# Software factory for industrial automation: focus on the PLC software quality

Denis Chalon
CTO

Wafaa Ait-Cheik-Bihi*, PhD
R&D Engineer

Itris Automation
2 square Roger Genin
38000 Grenoble – France
Tel: +33 (0)4 76 23 67 65 - Fax: +33 (0)4 76 23 43 21
E-mail: Wafaa.ait-cheik-bihi@itris-automation.com

---

***Abstract*** **–** In recent years**,** considerable effort has been put into the area of Programmable Logic Controller (PLC) in terms of performance, reliability, availability, communication, and integration of business functions. After this focus on hardware improvements, interest is increasing around PLC projects' lifecycle software (e.g. simulation, deployment, and requirement traceability). For example, the latest update of the IEC61131-3 standard includes the Object Oriented Programming (OOP) with the possibility to use interfaces and inheritance in order to allow software components to be more reusable.

Industry 4.0 is aiming to develop future and smart factory by merging automation and digitalization, resulting in more efficient production methods. A major challenge of the Industry 4.0 is to industrialize the production of software. The main purpose of this industrialization is to introduce methods and tools to make software controllable and measurable and to reduce production cost. This can be done for example, by integrating developer best practices (e.g. Good Automated Manufacturing Practices in Pharmaceutical industry) or certification requirements (e.g. IEC 61 508). It enables non-developers to better understand and evaluate the quality (e.g. testability, performance and robustness) of PLC programs. Benefits of a more formal PLC development process include, for instance, increasing the CMMI maturity level, specifying the required quality level depending on the certification area (e.g. PESSRAL for controlling lift, EN 50128 for railway applications), and continuously comparing the intrinsic quality.

In this paper, we present the architecture of a software factory for PLC programs that allows automatic synchronization between documentation and code, non-regression tests reporting, and software requirements traceability. We will also show how this can be used by PLC programmers and stakeholders to connect quality, productivity and efficiency during the whole PLC projects' lifecycle.

***Keywords*** **–** *Software factory, industry 4.0, PLC programming, software quality*

---

* Corresponding author

# 1. Introduction

PLC applications become increasingly complex and critical. PLC programs producers, end users, and stakeholders are majorly concerned by their quality. This quality relies heavily on the quality of the whole development process of PLC programs, including, requirements, design, development, tests, deployment and maintenance.

Great researches, process-oriented methodologies and techniques are widely used in computer science for the evaluation of software development process quality and maturity. For example the Capability Maturity Model (CMM) aims to support the assessment of the process, its improvement, and, consequently the development and deployment of software products [FLM+98] and [fda02]. Other examples are ISDS standard from DNV [dns12] with similar goal but focused on the Off-Shore Oil and Gas domain and GAMP 5 [gam08] in the pharmaceutical domain.

Because a lot of tools are available, these techniques are proven in computer science domain. There is no reason why PLC developments shouldn't benefit from them. For this reason, there is a need to build a software factory that integrates a set of tools covering the whole PLC development process. A software factory is a structured collection of related software assets that aids in producing computer software applications or software components from the input elements.

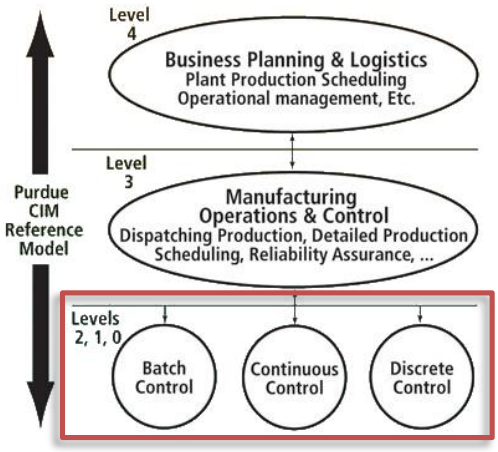


Figure 1- Purdue Reference Model

The software factory improves productivity by automating many repetitive tasks during the project lifecycle and in parallel continuously verifying conformance to customer expectations.

Our domain of interest is the industry where system architecture is using PLC and SCADA systems. Figure 1 illustrates the Purdue reference model from the ISA 95 (International Standard ANSI). The software factory described in this paper applies to the process control level 1 of this model.

PLC application development starts with high level customer functional specifications, test plan, and coding guidelines as shown in Figure 2. Depending on the development context, certification standards may be also considered as inputs of the process. At the end of the development, deliverables are one or more PLC applications with fulfilled test plan (i.e. functional and formal) and up-to-date documentation. Then the complete system certification may start.

