ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

LEVERAGING IT-INSPIRED WORKFLOWS FOR PLC SOFTWARE

J. M. de Paco*, W. Niinimaeki, M. Marin, O. Minaiev, CERN, Geneva, Switzerland

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Abstract

PLCs play a crucial role in operating, controlling, and interlocking high-power distributed systems at CERN, including main magnet power supplies and static VAR compensators. The increasing number, complexity, scale, and specialization of these critical applications make development and maintenance particularly challenging for a small team. To address this, we have introduced IT-inspired engineering workflows and technologies that wrap around the UNICOS framework, balancing customization with standardization. This approach enhances engineering efficiency and software quality. This paper highlights the benefits of using continuous integration pipelines, automated functional tests, reviewed merge requests, and the necessary virtual computing infrastructure, demonstrating how these innovations streamline development and maintenance in high-power system automation.

INTRODUCTION

At CERN, the Electrical Power Converters (EPC) group is responsible for power converters and Static VAR Compensators (SVCs) [1] across all accelerators and facilities. Its Converters Controls Software (CCS) section develops and maintains PLC-based applications using CERN's standard stack—Siemens S7-1500 PLCs, *UNICOS-CPC* [2], and Siemens WinCC OA SCADA [3]—for the control, interlocking, and operation of high-power systems.

Our PLCs play a crucial role in the control, operations and diagnostics of high power-systems that are critical for the availability and performance of the accelerators. They protect expensive equipment, orchestrate large distributed control systems, interface with other services of the accelerators (power converter regulation, electrical network, cooling plants, magnet protection, access systems, etc.) and provide human machine interface (HMIs) both for machine experts and for CERN's Control Center. The group has adopted in-house PLC-based control systems as the standard for integrating and controlling large distributed power converter systems, which were previously outsourced to industry. This includes the upgrades of the main power converters of the PSB [4], PS [5], and SPS [6], as well as the SVCs.

Challenges and Limitations of Traditional Workflows

With the growing number and complexity of new projects and systems under our responsibility, the team faces significant technical and resource-related challenges. While maintaining and supporting the systems currently in operation, we work in projects to be commissioned during the CERN's Long Shutdown 3 period (2026-2029). These are the challenges we are facing and the limitations we would have if we keep traditional workflows:

- Productivity: Manual builds, uploads, and testing in UNICOS-CPC and TIA Portal are slow, repetitive, and error-prone.
- Quality assurance: Updates to UNICOS-CPC framework and PLC logic modifications risk breaking existing functionality. Limited time during technical stops prevents full manual regression tests.
- Teamwork: Large projects require version control for shared development, code reviews, and traceability. However, traditional automation engineering environments lack the built-in version control and collaborative features that are standard in modern IT software development.
- Commissioning phase: Debugging during commissioning is costly and risky, as engineers must work under pressure within short technical stops to commission multiple machines simultaneously. Any issue at this stage can jeopardize accelerator restart schedules and significantly increase costs.
- Knowledge transfer and resourcing: Expertise is typically concentrated in a few senior engineers. Traditional methods and technologies discourage knowledge transfer and make it harder to retain young engineers with modern IT skills and expectations.

Strategic Plan Towards DevOps in Industrial Automation

To address the listed challenges, the team has strategically adopted IT-inspired technologies and workflows, transitioning from traditional PLC development practices to a modern industrial DevOps framework. DevOps unites software development with IT operations, leveraging automation, collaboration and fast iterative cycles. The core of this methodology are version control and CI/CD. Applying these principles into the context of Operational Technology (OT) aims to solve long-standing challenges related to software quality, efficiency and machine commissioning.

This is a gradual process involving changes in tools, processes, and team culture. In this paper we are presenting this transition in four adoption levels, according to our experience in the previous years. In any case, the process should be adapted to the needs and context of each organization. The four adoption levels are: 1. Version control and code editing; 2. Continuous Integration and Continuous Delivery (CI/CD); 3. Automated testing: virtualization and simulation; 4. Cultural adoption and advanced techniques.

* josemanuel.depaco@cern.ch

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

VERSION CONTROL AND CODE EDITING

Version Control of UNICOS-CPC Projects

Version control is the foundation for CI/CD. Our codebase is managed in *Git* [7] and hosted on CERN's *Gitlab* [8] infrastructure. It provides a single source of truth, enabling history and traceability, parallel development through branching, and quality control via peer-reviewed merge requests.

