# AN INTEGRATION TESTING FACILITY FOR THE CERN ACCELERATOR CONTROLS SYSTEM

N. Stapley, M. Arruat, J.C. Bau, S. Deghaye, C. Dehavay, W. Sliwinski, M. Sobczak,
CERN, Geneva, Switzerland

*Abstract*

A major effort has been invested in the design, development, and deployment of the LHC Control System. This large control system is made up of a set of core components and dependencies, which although tested individually, are often not able to be tested together on a system capable of representing the complete control system environment, including hardware. Furthermore this control system is being adapted and applied to CERN's whole accelerator complex, and in particular for the forthcoming renovation of the PS accelerators. To ensure quality is maintained as the system evolves, and to improve defect prevention, the Controls Group launched a project to provide a dedicated facility for continuous, automated, integration testing of its core components to incorporate into its production process. We describe the project, initial lessons from its application, status, and future directions.

## PROJECT BACKGROUND

### *Focus on Reuse and Standards*

At CERN, over the last few years, the focus of controls software has been shifting from many individual software projects towards providing a more generic singular control system capable of working with all CERN's accelerators. This can be seen with controls projects like LHC Software Architecture (LSA)[1] which was originally to provide the software only for LHC being adapted to provide the controls software for other accelerators like SPS. Furthermore, there has also been an increased re-use of software across projects. For example, LSA components are used in Injector Control Architecture (InCA)[2], a software project which is part of the PS complex renovation, and ancillary projects supplying common service components across all major projects. These components must interact with software devices, running on the FECs (Front End Computers normally running in VME crates), which control the equipment. Devices are provided by equipment specialists using the now common Front-End Software Architecture Framework (FESA)[3]. For controls hardware, although there is an effort to consolidate on a limited set of FECs and communication protocols, different generations and types will always co-exist. Expanding the notion of components to general sense – hardware, re-usable units of software, libraries, drivers and operating systems – there are often not only multiple versions running concurrently in operation, but also *multiple combinations* leading to a complex fusion of components deployed in operation at any moment in time.

### *The Case for Testing*

Testing presents a challenge for any project – how to deliver new features and bug-fixes into an increasingly intertwined set of operational components without any detrimental impact on operations. At best this problem can cause delays and increase the necessary effort for both the evolution of the control system, and for the project's development speed. For large software projects this is a known and well documented point [4], often made with the fact that doubling requirements squares complexity. It implies that an adequate and realistic quality assurance regime is in place to ensure components are validated together as part of the operational control system *prior to* deployment. Within this testing is a critical success factor.

Typically there is a perceived fear of slowing down progress by the addition of "extra" procedures. In reality has been shown *in all cases* that "quality is free"[5] – more than paid for by increased reliability of products and services. These lead to a reduction in the effort spent on problem resolution, time dedicated to operational support, and fixing costly mistakes. In essence the focus of an engineer's energy is shifted to where it is best placed – progressing with new work and in a more sustained manner, rather than heroically debugging "completed" work. In fact, quality levels in organisations can be crudely judged by the amount of corrective effort that is spent on work, which of course is a waste [6].

So considering the above, the Controls Group launched the System Testbed Facility (CSTF) project to automate integration and system testing. Its purpose is primarily to validate the control system components, both hardware and software, as a single cohesive product that can be certified together. It is an important step of defect prevention, as is unit testing. Projects do, and are expected to, test their components individually first before being integrated and tested as part of the control system.

### *Testbed Components*

The Controls System Test Facility aims to represent the operational environment as best it can. The topology of the CERN Control system can be described roughly as a 3-tier architecture (see Figure 1). The lowest tier is made up of many FECs exposing devices for control. These have an operating system, hardware drivers and libraries.

The middle-tier is a set of centralized servers running software to provide services to all tiers. And finally consoles for software concerned with display and user interaction.