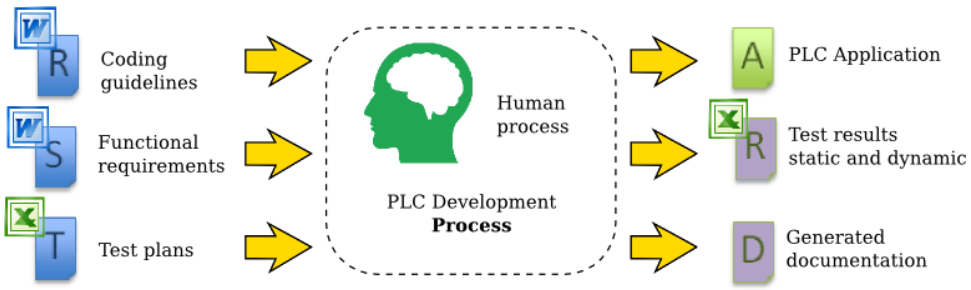


Figure 2: PLC development process inputs/outputs

The main purpose of this contribution is to present an integrated set of software to support a repeatable and traceable approach to PLC development. This effort increases PLC software quality and control PLC's software production costs based on common engineering techniques.

In the remainder of this paper, background and related work is given in section 2. Section 3 describes the software factory. In section 4, architecture of a PLC software factory is detailed. Section 5 contains the conclusion.

## 2. Background and Related work

Building PLC applications is a labour-intensive process that relies on a limited pool of talented PLC developers, especially if those applications are more complex and critical (i.e. with Safety Integrity Level 3 or 4). In one hand, the demand for software exceeds the capacity of this labour pool, thus current PLC development methods should be partially replaced by automatic techniques, meaning cheaper, faster, and more reliable application development. In the other hand, most traditional software environments highlight support for producing code and associated documents. Therefore, the focus, in software factories, shifts to coordinating information between producers and consumers so that the right person always has the right information at the right time [GS03] by integrating modern technologies.

Several researches have been conducted to define the main concepts of software factories in computer science domain. The authors of [B+75] describe the need to develop an integrated software factory for software development process. They gave some proofs on how these integrated software tools can increase software reliability and control software production costs. [GRKH09] gives specifications of well-built software factory by using new methodologies in engineering systems such as meta-models and Model Driven Development (MDD [Sel03]) in order to facilitate the software development. The authors of [ACD07] articulate the software factory for managing and orchestrating reusable assets for setting up distinct software factory. This is done based on several computer engineering principles such as architecture-driven software, and meta-model by tailoring them to form directly executable software assets. [WK06] describes some experiences in building a software factory by defining multiple small domain specific languages (DSL) and having multiple small models per DSL. The models behave like source code to a large extend, leading to an easy way to manage the models of large systems.

In the process control systems field, the author of [Dut07] provides software factory techniques applied at CERN. A software factory called the UAB (UNICOS Application Builder) is designed to enable faster and cheaper code generation in a context of often changing requirements.

In the PLC domain, there are few tools and advanced technologies used to support PLC users in their development process from specification to commissioning phases. Let's consider the test automation process in this domain. Unlike computer engineering for which we talk now about Behaviour Driven Development (BDD) and Test Driven Development (TDD) [CB10], these techniques are almost unknown of PLC developers. The purpose of BDD and TDD is to start developing applications by writing dynamic specifications and tests first and then starting developing the functional code. This helps to generate tests automatically that are run at any stage of the development. In our days, there is no tool that integrates these concepts for PLCs. The authors of [BK13] explain why test automation is needed in industrial automation. There are some existing tools for test automation such as Siemens PLM (Product Lifecycle Management) Software© which allows building products by optimizing their processes from the planning and development step till the deployment and maintenance. Another tool

is Dassault System Delmia[1]© which allows manufacturers to virtually experience their entire factory production from the impact of design to determining how to meet global demand. There are also some PLC IDEs (Integrated Development Environment) that are proprietary to PLC vendors. These tools are used for PLCs of the same vendor hence they cannot be used for other PLCs. In some cases, PLC developers validate the PLC software manually or by simulation using some existing tools (e.g. PLC Simulator[2], ControlBuild®, EasyPLC Software Suite[3], Brad® Simulation [bra]). This manual process sometimes is done directly in the actual factory (i.e. Factory Acceptance Test).