Git is optimized for managing text-based source files, performing detailed comparisons (diffs), and intelligently merging modifications. However, PLC programming is typically done using graphical languages (such as Ladder, Function Block Diagrams, or Sequential Function Charts). A TIA Portal project comprises the process logic, hardware configuration, and network settings. It is stored as a collection of files that are opaque for *Git*, able to detect that the file has changed but unable to interpret what changed or to merge contributions from different users.

Fortunately UNICOS-CPC was designed in a way that favors version control by generating the TIA Portal project processing a set of external source files. The UNICOS Application Builder (UAB) primarily uses the following text or table based inputs to generate the final TIA Portal project:

- Specification file: An Excel spreadsheet defining the UNICOS object instances (I/O, actuators, alarms, parameters, etc.).
- User sources: SCL files containing custom logic in functions (FCs), function blocks (FBs), and data blocks (DBs).
- Logic templates: Python files containing user-defined logic.
- Base project: The TIA Portal project containing the hardware and network configuration.

The core application logic is already in text-based files (SCL and Python), which are perfectly suited for *Git*. However, some limitations remain: the monolithic TIA Portal base project, the difficulty of tracking changes within the Excel specification file, and some other opaque UAB files.

When we first set up our Git repositories, the UAB project structure did not support sharing common code (like user sources or templates) across different projects, forcing us to duplicate code. For a long time we solved it by placing shared files in external folders and using system-level symbolic links to connect them to individual UAB projects. Responding to this limitation, UNICOS-CPC developers have recently implemented the ability to natively define external paths in UAB projects.

To achieve complete version control we require further evolution. The UNICOS-CPC specification file should ideally be transformed into a versioned database with a modern user interface. We also expect TIA to move towards a more software defined representation of project parameters and hardware configurations, making every aspect of the project fully traceable in *Git* and enabling a true CI/CD pipeline.

Code Editing With Visual Studio Code

Since UNICOS-CPC code development is mostly based on text-based SCL and Python files, we can move away from TIA Portal environment for day-to-day coding. This allows us to adopt powerful, modern code editors like *Visual Studio Code*(*VS Code*) [9] which provides modern IDE features and integrates smoothly with Gitlab workflows, offering a lighter alternative to TIA Portal.

It is important to note that tasks such as hardware configuration, online monitoring, and live debugging still must be performed in TIA Portal.

Version Control of Production Projects

The final step closes the loop between development and operations, ensuring the control logic in *Git* matches the logic deployed in production. Its purpose is to detect unauthorized or undocumented on-site changes, and to have a history of backed-up versions ready to be deployed if a rollback is needed. Currently, these functionalities are only provided by third-party platforms. At CERN, we have adopted *AUVESY-MDT Octoplant* [10]. Besides change management and version history, it can automatically connect to devices in production, upload the running program, and perform a detailed comparison against the last version stored on its server, immediately flagging any discrepancies. It also performs scheduled back-ups of the version running in the PLCs.

CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY (CI/CD)

The next step is to automate the build and validation process using a Continuous Integration/Continuous Delivery (CI/CD) pipeline [11]. In this methodology, developers frequently merge code changes into a shared repository, where each commit automatically triggers a build and test sequence. This provides rapid feedback, catching bugs early and eliminating the error-prone process of large, manual integrations. The pipeline ensures the codebase is always in a production-ready state, with the final deployment to the PLC remaining a deliberate and planned manual action.

Constructing a CI/CD Workflow

Implementing a CI/CD pipeline for Siemens S7-1500 PLC applications requires a hybrid architecture. This approach integrates native Siemens APIs, such as the TIA Openness API [12], with standard DevOps tools and advanced virtualized testing environments. We selected *GitLab CI* [11] due to its seamless integration with our *GitLab* source code repository. In *GitLab CI*, pipelines (comprising stages and jobs) are defined in a YAML configuration file (.gitlab-ci.yml) that we located alongside our UAB project, ensuring that the automation workflow is version controlled and reproducible.

The execution of pipeline jobs is handled by *GitLab* Runners. Our implementation uses runners hosted on Windows Virtual Machines within CERN's OpenStack cloud infrastructure [13]. These VMs are configured with all necessary

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Siemens software, including TIA Portal and TIA Openness, creating a consistent and reproducible build environment that eliminates the classic "it works on my computer" problem. Pipeline execution is highly flexible. By default, pipelines are triggered automatically on every git push to a branch, ensuring immediate feedback on code changes. Pipelines are always run as part of a merge request and need to pass in order to approve it. Additionally, they can be configured for scheduled runs (e.g., weekly for regression testing) or initiated manually through the *GitLab* web interface which allows a user to select specific applications and target PLCs for a given run. Our pipelines (see Fig. 1) follow these stages, defined in the project's .gitlab-ci.yml file:

Build The build stage is initiated by a Python script that orchestrates the entire process. This script leverages *alpaka* [14], a Python package developed at CERN that acts as a wrapper around the UNICOS Application Builder (UAB). The *alpaka* package uses *Apache Maven* [15] to drive the UAB generation process in three steps: 1. Generation of UNICOS-CPC source files (instances and logic); 2. Importation of source files into TIA Portal project; 3. Build and compile TIA Portal project into a deployable artifact.