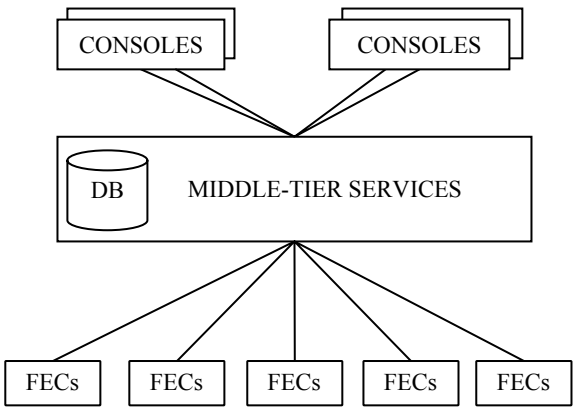


Figure 1: General topology of control system.

To mimic this, the CSTF contains an instance of every type of FEC and middle-tier server running their current operational operating system, libraries and drivers. The timing system is also included with the ability to simulate cycles from different accelerators. Each of the FECs has a FESA test device installed which is designed to help exercise the control system. The scope of the CSTF is limited to testing the main "technology stack", so there is no equipment attached to its FECs, and currently no consoles. What is tested are the core software components, communication protocols and libraries on the current hardware platforms upon which top level operator applications and accelerator equipment rely on for controls.

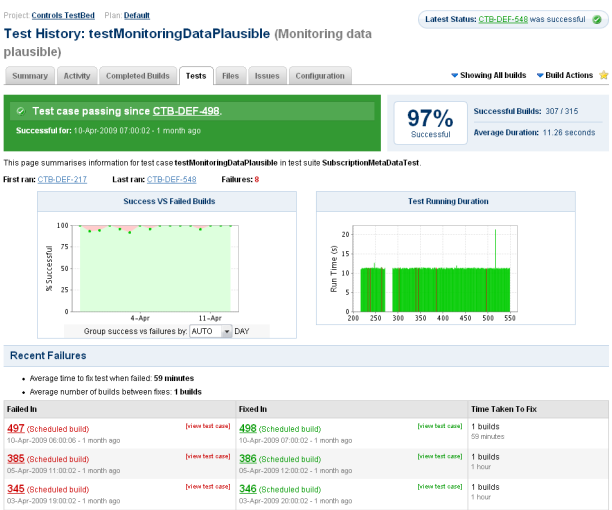


Figure 2: Overview report generated by Bamboo.

### *Running tests*

The tests are run by a web-based Continuous Integration Server (Bamboo from Atlassian[7]). Bamboo provides an environment for automated build and test execution – it schedules the tests, based on a policy, such as time periods or external triggers, runs them and provides reports. It can show which tests were successful, statistics like time taken, and the history of any particular test including logs or an overall summary. This system was chosen since it integrates well with other tools and in use for continuous integration and testing of Controls Java applications. Figure 2 provides an idea of an overview screen.

The tests are implemented as Java JUnit tests contained in a CO software project where the dependencies on other components and their versions can be specified, and a list of the test devices is supplied to test against thus exercising the whole controls stack from the top.

## INITIAL LESSONS AND STATUS

In the summer an initial version of the CSTF was put in place, based on the specification tests from the InCA project and the FESA test device.

### *Identifying Components*

The original concept was to run the tests when any core component or its dependencies changed. However this proved to be more difficult than originally foreseen since not all parts of the control system are identifiably versioned. For instance some components are controlled by external vendors, and even internally although care is taken to ensure source code is versioned, many final artifacts (e.g. executable binaries) are not. Consequently as the initial operation of the SCTF could not detect updates of all components, it was decided to run the rests continuously instead. A further point to address is that without the ability to identify all the artifacts tested there is no way to produce a certified component set.

### *Continuous Testing*

Although the repeated testing was automatic, at first sight it was considered pointless. After all, the intention was only to repeat the tests when a known change was introduced into the control system. After a few weeks, however, an intermittent error in the control system was exposed. As it turned out testing the same code repeatedly so often allowed this to be seen once or twice a week, and it probably would not have been exposed with a single test run after any update. Although the error was found to be a differing interpretation of specifications between two interacting components – a type of defect that integration testing and the CSTF is designed to find – it proved that test on change was not the better strategy.