We come up with the list of issues below that should be considered in the future software factories in the PLC area to improve productivity, quality, and visibility of the development process:

- Lack of development visibility
- Changing requirements during development
- Consistent documents
- Lack of design & verification tools
- Lack of software reusability
- Multiple PLC languages and platforms

Software factories have eventually come to the stage as an umbrella solution to software development productivity by assembling the applications with existing/new frameworks, patterns, models and tools. The main contribution of this paper is to propose an architecture of software factory in order to increase PLC software quality and master software production costs. Such factories will help to meet requirements of a lot of standards such as IEC61508, IEC61511, PESSRAL, IEC61848, GAMP5, and CMMI.

## 3. PLC software factory

As discussed in the previous section, there is a need to build PLC software factory based mainly on existing techniques and methods from the computer science field. In this section, we give a description of the scope of such a factory. We also define the requirements and benefits of such systems.

Figure 3 illustrates the main inputs and outputs that can be considered by the software factory. Compared to the common PLC development process (refer to Figure 2), the test plan is not anymore a spreadsheet file but it uses a format that can be processed automatically. This is also the case for coding guidelines that should be formalized for the static analyzer.

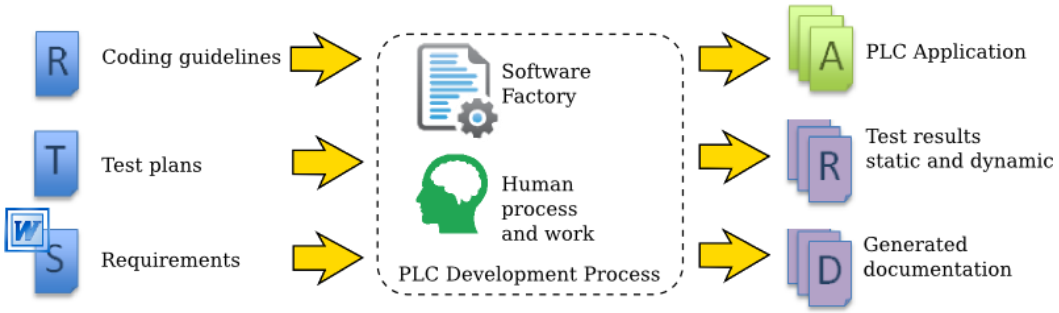


Figure 3: PLC Software Factory inputs/outputs

[1] http://www.3ds.com/products-services/delmia/
[2] http://sourceforge.net/projects/plcsimulator/
[3] http://www.nirtec.com/

### 3.1. Needs and requirements for the PLC domain

As mentioned in the previous figure, human work is still required in the process. The pragmatic approach we have is that focusing on 20% of the problem will save 80% of the effort. The figure below presents the target in terms of automation of software development tasks.

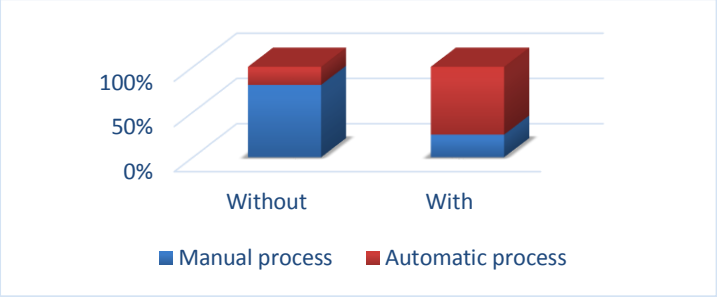


Figure 4: Manual versus automatic processes without and with Software Factory

The software factory we are proposing is mainly based on commercially available software products. The main requirements of such a system in terms of usability are the ability to allow integration in existing development environment, an easy deployment, and an intuitive user interface.

PLC developers should continue to develop using their usual Integrated Development Environment so as to minimize the learning curve and to maximize user acceptance. The only requirement for them is to use a version control system.

### 3.2. Benefits of Software Factory

From craft to industry, the major added value is quality. Using a software factory is a good way to define quality objectives and then to measure the software quality during the whole application development cycle and check that the objectives are met.

Hereafter are some advantages of using software factories in the PLC domain.

***Seamless integration through modern technology*** offers a transparency to PLC developers. Major tasks are handled automatically and the errors and bugs are reported automatically. A link is seamlessly established between code version, tests results and requirements. PLC developers keep their development environment with the constraint to commit their changes into a version control system. All generated information is accessible with Web technologies so it is easy to share with non-technical stakeholders. Deployment and maintenance are easy to manage.

***Faster, cheaper, and more effective***: the main goal of software factories is to shorten the development cycle. It also allows avoiding human errors.