Both the import and compile steps use a custom CERN-developed wrapper around the Siemens TIA Openness API. This service allows our Python scripts to programmatically interface the otherwise closed TIA Portal environment. If the build fails, for instance due to a compilation error, the pipeline stops and notifies the developer. The error log is available in *GitLab CI* for rapid debugging.

Deploy Once successfully built, the TIA Portal project is deployed to a virtual PLC (a Siemens *S7-PLCSIM Advanced* instance). This step is transformative, as it allows us to run comprehensive tests without depending on limited and expensive physical hardware. The deployment script uses CERN's TIA Portal API wrapper to manage the virtual PLC (reset, start, stop, upload). While the pipeline always targets a virtual PLC, developers can use the same script from their terminal to deploy to physical PLCs in the lab. The pipeline fails if the deployment is unsuccessful.

Setup The setup stage prepares the software for testing by bringing the system to a known initial state. This involves interacting with UNICOS-CPC objects to simulate initial plant conditions (e.g., clearing alarms, homing actuators, or setting initial process values). This ensures that tests are repeatable and not influenced by the state left over from a previous run.

Tests An automated set of unit, integration, and regression tests are executed against the virtual PLC. During this stage, a failed test does not stop the testing process; all tests are run to completion to provide a comprehensive report of all failures. However, the pipeline as a whole will only be marked as successful if every single test passes.

Release This optional stage implements the Continuous Delivery part of our pipeline. When triggered, the validated TIA Portal project is saved in *GitLab CI* as a versioned build artifact. This artifact is the production ready deliverable to be transferred to our *Octoplant* repository for a scheduled, manual deployment.

Report This final stage provides feedback to the developer. The complete terminal output from all previous stages is logged and attached to the pipeline run in *GitLab CI*. Test results are summarized, allowing developers to easily identify the cause of any failure and fix it quickly.

A FRAMEWORK FOR AUTOMATED PLC SOFTWARE VALIDATION

Paradigm Shift in PLC Software Validation

Limitations of traditional manual testing Manual testing in the context of PLC software involves a control systems engineer executing a series of predefined test cases by hand. This process requires manipulating physical inputs or forcing values, observing the system's response on an HMI, and visually comparing the outcome against expected results. This approach depends heavily on the experience of the engineer, is prone to human error and is inconsistent. In an industrial setting, an incorrectly checked value can have severe consequences. Furthermore, manual testing is time consuming. The nature of PLC logic requires testing numerous combinations of inputs and states. Executing these tests manually for every software version is a slow, monotonous task that scales poorly.

After every change, even small ones, regression tests should be performed to ensure that the modification has not broken a validated functionality. Manually executing these tests is prohibitively long and expensive. At CERN, machine time during technical stops is very limited, and having the necessary environment conditions to re-commission a system is difficult.

Advantages of automated testing Automated testing addresses the weaknesses of the manual approach by leveraging IT tools to execute tests and validate results. The main advantages are speed, comprehensiveness, reliability, and accuracy. Test scripts execute the same sequence of operations every time they are run. This repeatability is a cornerstone of quality assurance. Automated tests can run unattended overnight, and can execute thousands of test cases, reporting the results to the developers who can fix the mistakes early in the development process when the cost is the lowest. Automation also enables an increase in test coverage. Wider tests sets and more complex test scenarios can reveal hidden defects. While automated testing requires an initial investment in infrastructure, software tools and the development of test scripts, it is more cost-effective in the long term. The upfront cost is an investment that pays dividends over the entire lifecycle of the system.

A balanced strategy Automated testing does not replace manual testing. Automated testing excels in scenarios that are repetitive, deterministic and data-driven. However

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002



Figure 1: Successful CI pipeline for BEQ1 project. The pipeline includes steps for Build, Deployment, Setup, and tests for Digital Alarms, Analog Alarms, OnOff Objects and the Grafcet.

there are scenarios that demand human cognition, intuition and domain expertise. Manual testing remains indispensable for hardware tests, exploratory testing, and HMI usability testing. The time freed by automated testing during commissioning, can be used on high-value activities. At CERN, during busy technical stops, this efficiency will allow engineers to share their expertise across multiple systems simultaneously.

