ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR02

# BENEFITS OF LOW CODE DEVELOPMENT ENVIRONMENTS ON LARGE SCALE CONTROL SYSTEMS

B. Lefort, V. Costa, CERN, Geneva, Switzerland

*Abstract*

The rapid evolution of science and of scientific projects usually implies high levels of mobility among researchers, engineers and applied scientists. In parallel, software development has been getting easier as computing technology has evolved. A direct consequence of these two paradigms is the proliferation of custom, sometimes small, ad-hoc software. This software, usually quickly developed and with low code hygiene, later becomes a burden to the organization, especially if the original developer and responsible departs. Based on this experience many organizations are now successfully adopting low-code application development. Inspector is a low-code development platform to design control interfaces. It features a visual interface composer, a visual programming language and supports small integrated scripts in Python. More than 600 Inspector applications are actively used at CERN. We explain how developers with little experience of writing software can create applications that they could not otherwise explicitly code for themselves. Finally, we demonstrate how Inspector offers enhanced security, higher productivity and maintenance relief by delegating the core software development and maintainability to high skill developers and IT members.

## INTRODUCTION

The number of PhDs and postdocs in science has grown substantially. A report in Nature reported a jump by 150% in the number of postdocs between 2000 and 2012 (see Fig. 1) and the growth shows no sign of slowing since then. [1]

PhD graduates and Postdocs confront a dwindling number of academic jobs. As an example, only 15% of PhD graduates can attain academic positions in the USA [2]. Many of them go the entrepreneurial route and become involved in start-ups, research labs or commercial R&D centres.

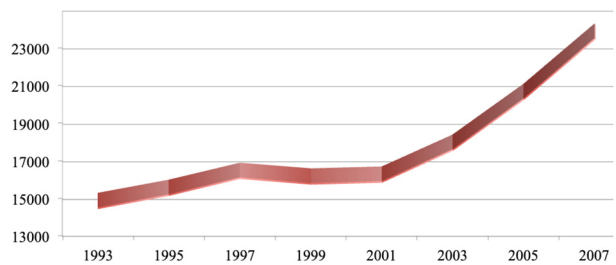


Figure 1: Annual number of science and engineering doctorates graduating from US universities.

The typical maximum funding period for doctoral students is four years. There is no set length for a postdoctoral researcher as it depends on a number of factors such as the university, country of research, PI, or funding. That being said, most positions are two to three years even it they can be extended up to 6 years.

Over the past 15 years, the number of public workers on short-term contracts has increased. In France they now represent 35% of university staff and 27% at research institutions. Contracts tend to be very short. 80% of them are lasting less than 2 years [3].

Consequently, Academia is facing a high employment turnover rate combined with short-term contracts. In regards to software engineering, such conditions make quality and productivity competing objectives. Code quality is naturally seen as less important when compared to fast/cheap development, as such, software projects are showing signs of poor code quality, undetected vulnerabilities and low code readability – contrary to the fundamentals of software maintainability.

Considering the trends, we will show how a low-code application, such as Inspector, a Rapid Application Development (RAD) framework, gives access to application development for people with little experience of writing software and how it reduces the number of Lines Of Code (LOC) that has to be maintained when the person who developed and maintains it departs.

To boost the productivity, Inspector proposes a separation between the UI and the software technology, essentially allowing the creation of zero code GUI in order to diminish the cost of developing and maintaining such applications. Inspector itself is built on proven technologies (such as Java) and takes care of critical software aspects such as multithreading, data communication and application reliability. Final application developers don't need to worry about these aspects; they can create their entire application using the GUI and tiny scripts for complex operations. This concept also introduces indirect benefits, for instance all Inspector applications are uniform. They'll look and behave the same way. This helps create a sense of familiarity and control that helps users going seamlessly from one application to another without the need to learn new ropes to get around.

## LINES OF CODE AND CODE QUALITY

Capers Jones, an American specialist in software engineering methodologies, has compared many methodologies (RUP, XP, Agile, Waterfall, etc.) and programming languages over thousands of projects and has determined that programmers write between 325 and 750 production lines of code (LOC) per month. [4, 5] These numbers apply to software scientists working in teams that follow a strict development cycle where specification is followed by development and precede unit & integration tests.

Short-term contracts workers in a lab environment are typically result driven. User stories such as "develop a

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR02

driver that controls this hardware so I can start doing science with it"; "This data reduction pipeline has to work so I can see my results" are commonly heard. In such cases, the specification phase is replaced by a trial and error loop (develop > test > repeat) and the final code only reflects an interpretation of a problem and a solution for it with too little considerations about the fact that it has to be used and maintained by others.