### *The Actual Tests*

The CSTF runs primarily with specification tests provided by the InCA project. These are a good starting point because they contain the expectations of InCA as a client of the other controls system components, but they are not a complete testing suite. There is much scope to improve the current test set but it requires a certain level of resources. The danger being that if the test set does not evolve with the control system, it will lose its relevance lowering the value of the CSTF. One discovery was that it is actually quite hard to write good tests! There is no overall single control system specification to develop a set of conformance tests from, instead specifications are the responsibility of the individual projects and therefore focused on their own requirements. However, there are plenty of other sources such as the organisational history of issues from meeting minutes, operator's logbooks, and the defect tracking system for failures reported by operations which can be analyzed as well as how operators complete acceptance testing for updated systems and the test code that projects write. Developing Integration tests also requires a mindset independent from any particular project being worked on by the developer; and of course tests need to be tested too!

As stated before, the test set is implemented as JUnit tests which is very convenient for code tests, but less so for overall system testing. Other (mainly commercial) tools exist which develop tests from specifications and data supplied by users.

### *Software Releases*

At the moment, in the controls software release system, components are released individually, and are then available to be deployed. Until more recently this worked well as projects tended to be less dependent on each other and historically the control system was more a set of disparate applications. The CSTF currently tests newly released components with the current operational set. Now with trend towards increased component inter-dependencies, the CSTF would be more useful executing tests earlier in the development activity stream; for two main reasons. The first is that integration testing takes place when a component is released (but not necessarily deployed), and hence any defect found requires a return to the older version with the hope that it was not accidentally deployed in the mean time. The second is that, on some occasions, component updates have an impact on dependent components or clients (e.g. no backwards compatibility). A transactional multi-component approach to releasing would now be more appropriate. The CSTF is a tool, and as such can be placed anywhere in the software development process. Aiming to place it earlier, before release, would help prevent defects from being released at all where there is a lower associated cost for fixing them (typically 60 times lower[8]). This implies that components are to be available for testing before release in conjunction with its dependents.

## FUTURE DIRECTIONS

In the previous section, the experience from the initial operation of the CSTF discovered potential improvements which can be broken into 3 simple aspects. That is "the right tests applied to the right components at the right time". Many of these improvements are beyond the mandate of the CSTF project, and consequently have been taken up as group-wide concerns. The CSTF experience has also highlighted differences in the way projects approach software development. Rather than leave the CSTF project to overcome or work around these, the Controls group has preferred to try to standardise these instead. Consequently, there is an initiative to help identify all components at all levels in a common way, and the outcome of this will lead to the ability to certify a complete set of components together. Another initiative is investigating coordinating releases of components together and providing release candidates. The CSTF will then test all release candidates together as an atomic set and they can then be released together.

The tests are perhaps the most difficult part, as good tests are the key to the CSTF success. A view sometimes expressed is that only operational testing "will discover all the bugs". The CSTF is only part of the overall testing chain and does not expect to find more than 70% of testable defects. It is not a replacement for unit or acceptance testing – the point is it can be used to achieve a significantly higher reliability factor, reducing defects in operation; and this is its eventual measure of success – a more reliable control system.

## REFERENCES

[1] G. Kruk *et al*., "LHC Software Architecture [LSA] – Evolution toward LHC Beam Commissioning", ICALEPCS'07, Knoxville, USA, Oct 2007, RPPA03, p. 526 (2007).

[2] S. Deghaye *et al.*, "CERN Proton Synchrotron Complex High-Level Controls Renovation", ICALEPCS'09, Kobe, Japan.

[3] M. Arruat *et al*., "Front-End Software Architecture", ICALEPCS'07, Knoxville, USA, October 2007, WOPA04, p. 310 (2007).

[4] S. McConnell, "Code Complete, A Practical Handbook of Software Construction", Microsoft Press 2004, "How size affects construction", C27.

[5] P. Crosby, "Quality is still free: Making Quality Certain in Uncertain Times", McGraw-Hill 1996.

[6] M. Poppendieck, "Implementing Lean Software Development", Addison-Wesley 2006, C4.

[7] http://www.atlassian.com/software/bamboo

[8] R. Pressman, "Software Engineering; A Practitioner's Approach", McGraw-Hill 2004, "Cost Impact of Software Defects", S8.4.1.