Virtualization

The technology that enables modern PLC software testing is the virtual controller, a software simulation of the physical PLC hardware. It breaks the long-standing and expensive dependency on physical PLC test benches. *S7-PLCSIM Advanced* [16] is a Siemens product that creates one or more virtual S7-1500 controllers on a standard PC. A key feature is its ability to create a virtual Ethernet adapter on the host PC, allowing each virtual PLC instance to be assigned an IP address and communicate over the network just like a physical controller. This is a game-changer for testing, since a virtual PLC can communicate with real HMI and SCADA systems, other PLCs (real or virtual) and, most importantly, external test scripts.

Our team uses CERN's OpenStack cloud infrastructure to create Windows virtual machines (VMs) that host our *S7-PLCSIM Advanced* instances. Each virtual PLC is assigned a fixed IP address on the CERN network, making it reachable from our *GitLab CI* runners and from developers computers. For each application, we maintain a dedicated set of instances: one for pipelines, another for shared development, and a third for a production virtual environment. This environment consists of a software process simulator controlled by the virtual PLC and a WinCC OA SCADA project, used for HMI development and operator training. This on demand, scalable infrastructure is significantly cheaper than a physical equivalent, which we estimate would cost at least 150 kCHF.

Simulation

Process simulation is the basis for automated system testing. It can be performed using two techniques: *Software-In-The-Loop* and *Hardware-In-The-Loop*. These are not mutually exclusive choices but rather two levels of validation during a project's life cycle.

Software-In-the-Loop (SIL) SIL is a method for testing software using a virtual simulated environment. The PLC program runs on an *S7-PLCSIM Advanced* virtual controller, and the controlled process is simulated by another software. In our case, we have developed a Python simulator which implements a set of classes that mimic the devices in our systems, such as power converters, circuit breakers, switches, and regulators. The simulator is created dynamically by parsing the UNICOS specification file, enabling it to communicate with the PLC's I/O via an OPC-UA client. The architecture is depicted in Fig. 2.

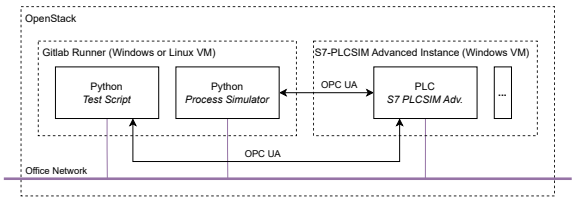


Figure 2: Implemented SIL testing architecture.

This technique allows us to debug and test logic before any physical hardware is assembled. It decouples software development from hardware procurement and installation, enabling these activities to progress in parallel. This shortens the project timeline and shifts debugging from on-site commissioning to the much cheaper development phase.

Hardware-In-The-Loop (HIL) In a HIL setup, the real physical PLC (the "hardware" in the loop) is connected to a real-time computer that runs an accurate dynamic model of the plant. This simulator provides real signals to the PLC's physical IO modules, and the PLC, in turn, sends actuator commands back to the simulator, closing the control loop. HIL's main advantage is its ability to test real-time performance and physical I/O behavior that SIL cannot capture. The HIL platform can be used to run automated test scripts, which can be run by a CI/CD pipeline or executed by a human in the terminal. The HIL can also be used for the validation of the HMI or even for operator training, being a complete replica of the real machine. The benefits of HIL testing are immense. It allows us to fully validate the control system under realistic dynamic conditions, without any risk to expensive physical equipment or personnel.

We are using HIL in the POPS+ project [17]. The power converter system is modeled in detail using Matlab Simulink and simulated in real-time on a Speedgoat HIL platform. The distributed control system is composed of an S7-1500 PLC and multiple FGC4 [18] controllers that communicate via a custom real-time Ethernet protocol. The HIL platform is indispensable for integrating and validating this heterogeneous system.

Implementation of Automated Functional Tests

Testing framework In our CI/CD setup the automated tests are run as stages in a *GitLab CI* pipeline. The built

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

PLC application is deployed to an *S7-PLCSIM Advanced* instance, the process state is initialized (setup stage), and finally, a set of Python test scripts is executed by a *GitLab CI* runner. Developers can also run these scripts on their desktop computers against any virtual or physical PLC.

The implementation is based on a custom Python framework named *pypowerplc*. This framework provides a simple, high-level API for interacting with UNICOS-CPC objects. It is built on top of the standard UNICOS package (*pyplctest*), which handles the OPC-UA client communication to set UNICOS-CPC object attributes. *pypowerplc* extends this functionality by providing object-oriented representations of UNICOS-CPC objects with high-level methods that streamline test script development. For instance, the digital alarm class includes methods to activate or deactivate the alarm according to defined boolean logic. Additional methods enable verification of interlocks on child actuators, including the type of interlock applied (e.g., temporary stop, full stop, start interlock, or alarm warning).