The productivity and cost reduction is the immediate benefit of such a software factory. The Figure 5 shows the remediation cost of one bug depending on when the bug is found. In the example given further, the discovered bug may cost 10 to 100 times more if not solved during the development. However, the software factory

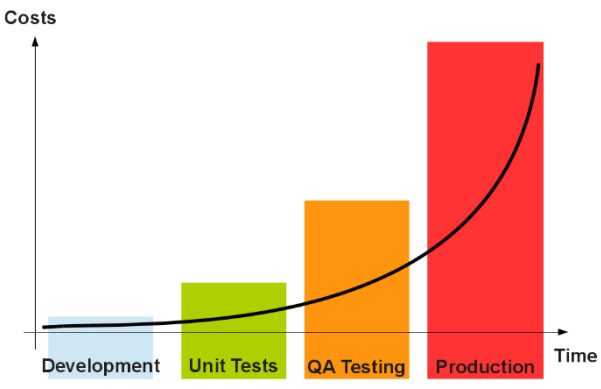


Figure 5: Cost versus time

helps to find bug earlier.

***Adaptation to regulated contexts:*** for all critical developments, where standards are enforced, the software factory helps to formalize development process, to communicate about it, to enforce its usage. Even with a partial software factory, it is more expensive for the developer to cheat the system than to follow it:

- In the pharmaceutical domain, GAMP5 from ISPE requires to detail the Quality Management System. In critical system development, IEC61508 requires a Software Quality Plan. Part of those documents can describe the software factory processes.
- For third party audit, having a centralized repository with all versions of all project documents, gives confidence that the process is defined, deployed and correctly used.
- The required code review processes are easier to manage because the automatic generated content helps to get an objective and a broad view of the software status. Thus, such a software factory is useful during the code review meeting and for summarizing notes for archiving.

***Improved (and inexpensive) traceability:***

- All files are identified by a version number. All versions of all project files are kept under version control.
- Each PLC program version is tested against the current version of the functional requirements. There is no risk to run wrong test version on wrong program version.
- Functional requirements and PLC application are decoupled: no need to synchronize them explicitly, but a link between requirements and application exists when looking at the test results (example on page 10).
- Automatic regression detection is possible. Whenever a test case which used to be passed becomes failed, an alert is sent to the developer who had performed the commit (example on page 10)

***Clear visibility of the product state and quality*** provides a better understanding of key metrics and measures for all PLC application's stakeholders. The status of software is shared among everyone at a given stage during its life cycle. This allows projects being more collaborative. Examples of indicators and information are:

- Up to date documentation in format suitable for non-developers : data flow and control flow documentation, listing in graphical language (SFC, Ladder, FBD)
- Code with its associated score: percentage of passed tests, complexity metrics, etc.
- Project completion indicators, test coverage, etc.
- Trends over the different measures performed around the PLC application.

The tools of the software factory bring their own benefits:

- Model checker can prove that the PLC application respects some safety properties
- Advanced documentation generator helps designers performing impact analysis when specifying PLC application evolution.
- A connexion to real PLC may allow animating generated documentation with real values from the field.

In the next section, an example of software architecture is presented with the main components and the required functions/features in PLC development process.

## 4. Software Factory architecture

In this section, we present the architecture of one possible software factory designed for PLC applications. As shown in Figure 6, the software factory contains a version control system and a continuous integration server. All software stakeholders use a dashboard available as a Web interface which gives access to all persistent information. In between, tools are generating content: tests and static analysis results and documentation. A communication to a real PLC is also an option either for automated testing or diagnostic support on the real hardware.

