verification and validation of the software, the CI/CD approach also offers savings potential here. For better illustration, the following two safety test cases have been implemented as examples in the Pilot Factory.

4.1. Safety Configuration Testing

Parameters such as the maximum permissible speed or the collision force of the robot are defined in the safety configuration on the robot. This safety configuration could, for example, be manipulated by an attacker. With the help of the CI/CD server, it is possible to introduce an additional safety and security layer which, on the one hand, checks whether the current safety configuration is correct whenever the robot program is updated. On the other hand, it is also possible to continuously test at regular intervals whether the configuration has been changed. The technical implementation with the CI/CD tool Jenkins looks as follows. The currently valid safety configuration was also stored in the version management tool with special user access and modification restrictions. The version management system thus makes it possible to track which safety configuration is currently valid at any given time. This safety configuration is compared with the current configuration on the robot when changes are made to the application software, but also at regular intervals. If there are any deviations, the test fails (see Fig. 4) and the integrator is informed.

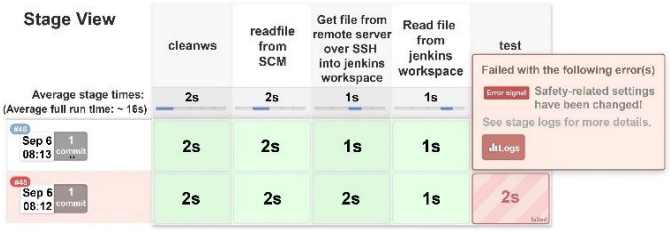
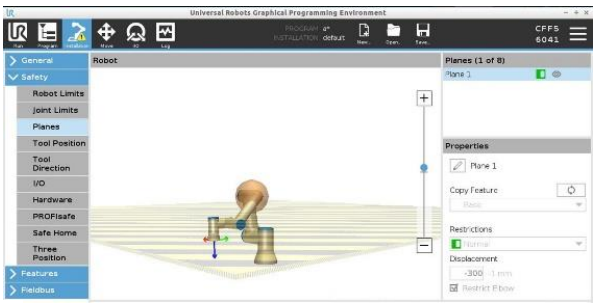


Fig. 4. Pipeline for safety configuration testing of Universal Robots implemented with the CI/CD tool Jenkins.

4.2. Automated Safety Plane Simulation Testing

Virtual safety planes (see Fig. 5a) are a measure to increase safety when humans and robots work together. They can be used to define zones in which a) the robot is allowed to move at high speed, b) at reduced speed, and c) in which the robot is not allowed to move. If the robot reaches a restricted zone, the robot must stop. In the implemented test case scenario (see Fig. 5b), every time the robot application code is changed, it is automatically checked whether the modification of the robot's program code is still consistent with the currently valid virtual safety planes.

(a)



(b)

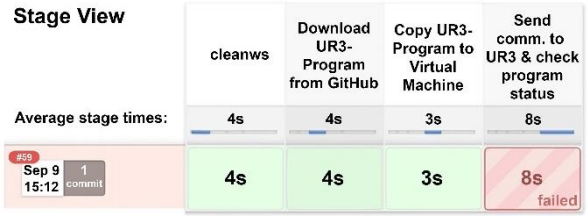


Fig. 5: (a) Functional testing of robot application code with the URSim simulation environment; (b) Pipeline for the automated safety plane simulation testing implemented with the CI/CD tool Jenkins.

This was technically implemented using the CI/CD tool Jenkins and the URSim simulation environment from Universal Robots (see Fig. 5). The CI/CD pipeline in Jenkins was implemented in detail as follows (see Fig. 5b):

1. cleanWs: In the first stage the workspace is cleaned, to have the same environment for each test.
2. Download UR3-Program: In this step the latest version of the robot application code is fetched from the GitHub repository.
3. Copy UR3-Program to Virtual Machine: In this phase the application code is copied to the simulation environment URSim, which has been set up in a virtual machine.
4. Send command to UR3 simulation and check program status: In the last step different commands like: Power on, brake release, etc. are sent to the robot simulation, to ensure that the (simulated) robot is ready for testing. Afterwards, the latest version of the robot application code is executed in the simulation software. As shown in Fig. 5b the pipeline tests whether the program can be executed without any issues. The test was not successful; the reason was a violation with a safety plane, which caused the robot to stop. Finally, the feedback from URSim is forwarded to the integrator.