Test scripts developed with *pypowerplc* employ *pytest* [19] to assert expected outcomes. *pytest* is a widely used open-source Python framework for automated testing.

Specification-driven functional testing In order to integrate the test specification into the CI/CD workflow, we have adopted a *specification-driven testing* methodology. Instead of manually writing a test script for each alarm and interlock, our framework automatically generates tests from a specification file. This file is completed by the test engineer based on the functional specification document provided by the power converter specialist. It formally defines the expected behavior of the control logic.

This technique is currently applied to functionalities that do not require a dynamic process simulator:

- Digital Alarms (Fig. 3): The test script parses the alarm's boolean logic from the specification, generates a computed truth table which optimizes the test cases to keep 100% coverage (see Test case optimizer), and then simulates each combination to assert that the alarm activates correctly. It also asserts that the correct interlocks are triggered at the specified level.

- Analog Alarms: The script reads the alarm thresholds and then simulates the analog input signal at various values (below, between, and above the thresholds) to assert correct activation and interlocking.

- OnOff: For simple actuators whose state is determined by a boolean combination of variables. It follows the same procedure as Digital Alarms for generating test cases.

For more complex, sequential operations controlled by Grafcets, tests are specified in a JSON file. A generic Python test template parses this file and navigates every step of the Grafcet, using the SIL process simulator to provide the necessary feedback signals. It asserts that the correct actuators

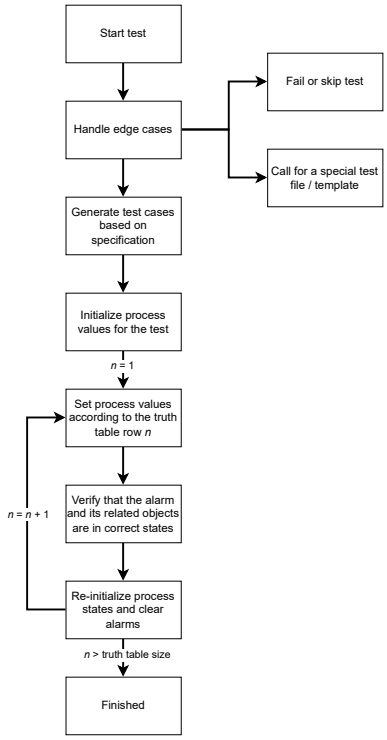


Figure 3: Digital Alarm test flowchart.

are engaged at each step, ensuring the entire sequence operates as designed. Grafcet transitions are tested as OnOff objects.

Currently, the specification is maintained in an Excel spreadsheet. To improve usability, traceability and formalization, we are developing a web-based interface and database.

Customized and Complex Functional Tests

Specification-driven testing covers a majority of our standard test cases. However some unique functionalities require custom scripts. To streamline the creation of these complex test sequences and make them easier to read and maintain, we have implemented a YAML-based testing format, as shown in Fig. 4. In this approach, a test engineer defines the sequence of actions and expected outcomes in a structured YAML file. A generic pytest script then parses this file and executes the steps using our *pypowerplc* framework. Depending on the test complexity and the engineer's expertise, tests can either be defined in YAML or implemented directly in Python.

System Functional Tests

The previously described tests are valid for standalone, single CPU applications. Modern control systems are often distributed, comprising multiple PLCs or other controllers. To solve this, we have extended the capabilities of our test framework to connect to and orchestrate tests across multiple controllers simultaneously (either virtual or physical).

As a proof of concept, we have developed a set of system level tests to validate the Control and Interlock System (CIS) of the SPS main power converters using a dedicated HIL

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

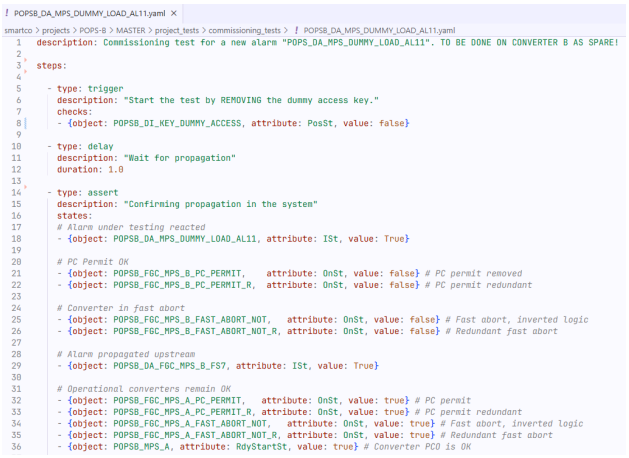


Figure 4: Partial screenshot of a test defined in YAML format, showing Trigger, Delay and Assert steps.

platform (Fig. 5). A customized Python test script opens connections with all PLCs and controllers in the HIL platform, runs a defined sequence of operations, and asserts that the entire distributed system behaves as expected. These tests are useful for detecting integration issues, timing problems, and protocol errors that are invisible at the individual controller level.