A clear indication of a code quality braking point is when, trying to squeeze an unforeseen use-case in a clearly incompatible design, one promises to "clean it up later, when I have the time". Beyond this point, the development cycle slows down, complexity goes up and quality is lost. Technical debt is easily generated in R&D environments and really difficult to overcome. A side effect is toil. These applications typically require intensive maintenance and are a burden for development teams. Badly designed software also affects all its dependencies and, indirectly, the everyday life of the corresponding developers and users.

## COMMONLY REINVENTED WHEELS

Most of the scientific applications can be resumed to a simple list of actions: Sample, process and display.

Data sampling is generally limited to a few calls to driver setup functions before starting the acquisition. This part of the process is hardware specific but access can be generalised using standard APIs. In more complex cases handling data subscriptions or polling may be required. Handling high-rate data streaming can be hard for less experienced programmers, leading to less reliable and less secure systems.

For very complex processing one usually relies on a scripting or programming language. When the processing is simple, it can be addressed using a visual programming language like LabView (from National Instruments) or the ones offered by Inspector.

When it comes to GUI, even if new reality-based interactions are being explored [6], we still rely on the "Windows, Icons, Menus, and Pointer (WIMP)" graphical elements to represent information [7].

In academic software productions the final GUI will probably be an afterthought and the basic graphical elements set will have to be extended with scientific plotting capabilities. A nice and functional GUI is usually hard to implement and requires a fair amount of LOC. The actual trend is to delegate this task to a GUI builder that simplifies the creation by allowing the designer to arrange widgets using drag-and-drop (DnD) gestures. Once designed, the builder automatically generates all the source code. These tools are effective but require a non-negligible learning phase.

The development of GUI is time-consuming. It can represent about 48% of the source code, and 45% of the development time and covers 37% of the maintenance time [4]. If we stick to our academic goal, which is producing results, these tasks are repetitive and can be considerate as rebarbative by many programmers.

## LOW-CODE DEVELOPMENT PLATFORMS

Low-code development platforms provide an environment to create applications through graphical user interfaces and configuration instead of the traditional programming. They are based on the principles of model-driven design and visual programming and nearly anyone can learn how to use them. Most of these platforms keep the ability to let the user insert custom code when needed.

Forrester, an American market research company that provides advice on existing and potential impact of technology to its clients and the public, estimates that the total market for low-code development platforms is growing by 50% every year [8] (see Fig. 2).

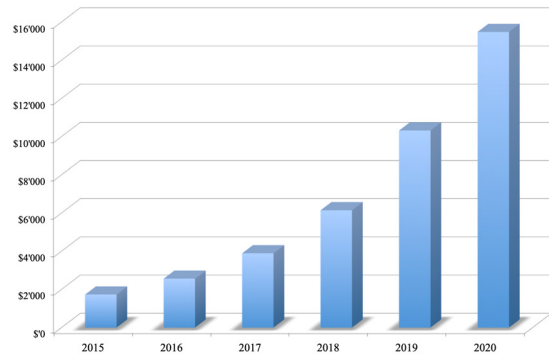


Figure 2: Low-code platforms forecast (US$ billions).

Gartner Inc., a public global research and advisory firm has developed methods to demonstrate market trends, such as direction, maturity and participants. Gartner predicts, due to continued demand for applications and a shortage of skilled developers, that low-code development tools will be used for most application development by 2024 [9].

## CERN CONTROL MIDDLEWARE INFRASTRUCTURE

The CERN accelerator complex control systems are fully data-driven thanks to the Controls Configuration Database (CCDB) [10]. All the accelerators equipment can be accessed as a device. A device is a named entity of the control system, which usually corresponds to a physical equipment. Using an identifier, the logical interface of a device can be retrieved from the CCDB.

Communication is made through the Java API for Parameter Control (JAPC) [11]. JAPC is an abstraction communication layer that unifies CERN control systems thanks to the concept of JAPC parameters. A parameter represents a controlled value of an accelerator. Client programs can access JAPC parameters with set and get functions in an asynchronous way. They can also subscribe to them in order to be notified when values change. JAPC also serves as a wrapper for Java applications hiding the complexity of different protocols.