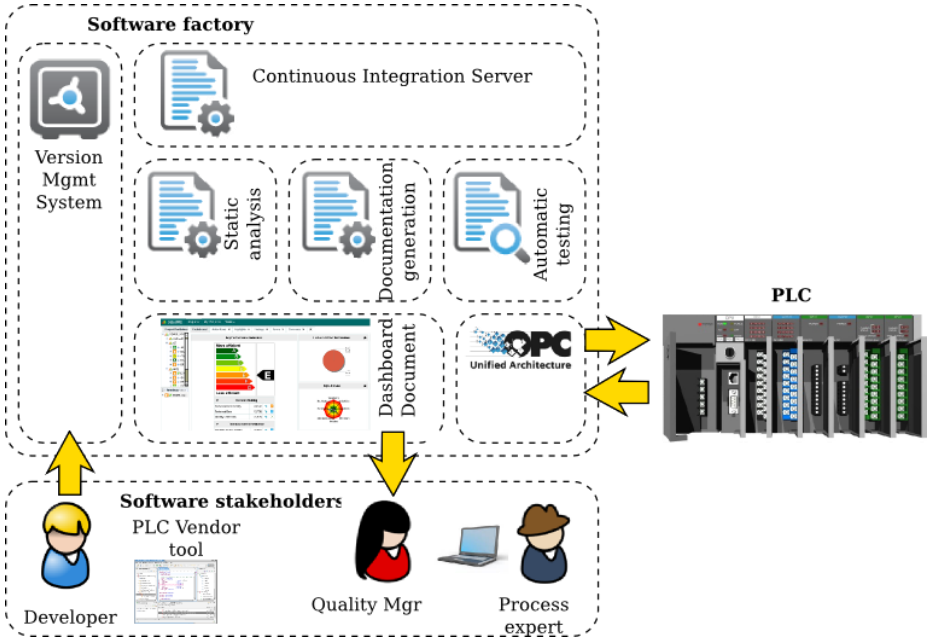


Figure 6 - Software factory architecture

But before detailing those different elements, a software factory requires computable files as inputs. The next section details the process of formalizing these inputs.

### 4.1 Inputs formalization

Currently, most processes are relying on humans. To exchange information between stakeholders, office automation software is used: spreadsheet editor, document editor, PDF format, etc. All those documents are written in natural language and from a semantical point of view, they are difficult to understand by tools.

The input documents that should be understood by the software factory are: test plan, functional requirements and coding guidelines.

This formalization of input documents is also useful to comply with standard requirements such as the *Supplier Good Practice #2 from GAMP5: Establish requirements*

#### *4.1.1 Functional requirement and test plan*

Functional requirements and test plans are two different ways to describe the behaviour of the system. So we decide to rely on a language called Gherkins used in some computer science developments. It is a DSL (Domain Specific Language) that lets you describe software's behaviour without detailing how

that behaviour is implemented. This language can be used by domain experts without computer science or programming knowledge.

The functional requirement is composed of scenarios, in which there are test cases. Each test case verifies that the system is in a given state, and then when some stimuli are received, the system evolves to a well-known new state. If during test execution, it is not the case, the test is failed.

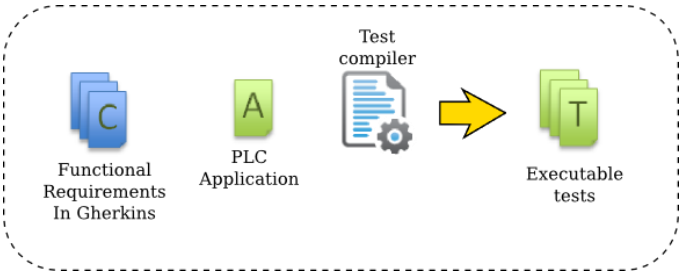


Figure 7 - Generation of executable test from application and functional requirements

Later in the development process, the functional requirements and the given application will be analysed to generate executable tests as shown in Figure 7. This tool is in charge of measuring compliance of the application with requirements.

### *4.1.2 Coding guidelines requirements*

On big PLC applications or critical applications (smaller but subject to a regulated context), the usage of coding guidelines may be a requirement. To be able to measure the software quality along the development, it is important to formalize the coding rules as manual verification can be really expensive.

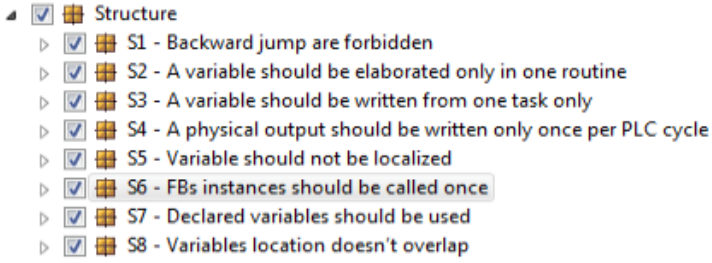


Figure 8 - Some coding guidelines

This formalization really depends on the capabilities of the tool that will be used. Verification of the compliance of an application with a set of rules can mostly be done using static analysis techniques without having the PLC program executed.

Although no industry wide set of guidelines currently exists, the PLC Open[4] association is currently elaborating such a document that could be applied to the whole industry.

## 4.2 Continuous integration

Once the tools are able to understand the customer requirements, and check a PLC program compliance to them, it is important to define a way to identify a given software version. The version control system is a key element of the software factory. All input documents should also be part of the version control system, as they can change during the application development.

The version becomes the identifier of a PLC program and its corresponding input documents. This identifier will be used through the whole software factory to tag the test results, quality results, and the automatic generated documentation. Using this tag, traceability is much easier to manage.