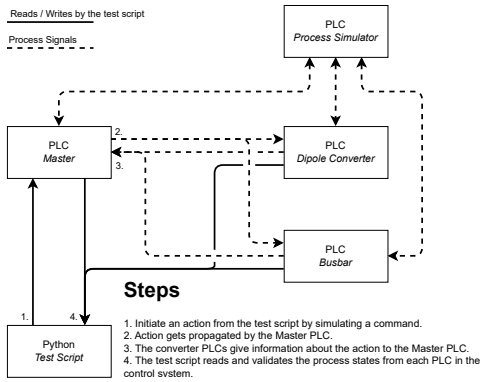


Figure 5: System tests in Mini-CIS HIL platform.

The system tests of the CIS complement the automated functional tests performed in the CI/CD pipelines of each individual PLC. System functional tests performed on HIL platforms can also be fully automated and integrated into scheduled CI/CD pipelines, providing continuous validation of the entire distributed system.

Test reports and feedback The final step of any automated test is a clear report that provides immediate feedback to the developer. The results of our tests are logged in the developer's terminal for manual execution and available directly in the GitLab CI/CD interface for pipelines. When a test fails, the report provides not just a simple pass/fail result but also a detailed trace of what went wrong. This includes the specific assertion that failed, the expected versus actual values, as depicted in Fig. 6. This information is essential for developers to debug and resolve issues quickly.

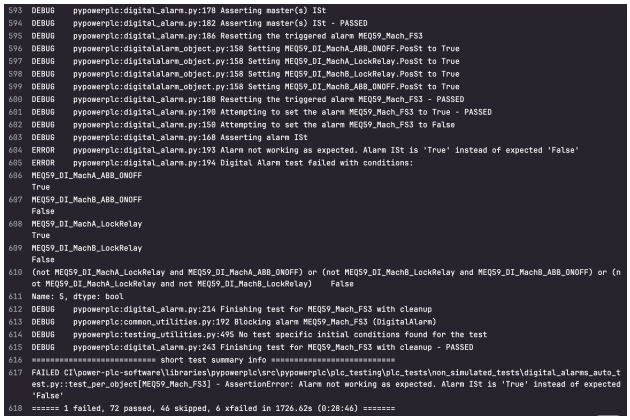


Figure 6: Example of a test report output for a failed test.

Test case optimizer Logic of Digital Alarms or On/Off objects can become highly extensive, particularly when output state of an object depends on a large number of inputs. Exhaustive testing through a standard truth table quickly becomes infeasible, as the number of possible input combinations grows exponentially (2^n) with the number of conditions. For instance, a system with 30 inputs would require over one billion test cases, iterating through them is not feasible. With the proposed approach we were able to obtain the minimum number of test cases. For a logic with 30 conditions, in most cases it is reduced to 31 tests.

A combination of Modified Condition/Decision Coverage (MC/DC) and Binary Decision Diagrams (BDDs) was incorporated [20]. BDDs represent Boolean functions as binary trees. Based on paths in BDDs, the Boolean domain can be extended to $B' = \{0, 1, ?\}$ where ? is not influencing the output. MC/DC is a methodology for identifying independence condition pairs in a Boolean function based on its truth table. After applying this methodology, pairs are selected such that each pair ensures an independent influence of a tested condition on output (see [21]).

By applying the MC/DC methodology to a set of paths from a BDD within the B' domain, we can identify at least one independence pair for each condition in a Boolean expression. Pairs derived from paths demonstrate the independent influence of a selected condition.

Standard Truth Table

A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Paths from BDD

A	B	C	D
1	1	?	1
1	0	1	1
1	0	0	0
0	?	1	1
0	?	0	0

Final Test Table

A	B	C	D
1	1	0	1
1	0	1	1
0	1	0	0
1	0	0	0

Table 1: Example of truth table optimization.