Access authorization is provided by a Role-Based Access Control (RBAC) that restricts the access to a system only to authorized users [12]. Each user has assigned

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR02

roles that specify what he is allowed to do in the system. Every get, set or subscription demand is subject to the authorization process and its execution can be denied when a user has insufficient privileges.

## INSPECTOR

Inspector is a Low-Code IDE developed at CERN [13]. Inspector allows users to concentrate their efforts on the data presentation, rather than on implementation details. It takes responsibility of making the applications portable, within the production environment, handles data acquisition and streaming, multithreading processing and provides a reliable common experience for users of all skill sets (see Fig. 3).

The Inspector uses the CCDB to automatically expose and control all the properties of a device without the need for any manual configuration. Users can select devices and their properties. These elements can be *drag-and-dropped* into a canvas called *Panel* to start creating a GUI. Inspector internally obtains the property metadata, such as data type and access mode. Based on this information a visual menu appears with all the available graphical elements that can display or interact with the property value. Such Visual elements are called Monitors.

The vast assortment of monitors, including polar plots, multi-state switches, buttons, dials, etc. can be organized via automated layout options. Visual elements can also be organized in a grid system and layout constraints can be applied to them.

## INSPECTOR AS AN INTERFACE FOR OPERATORS

On a large-scale control system, when it comes to tune a process or to investigate a failure, one needs versatile tools that can quickly and easily display and correlate data. Especially during machine upgrade or commissioning where many changes can suddenly be required. In Inspector, every single functionality is available as soon as the property is dropped into the panel. There is no separation between the design phase and the deployment of the application. This feature allows inspector to be used as a black board where properties from multiple devices are shown at once. Panels and monitors are easily created and disposed. In many cases Inspector applications are created temporary given its flexibility, ease to use and low development effort. This low commitment allows operators to use Inspector to perform quick measurements, debugging and create temporary dashboards and control panels, an important feature in the ever-changing CERN accelerator complex.

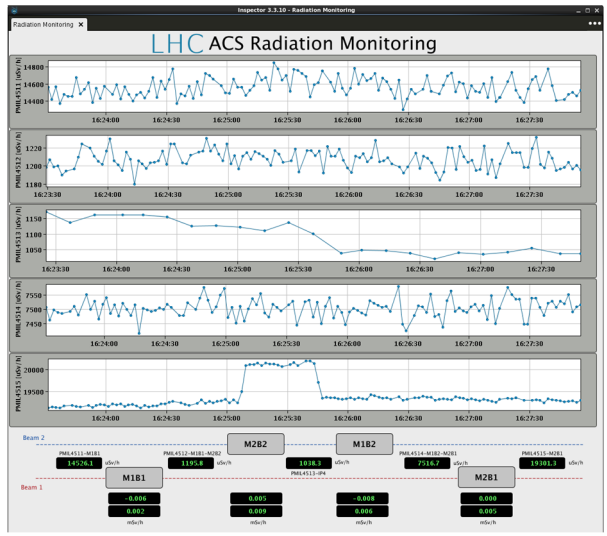


Figure 3: Simple example of applications developed with Inspector. Development time for this kind of application may vary from a few minutes to a few hours.

## INSPECTOR AS AN INTERFACE FOR ENGINEERS AND SCIENTISTS

Inspector is developed to be flexible. It was built following modern development practices, with modularity in mind. It was developed at CERN but is not highly coupled with any CERN system - it can be adapted for any other complex and can be extended. With that in mind, Inspector was also built to be generic - it allows quick and easy application development but also supports complex data processing and closed loop control. Monitors allow for simple data transformation, such as ranges, formatting, blinking, etc. Inspector introduces two distinct scripting modules for data transformation and control.

A visual equation editor is included in the application (see Fig. 4). This equation editor facilitates the creation of mathematical equations that are executed on the server side. Equations may receive inputs from properties and can apply functions to values or other properties. The computation happens based on a timer or an event.

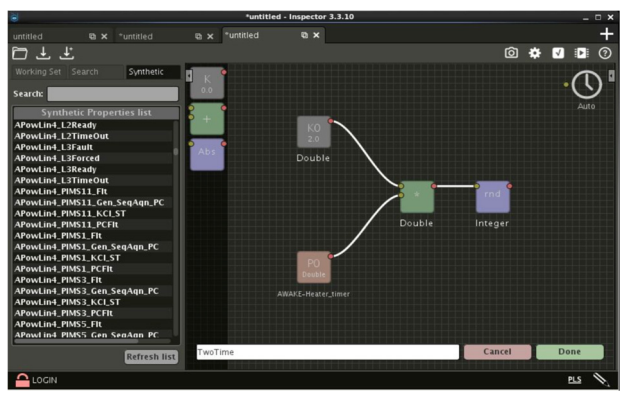


Figure 4: The equation editor allows processing the data subscription in real time (simple arithmetic, averaging, FFT).