[4] http://www.plcopen.org

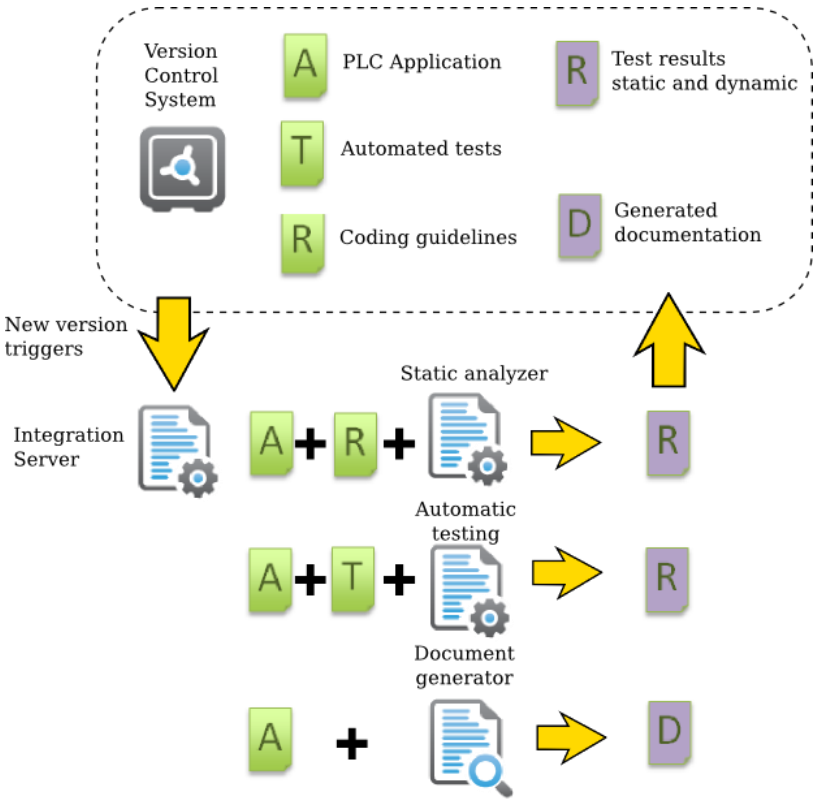


Figure 9 - Automatic generation of test results and documentation

The heart of the software factory is then the continuous integration server. It is automatically launched by the creation of a new software version. It triggers the complete automatic testing.

Continuous integration server connects all testing tools together. In our example, the following tools are used:

- Static analysis tool ensures compliance with coding guidelines
- Automatic testing tool ensures compliance with functional requirements
- Automatic documentation generator ensures consistency of the documentation

Then the development process is iterative. The developer writes code to implement functional requirements, and then he commits code into the repository of version control system. The commit triggers the execution of all tests (static analysis and automatic testing) and the generation of documentation. When all tests are executed, reports are generated. Those reports are used by the developer to organize the next development iteration. They are also available for other software stakeholders.

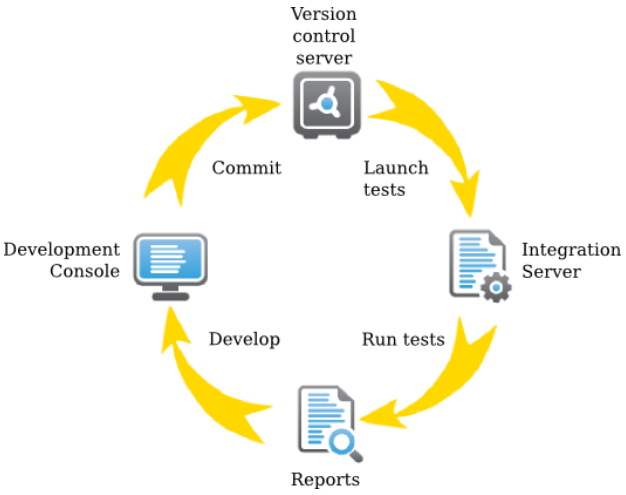


Figure 10 - PLC development process

### 4.3 Dashboard and reporting

Understanding the software means often being a developer. But all software stakeholders are not (and should not be) developers. The software factory should present all information related to the software so that even non developers can find and understand easily the required information (e.g. test results, regression, quality level, milestone visibility). This software factory uses a quality dashboard deployed mainly in computer science context that is flexible enough to adapt to PLC domain constraints and to present clear views of quality measures.

Figure 11 illustrates an example of quality dashboard, SQuORE© from Squoring Corporation. For each result version, including static analysis and automatic tests, there are different views from broader view giving a percentage of passed tests to a very detailed view allowing looking to actual findings.