As a result, an optimized set of only $n + 1$ test cases is obtained. For example for small logic $(A \wedge B) \vee C = D$

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

is halved as shown in [Table 1]. The proposed algorithm enables the automatic generation of a minimal test suite for Boolean logic of any size, including logic that was previously untestable, thereby improving both coverage and execution efficiency. Integrated into a Python-based testing framework, it processes logical expressions, generates test tables, and defines initial conditions to ensure comprehensive verification. This approach was essential for achieving full test coverage of the Digital Alarm and OnOff objects in our projects.

CULTURAL ADOPTION AND ADVANCED TECHNIQUES

Adopting a DevOps approach is as much a human and organizational challenge as it is a technical one. Our strategy focused on creating cross-functional teams and fostering a culture of open communication, shared ownership, and quality improvement. We have supported the CI/CD methodology with CERN supported collaboration tools such as *GitLab* , *Jira* [22], *Confluence* [23], *CERNBox* [24], and *Mattermost* [25].

This required developing a hybrid skill set. Automation engineers were trained in the principles of version control, CI/CD, and automated testing. IT engineers who supported these initiatives were educated on the unique constraints of the PLC environment.

We treat our CI/CD framework as a living, internal product that is subject to continuous improvement. We regularly review and refine our tools, incorporate new ideas, automate more manual steps, and adapt best practices as they emerge.

This mindset has allowed our team to not only adopt new workflows but also to develop the next generation of tools that push the boundaries of PLC engineering. The following are some of the key technologies we are now integrating into our workflows:

- Object-oriented development: *Simatic AX* [26] is a modern, *VS Code* integrated development environment from Siemens that integrates natively *Git* version control, and Structure Text (ST) programming language. It enables object-oriented programming, and supports offline unit testing that runs without needing a virtual or physical PLC. We have implemented a proof-of-concept UNICOS-CPC library for a power converter object, completed with an extensive set of unit tests.
- Software-In-The-Loop with Simulink: By using Siemens libraries that connect *S7-PLCSIM Advanced* with online Simulink simulators, we can reuse the detailed plant models developed for HIL platforms much earlier in the development cycle. This allows for more accurate models than our Python-based simulators can provide for certain complex systems.
- Guided commissioning tool: *Smartco* [27] is a web based tool designed to bridge the gap between automated testing and on-site validation. It provides step-by-step guided procedures to verify the desired system states automatically, and generates a report summarizing the test session results.
- Specification manager: This project is the future of the specification-driven testing concept detailed in the section Specification-driven functional testing. We are developing a web based user interface with a database backend to replace our current Excel specification files. This will transform our specifications from static documents into a formal, version-controlled single source of truth.

CONCLUSION

The adoption of IT-inspired practices like version control and CI/CD presents unique challenges in industrial automation. Unlike traditional software, this field often relies on proprietary, graphical programming tools, binary project files, direct dependency on physical hardware, and real-time constraints. These factors have isolated industrial automation from the advancements that have revolutionized mainstream software development.

In this paper, we show how these barriers can be overcome by applying a modern IT workflows and technologies into the lifecycle of critical PLC applications at CERN. By introducing *Git* version control, we enabled a collaborative, transparent and traceable development workflow. Building on that foundation, we developed a CI/CD pipeline with *GitLab CI*, with custom tooling to automate builds and deployments of UNICOS-CPC projects, and with Openstack hosted *S7-PLCSIM Advanced* virtual PLCs. This has been a game-changer, giving us rapid automation, error reporting, and breaking the dependency on physical hardware.

Our automated testing framework *pypowerplc*, built on virtualization and simulation, has significantly improved software quality and reduced commissioning risks. With innovations like the specification-driven testing methodology and the test case optimizer, we have increased our testing coverage and efficiency, making it possible to validate logic that was once impractical to test. By combining Software-in-the-Loop and Hardware-in-the-Loop, we ensure thorough validation of the control logic from the earliest design stages all the way to deployment.

Ultimately, this work offers a roadmap for modernizing PLC software engineering. By embracing a DevOps culture and adapting IT-inspired tools to the specific constraints of industrial automation, it is possible to enhance productivity, improve software quality, and deliver more robust and reliable control systems for large scale facilities.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-TUBG002

ACKNOWLEDGEMENTS

The authors would like to thank the former members of the section who contributed significantly to the conception and implementation of our framework (Maria del Mar Gomez-Jordana, Silvie Nemcova, Liam Cantu and David Carvalho). We would also like to acknowledge the excellent collaboration with the UNICOS-CPC team and express our gratitude for the tools and support they provided.