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR02

For more complex logic and data processing, Inspector includes a specialised module called Blueprint (see Fig. 5). Blueprints are small programs on their own that combine visual scripting, for users with limited programming knowledge, and high-level integrated language scripting, including Python support. Blueprints execute in the client Inspector application and can interact with visual elements and properties. Visual elements, such as monitors, can be configured in runtime with Blueprints, for instance: visual components can be altered when a state of a device changes or controls can be hidden or displayed based on user access permissions.

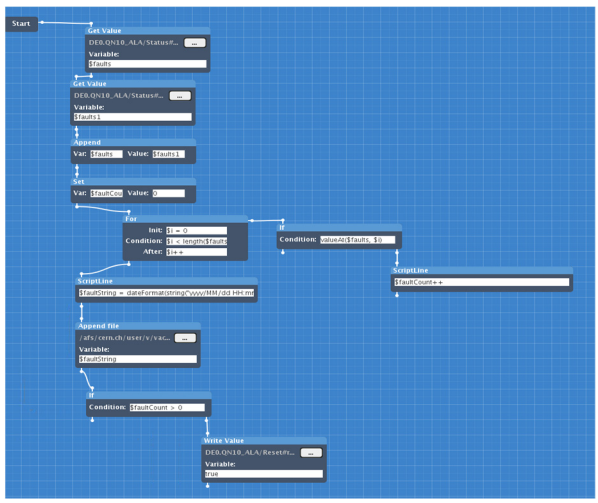


Figure 5: Example of Blueprints that read all the faults from a device, log them in a local folder and reset the device if there are any faults.

Blueprints also support Python scripting (see Fig. 6). Python code can be externally invoked or it can be directly integrated in Blueprints. A Python bridge is provided in order to be able to access all the Inspector features from Python scripts (including getting/setting variables, subscribing to properties and interacting with the GUI appearance) (see Fig. 7). This Python bridge allows scientists to run their simulation algorithms against real hardware with almost no need of code modification while Inspector graphical capabilities can show the results of the running code.

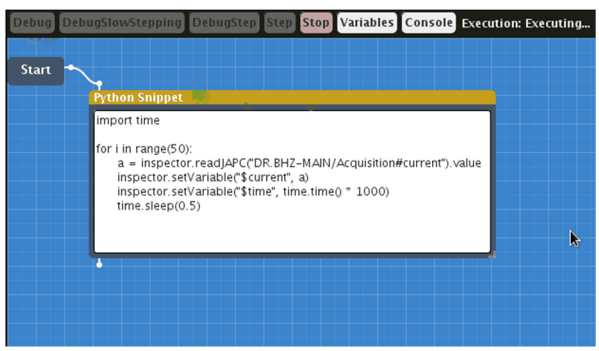


Figure 6: In This example Python is used to read a device. The value is returned into a Blueprint variable.

## INSPECTOR AS AN INTERFACE FOR AUTOMATION SPECIALISTS

The Programmable Logic Controllers (PLCs) are essential components of the CERN accelerator complex control system. PLCs typically offer custom communication protocols and therefore are difficult to integrate into a control system using a single communication strategy.

CERN *Software Infrastructure for Low-level Equipment Controllers* (SILECS) framework offers one API to communicate with most of the PLC brands and models. [14].

At CERN, several steps are needed to integrate a PLC into the control infrastructure. First, the PLC has to be programmed by an automation specialist. Then the PLC interfaces have to be detailed in an XML file in order to be exposed using SILECS. Finally, using SILECS stubs, a middleware process, integrating the communication protocol, has to be manually generated, compiled and deployed on a server. After these mandatory steps the device can be accessed through JAPC and Inspector. SILECS and all the other upper layers are tightly coupled to the PLC interface. This means that any modification to the PLC interface map implies a modification in all the mentioned layers - including manual code compilation and deployment.