As outlined in chapters 3 and 4, the integration of CI/CD pipelines goes beyond the automated provision of robot programs and also includes systematic test automation. This approach enables the execution of functional, safety and security tests in an efficient, reproducible and transparent manner. By defining test procedures once and embedding them in the pipeline, subsequent executions achieve a high degree of reliability, traceability, and maintainability, thereby improving both software quality assurance and overall system reliability.

In terms of efficiency gains through reduced downtime, theoretical savings of up to 75% appear achievable, assuming

that 25% of the total effort remains for necessary on-site validation on the physical robot. This residual effort primarily concerns functional testing; specific safety-related collision tests were not considered within the scope of this estimation. Estimates in the literature [25, 26] for reprogramming cobots range from a few minutes to several hours, depending on task complexity, the user interfaces and operator experience. Based on the authors' extensive practical experience in international training programs, it is assumed that a highly experienced user requires an average of 20 minutes for modifications directly on the robot (10 minutes for small, 20 for medium, and 30 for larger changes). Since simulation-based programs often deviate from real-world conditions, minor manual adjustments remain necessary, even with CI/CD approaches. Assuming that this applies in 50% of the cases, efficiency gains of 40–60% on average appear achievable in practice.

5. Conclusion, Discussion and Future Work

This study employed a comprehensive use case to illustrate how principles from Continuous Integration and Continuous Deployment (CI/CD), originally developed in the context of software engineering, can be effectively transferred to the domain of collaborative human–robot processes. Furthermore, two safety-related test cases were defined and integrated into a CI/CD pipeline, demonstrating the feasibility of automating manual testing procedures for collaborative robots. The integration of such automated deployment and testing significantly enhances software quality, improves the traceability of program modifications, supports systematic documentation practices, and ultimately contributes to increased reliability, efficiency and safety in human–robot interaction. In terms of efficiency gains, improvements of 40 - 75% appear to be possible, ultimately depending on the accuracy of the simulation with the real environment. As digital twins become increasingly prevalent in industry [27], requiring reliable and continuous updates, this use case demonstrates how CI/CD also enables the implementation of safe and secure update workflows in safety-related industrial environments.

This study is subject to several limitations. First, the use case was developed as a prototype in the Pilot Factory at TU Wien and has not been tested in real industrial production. Therefore, future work should focus on (a) implementing it in a real industrial production and (b) measuring the actual efficiency gains. Second, only two safety test cases were implemented as proof of concept. To ensure broader applicability and compliance with safety standards, the development of a generic, extensible safety test catalogue is essential, which should cover a wide range of functional and safety-relevant scenarios in collaborative human–robot processes. Third, the solution was implemented on a single robot platform. Further research should explore applicability to other robot systems and identify platform-specific constraints.

Acknowledgements

This paper was supported by TÜV AUSTRIA #SafeSecLab Research Lab for Safety and Security in Industry, a research cooperation between TU Wien and TÜV AUSTRIA.