REFERENCES

[1] K. Kahle, F. R. Blanquez, and C.-M. Genton, "The design and performance of Static Var Compensators for particle accelerators," in *Proc. 2015 17th European Conference on Power Electronics and Applications (EPE'15 ECCE-Europe)*, Geneva, Switzerland, Sep. 2015, pp. 1–10. doi:10.1109/epe.2015.7311721

[2] E. Blanco, F. B. Bernard, P. Gayet, and H. Milcent, "UNICOS: An Open Framework", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THD003, pp. 910–912.

[3] Siemens WinCC OA documentation, https://www.winccoa.com/documentation/.

[4] S. Rocca, F. Boattini, and L.M. de Paco Soto, "Development of a control strategy for the 18 MW power converter of the CERN PS Booster accelerator", in *Proc. COMPEL'18*, Padua, Italy, 25-28 Jun. 2018, pp. 1-8. doi:10.1109/COMPEL.2018.8460164

[5] Q. King, S. T. Page, H. Thiesen, and M. Veenstra, "Function Generation and Regulation Libraries and their Application to the Control of the New Main Power Converter (POPS) at the CERN CPS", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper WEPMN008, pp. 886–889.

[6] B. Denis, P. Malacarne, C. Mugnier, and J. Varas, "The New Control and Interlock System for the SPS Main Power Converters", in *Proc. ICALEPCS'99*, Trieste, Italy, Oct. 1999, paper MC1P28.

[7] S. Chacon and B. Straub, "Pro Git", Apress, https://git-scm.com/.

[8] Gitlab platform, https://about.gitlab.com/platform/.

[9] Visual Studio Code, https://code.visualstudio.com

[10] Octoplant – Version control and backup solution for industrial automation, https://amdt.com/en/octoplant

[11] What is CI/CD?, https://about.gitlab.com/topics/ci-cd/.

[12] TIA Portal Openness: Automation of engineering workflows, Siemens support ID-109792902, https://support.industry.siemens.com/cs/document/109792902/.

[13] J. Castro, "Advanced Features of the CERN OpenStack Cloud", in *Proc. CHEP'18*, Sofia, Bulgaria, Jul. 2018, pp. 07026. doi:10.1051/epjconf/201921407026

[14] B. Schofield, E. Blanco Vinuela, and J.H.P.D.C. Borrego, "Continuous Integration for PLC-based Control System Development", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 478–483. doi:10.18429/JACoW-ICALEPCS2021-TUPV035

[15] Apache Maven, https://maven.apache.org/.

[16] S7-PLCSIM Advanced, https://www.siemens.com/global/en/products/automation/systems/industrial/plc/s7-plcsim-advanced.html

[17] M. Marin, J. M. de Paco, O. Minaiev, and W. Niinimaeki, "A new PLC based control system to orchestrate the PS main power converters", presented at ICALEPCS'25, Chicago, IL, USA, Sept. 2025, paper TUPD012, this conference.

[18] M. Cejp, D. Zielinski, and R. Murillo-Garcia, "Software architecture of FGC4, CERN's next-generation power converter control platform", in *Proc. IPAC'23*, Venice, Italy, May 2023, pp. 3754–3757. doi:10.18429/JACoW-IPAC2023-WEPM080

[19] Pytest documentation, https://docs.pytest.org/.

[20] F. Ahishakiye, J. I. Requeno Jarabo, L. M. Kristensen, and V. Stolz. "MC/DC Test Cases Generation Based on BDDs", in *Proc. SETTA 2021*, Beijing, China, Nov. 2021, pp.178-197. doi:10.1007/978-3-030-91265-9_10

[21] K. J. Hayhurst and D. S. Veerhusen, "A practical approach to modified condition/decision coverage," in *Proc. DASC-01*, Daytona Beach, FL, USA, vol. 1. IEEE, pp. 1B2/1-1B2/10. doi:10.1109/dasc.2001.963305

[22] Atlassian Jira, https://www.atlassian.com/software/jira

[23] Atlassian Confluence, https://www.atlassian.com/software/confluence

[24] H. Gonzalez, G. Del Monte, J. Ferer, and O. Keeble "CERNBox: Storage gateway for CERN and beyond", in *Proc. CHEP'23*, Norfolk, Virginia, USA, May 2023, pp. 01012. doi:10.1051/epjconf/202429501012

[25] Mattermost, https://docs.mattermost.com/.

[26] Simatic AX, https://www.siemens.com/global/en/products/automation/industry-software/automation-software/simatic-ax.html

[27] W. Niinimäki, Z. A. Morrissey-Ralevic, M. Marin, and H. B. Dennis, "Smart Guided Commissioning of Industrial Control Systems", presented at ICALEPCS'25, Chicago, USA, Sep. 2025, paper THPD100, this conference.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2025). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.