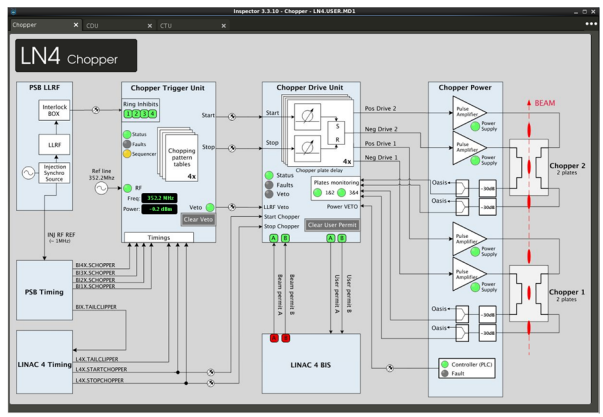


Figure 7: A low-level application with embedded Blueprints processing that shows the status of the control electronic and PLCs.

While Inspector was developed to use JAPC as a first-class data acquisition interface, it contains a modular Data Access Layer (DAL) that provides other access interfaces. A module for SILECS allows Inspector applications to access data directly from SILECS enabled PLCs. Therefore, automation specialists can develop graphical control interfaces to test the PLC functionality before performing its costly integration in the control infrastructure - where they have to manually create and deploy an acquisition server. The very same graphical interfaces can be reused and easily modified to use JAPC instead of SILECS.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR02



## INSPECTOR AS AN INTERFACE FOR NON-IT EXPERTS

A survey on Inspector has revealed that 60% of the users come from non-IT disciplines (like electrical, electronics, civil, chemical…). A majority of these users have no prior knowledge on visual programming and only have basic programming knowledge, usually acquired by scripting and running simulation software.

The survey shows that being able to create applications without the need to rely on expert developers for help is a very positive aspect. The stakeholders have no need to transmit any requirements to developers – which typically is not a lossless communication process, and there is no need for lengthy acceptance procedures. For simple applications it means shortened development cycle and significantly decreased cost for the organisation.

For users with absolutely no grip in programing, CERN has created a one-day course that covers the concepts and methodology of Inspector. It covers how to perform a full development cycle of creating and deploying an Inspector application. The feedback questionnaires of the course shown that most of the attendees find one day enough to be autonomous while some people would have been interested in doing another day specifically dedicated to complex aspects, such as Python integration.

## THE BENEFITS OF THE PROCRASTINATION PRINCIPLE

The procrastination principle dictates that one should wait for problems to arise before solving them. Operational applications are designed to check the status of the apparatus in the accelerator complex in real time. Many supervision applications are composed of synoptics that show *green* when everything is nominal and that start showing *red* when something goes wrong. The status of an apparatus can be complex to process. Parameters can be interdependent and, when it comes to beam production, there are qualitative aspects that can depend on an infinite number of causes.

Because Inspector applications are easily modifiable it has been noticed that operators were continuously adapting them according to their knowledge so that they react when a newly discovered set of conditions is reached. Like the *Wikipedia encyclopaedia*, Inspector operational applications become more accurate with time due to their collaboration openness. Authorized individuals can adapt and improve Inspector applications with little effort.

Based on this observation, and considering the importance of the operational applications, when it comes to saving modifications, history tracking and versioning, Inspector has direct integrations with version control systems for application storage.

## INSPECTOR APPLICATION MAINTENANCE

Inspector provides an environment for low-code application development. It also provides a runtime for executing these applications. Each application is saved as an XML file and contains no dependency. Every Inspector release is backwards compatible. Therefore Inspector not only lessens development effort but also reduces the maintenance costs. Existing applications don't need to be updated as long as the model - device and property interfaces, stays the same.

The Inspector development team takes responsibility for security updates, fixing defects and providing new features regularly. Unit and integration tests are used in development and deployment pipelines for quality and compliance assurance.

The deployment environment at CERN [15] also ensures that every user is always running the latest version of Inspector without manual and disrupting updates.

## OBSERVED BENEFITS OF USING INSPECTOR

The Inspector project was established at CERN over 4 years ago. Nowadays there are more than 600 mission critical Inspector based applications in use at CERN. Inspector is used in several accelerators throughout the complex, such as the LHC and Linac 4. Its uses range from replacing legacy applications in LabView or for the development of control applications for the SM18 test area [16] up to the creation of first-class control applications for new accelerators and subsystems.

Inspector is also the main software for development of dashboards and control applications for the Radio Frequency group. Inspector was used to successfully port and augment legacy LabView applications for the LHC RF as part of the upgrade of the high-level control of the superconducting cavities of the LHC [17]. This work, undertaken in 2018, included porting over 25 different LabView applications. It was completed in less than 6 months, from the conception of the project, understanding of LabView functionality, development of Inspector based applications and final commissioning. It was estimated that using GUI frameworks with programming languages (Java, Python, etc.) would amount to at least double the effort that would be required for the porting of such applications. Subsequent training and maintenance efforts would also be necessary for these types of applications.