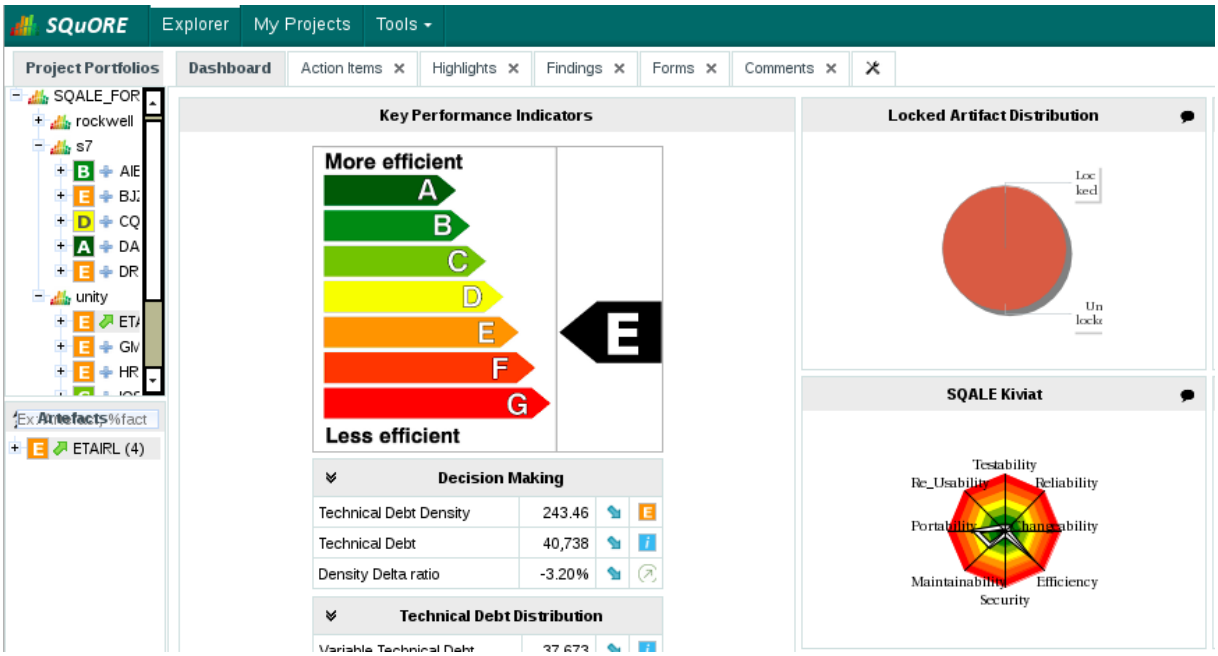


Figure 11 - Quality dashboard

Also, as software factories are based on web technologies, they can easily be integrated on both office computers and development workstations.

### 4.4 Case study

To illustrate the actual applicability of the proposed software factory, we show an example of a Safety-related application for lifts. This development is subject to the PESSRAL standard (ISO/FDIS 22201 - PESSRAL).

Here is an example of a requirement extracted from PESSRAL document ([pes09]):

> **"Check final limit (automatic or inspection)"** implies the **"removal of power from machine motor and brake"**

This requirement should be modelled into Gherkins language so that it can be used by software factory. The corresponding Gherkins requirement is then:

**Scenario**: lift cannot go above top final limit

**Given** the lift is at the top final limit

**When** inspector wants to go up

**Then** the motor and the brake are not powered

The PLC developer will write some code to implement this requirement. From a hardware point of view, a SIL3 coder provides an input to the PLC, the brake is driven by a redundant digital output and its associated relay, and the motor is controlled by a drive.

In the PLC application database, the developer creates some variables and tags them with information for traceability:

```
IW_LiftPosition : DINT AT %IW2.4;
Attribute : “the lift is at the top final limit” IW_LiftPosition >= 35452
            “the lift is in normal zone” IW_LiftPosition < 35452
OX_MotorPower : BOOL AT %QX1.1;
Attribute : “the motor is powered” OX_MotorPower = TRUE
            “the motor is not powered” OX_MotorPower = FALSE
```

Let’s say, this code is committed in version #7. This commit triggers the execution of the tests thanks to the continuous integration server.

The executable test cases are generated from the application and the functional requirements as mentioned in Figure 7 page 8. There is no need to write the test case code because a link is already established between the requirements and the application.

Let’s say that the result of this test happens to be a PASS.

Later in our story, the code is being changed in commit #12, the tests are again automatically triggered and the result is a FAIL. It means there is a regression. Thanks to our software factory, this regression is detected immediately. The PLC developer is informed, fixes the error, creates a version #13 and tests are being run again.

That story can also be represented graphically as below.