References

[1] Becker W, Ulrich P, Schmid O, Feichtinger C. Industrielle Digitalisierung: Entwicklungen und Strategien für Mittelständische Unternehmen. Wiesbaden: Springer Gabler; 2020.
[2] Vicentini F. Collaborative Robotics: A Survey. Journal of Mechanical Design 2021; 143(4). doi: 10.1115/1.4046238.
[3] Schmidbauer C, Komenda T, Schlund S. Teaching Cobots in Learning Factories. Procedia Manufacturing 2020.
[4] Jani Y. Implementing Continuous Integration and Continuous Deployment (CI/CD) in Modern Software Development. IJSR 2023; 12(6):2984–7.
[5] Bilancia P, Schmidt J, Raffaeli R, Peruzzini M, Pellicciari M. An Overview of Industrial Robots Control and Programming Approaches. Applied Sciences 2023; 13(4):2582. doi: 10.3390/app13042582.
[6] George P, Cheng C-T, Pang TY, Neville K. Task Complexity and the Skills Dilemma in the Programming and Control of Collaborative Robots for Manufacturing. Applied Sciences 2023; 13(7):4635.
[7] Hosseini AM, Fischer C, Bhole M, Kastner W, Sauter T, Schlund S. A Safety and Security Requirements Management Methodology in Reconfigurable Collaborative Human-Robot Application. In: 2023 IEEE WFCS; 2023. S. 1–8.
[8] European Parliament, Council of the European Union. Directive 2006/42/EC of the European Parliament and of the Council of the European Union: Machinery Directive; 2006.
[9] Adetokunbo A.A. Adenowo, Basirat A. Adenowo. Software Engineering Methodologies: A Review of the Waterfall Model and Object-Oriented Approach. Internat. Journal of Scientific and Engineering Research 2020.
[10] Alazzawi A, Rahmatullah B. A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions. Iraqi Journal for Computer Science and Mathematics 2023:173–90. doi: 10.52866/ijcsm.2023.04.04.014.
[11] Manifesto for Agile Software Development; 2024 [Accessed: 15.08.2025]. Available: https://agilemanifesto.org/.
[12] Ken Schwaber and Jeff Sutherland. The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game. 2020.
[13] Jürgen Halstenberg, Bernd Pfitzinger, Thomas Jestädt. DevOps, Ein Überblick. 1436-3011 2020; 56.
[14] Michelle Ribeiro. Learning DevSecOps: O'Reilly Media, Inc; 2022.
[15] Git; 2025 [Accessed: 21.08.2025]. Available: https://git-scm.com/.
[16] Kangas I. Integration of a robotic arm into test automation: Master Thesis in Mechanical Engineering; University of Oulu; 2023.
[17] Schlingloff B-H, Hoppe N. A Framework for Cloud-based Testing of Multi-variant Cyber-physical Systems. In: 2021 10th Mediterranean Conference on Embedded Computing (MECO); 2021. S. 1–4.
[18] Winkler D, Meixner K, Novak P. Efficient and Flexible Test Automation in Production Systems Engineering. In: Biffl S, Eckhart M, Lüder A, Weippl E, Hrsg. Security and Quality in Cyber-Physical Systems Engineering: Springer, Cham; 2019. S. 215–42.
[19] Winkler D, Meixner K, Biffl S. Towards Flexible and Automated Testing in Production Systems Engineering Projects. In: 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA); 2018. S. 169–76.
[20] Kollaborierende Roboter - Cobots | Universal Robots; 2025 [Accessed: 22.08.2025]. Available: https://www.universal-robots.com/de/.
[21] Simulator für Roboterarme und Offline-Programmierung - RoboDK; 2025 [Accessed: 22.08.2025]. Available: https://robodk.com/de/.
[22] Apache Subversion; 2025 [Accessed: 22.08.2025]. Available: https://subversion.apache.org/.
[23] GitHub. GitHub · Build and ship software on a single, collaborative platform; 2025 [Accessed: 22.08.2025]. Available: https://github.com/.
[24] Sergi Ponsà Cobas. Strategies for remote control and teleoperation of a UR robot: Master Thesis; Barcelona School of Ind. Eng. ETSEIB; 2020.
[25] Giberti H, Abbattista T, Carnevale M, Giagu L, Cristini F. A Methodology for Flexible Implementation of Collaborative Robots in Smart Manufacturing Systems. Robotics 2022; 11:9.
[26] Mandryk R, Hancock M, Perry M, Cox A, Hrsg. Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices. New York, NY, USA: ACM; 2018.
[27] Anwer N, Stark R, Tao F, Erkoyuncu JA. Developing and leveraging digital twins in engineering design. CIRP Annals 2025; 74(2):843–68. doi: 10.1016/j.cirp.2025.05.002.