## CONCLUSION

Developing portable applications with Inspector is fast and simple, look and feel is consistent and no maintenance is required if the hardware interfaces remain unchanged. Because Inspector core in under skilled developers' responsibility it globally offers enhanced security and reduces shadow IT.

Inspector allows non-IT experts to provide subsystems along with an engineering interface to test them. It allows accelerator operators to efficiently correlate information coming form different data sources. It allows anyone to improve any existing application and, finally, it allows scientist to easily test simulation code against real hardware.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR02

Inspector development has required 6 man/year of development. Maintenance and users support now only require 0.2 man/year. The 600 mission-critical applications can generates up to 8000 data subscriptions on the Inspector servers. The servers are responsible to handle the subscriptions to the final hardware but also to execute the soft-real-time computations required by the users. Even if the Inspector architecture is distributed, the actual deployment holds on one $4000 40 threads bi-Xeon blade computer with 128 gigabytes of RAM.

Inspector is a cost effective way to develop high-level control system portable applications that will never become a burden to the Organization as long as Java is supported.

## REFERENCES


[1] Nature, https://www.nature.com/news/the-future-of-the-postdoc-1.17253

[2] National Center for Science and Engineering Statistics, https://www.nsf.gov/statistics/srvydoctoratework

[3] Enquête sur les conditions d'emploi des personnels non permanents de l'Enseignement Supérieur et la Recherche, http://sciencesenmarche.org/fr/wp-content/uploads/2017/06/270617-bilan-enquete-SeM-Emploi-non-permanent.pdf

[4] C. Jones, Mcgraw-Hill, *Programming Productivity*. ISBN 978-0-07-032811-2, 1986.

[5] C. Jones, O. Bonsignour, The Economics of Software Quality, Addison Wesley, 2011.

[6] R. Jacob *et al*., "*Reality-based Interaction: A Framework for post-WIMP Interfaces*", in *Proc SIGCHI Conference on Human Factors in Computing Systems. CHI '08,* New York, NY, USA: ACM. pp. 201–210.

[7] B. A. Myers and M. B. Rosson, "*Survey on user interface programming*", in *Proc. SIGCHI Conference on Human Factors in Computing Systems, CHI '92*, pages 195– 202, New York, NY, USA, 1992. ACM.

[8] Forrester, https://www.forrester.com/report/Vendor+Landscape+The+Fractured+Fertile+Terrain+Of+LowCode+Application+Platforms/-/E-RES122549

[9] Gartner, https://www.gartner.com/doc/reprints?id=1-1FKNU1TK&ct=190711

[10] Z. Zaharieva, M. M. Marquez, and M. Peryt, "Database Foundation for the Configuration Management of the CERN Accelerator Controls Systems", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper MOMAU004, pp. 48-51.

[11] V. Baggiolini *et al.*, "JAPC - the Java API for Parameter Control", in *Proc. ICALEPCS'05*, Geneva, Switzerland, October 2005.

[12] P. Charrue *et al.*, "Role-Based Access Control for the Accelerator Control System at CERN", in *Proc. 11th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'07)*, Oak Ridge, TN, USA, Oct. 2007, paper TPPA04, pp. 90-92.

[13] V. Costa and B. Lefort, "Inspector, a Zero Code IDE for Control Systems User Interface Development", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 861-865. doi:10.18429/JACoW-ICALEPCS2017-TUPHA184.

[14] SILECS, https://be-dep-co.web.cern.ch/content/silecs

[15] L. Cseppento, V. Baggiolini, E. Fejes, Zs. Kovari, and N. Stapley, "CBNG - The New Build Tool Used to Build Millions of Lines of Java Code at CERN", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 789-793. doi:10.18429/JACoW-ICALEPCS2017-TUPHA163.

[16] N. Stapley *et al.*, "CERN's SRF Test Stand for Cavity Performance Measurements", presented at the 19th Int. Conf. RF Superconductivity (SRF'19), Dresden, Germany, Jun.-Jul. 2019, paper THP078.

[17] Y. Brischetto, V. Costa, D.C. Glenat, L. Arnaudon, D. Landr, "Upgrade of the Control System for the LHC High Level RF", presented at the *17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19)*, New York, NY, USA, Oct. 2019, paper MOPHA019.