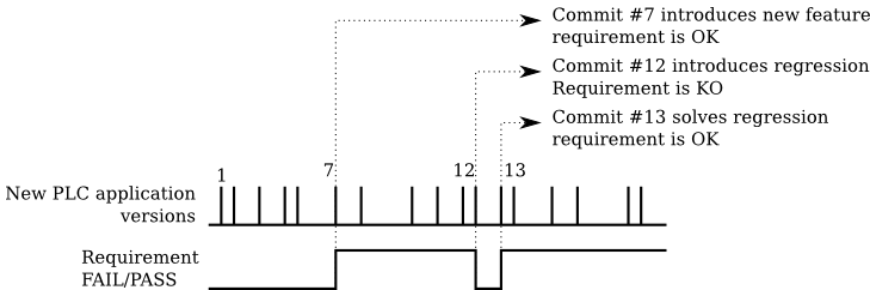


Figure 12 - Traceability between requirements and application

Without the software factory, the regression wouldn't be detected immediately. The cost of remediation would be much higher and the developer wouldn't be as efficient as when modifying freshly the code.

Moreover, when using a software factory, the PLC developer is focused on its main task: software development. Other tasks are handled by the tools.

## 5. Conclusion

In this paper a software factory is presented for PLC development process. Its advantages and benefits are mainly enhancing and improving the quality of PLC application and improving the productivity of the developer.

Software factory is not a magic wand. It won't replace the PLC developer. It won't solve the required certification process. But a development team using a software factory is more mature (CMMI). The PLC applications have a defined level of quality that matches the quality requirements. The PLC developer is freed from repetitive manual tasks. He can focus on code development, and on special features like communication, regulation, motion. The certification body gets a good confidence in the compliance with the requested process just by looking at the software factory and the associated dashboards.

Finally it should be noted that software factory will be all the more important when taking into account cybersecurity requirements. Cybersecurity is a question of code robustness. Ensuring code robustness means doing a lot of tests, continuously. Without a software factory this is just economically not possible.

We also believe that software factory should be enriched according to industry requirements. This can include code generation and extension to take into account SCADA. These two possibilities will be further developed in our ongoing work.

## References

[ACD07] N.Ilker Altintas, Semih Cetin, and AliH. Dogru. Industrializing software development: The "factory automation" way. In Dirk Draheim and Gerald Weber, editors, *Trends in Enterprise Application Architecture*, volume 4473 of *Lecture Notes in Computer Science*, pages 54–68. Springer Berlin Heidelberg, 2007.

[B+75] Harvey Bratman et al. The software factory. *Computer*, 8(5):28–37, 1975.

[BK13] Shiraz Gilani Barath Kumar. Test automation in industrial automation. Paris, October 2013.

[bra] Brad simulation tools.

[CB10] S. Bouchez-Mongardé C. Baillon. Executable requirements in a safety-critical context with ada. In *Ada User Journal*, volume 31, pages 131–135. June 2010.

[dns12] Integrated software dependent systems (isds). Technical Report DNV-OS-D203, DNV & DET NORSKE VERITAS AS, December 2012.

[Dut07]Mathias D Dutour. Software factory techniques applied to process control at cern. Technical Report CERN-IT-Note-2007-035, CERN, Geneva, September 2007.

[fda02] General principles of software validation; final guidance for industry and fda staff. Technical report, U.S. Department Of Health and Human Services, Food and Drug Administration, Center for Devices and Radiological Health, Center for Biologics Evaluation and Research, 2002.

[FLM+98] Alfonso Fuggetta, Luigi Lavazza, Sandro Morasca, Stefano Cinti, Giandomenico Oldano, and Elena Orazi. Applying gqm in an industrial software factory. *ACM Trans. Softw. Eng. Methodol.*, 7(4):411–448, October 1998.

[gam08] Gamp5 a risk-based approach to compliant gxp computerized systems, 2008.

[GRKH09] J. Greenfield, M. Regio, W. Kozaczynski, and T.J. Hollander. Software factory specification and execution model, April 16 2009. US Patent App. 11/974,723.

[GS03] Jack Greenfield and Keith Short. Software factories: Assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, pages 16–27, New York, NY, USA, 2003. ACM.

[pes09] Design and development of programmable electronic systems in safety-related applications for lifts (pessral), 2009.

[Sel03] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003.

[WK06] J. B. Warmer and A. G. Kleppe. Building a flexible software factory using partial domain specific models. In *Sixth OOPSLA Workshop on Domain-Specific Modeling (DSM'06), Portland, Oregon, USA*, pages 15–22, Jyvaskyla, October 2006. University of Jyvaskyla.