BRINGING QUALITY IN THE CONTROLS SOFTWARE DELIVERY PROCESS

Z. Reszela, G. Cuni, C. M. Falcón Torres, D. Fernandez-Carreiras, G. Jover-Mañas, C. Pascual-Izarra, R. Pastor Ortiz, M. Rosanes Siscart, S. Rubio-Manrique, ALBA-CELLS Synchrotron, Cerdanyola del Vallès, Spain

Abstract

The Alba Controls Section (ACS) develops and operates a diverse variety of controls software which is shared within international communities of users and developers. This includes: generic frameworks like Sardana [1] and Taurus [2], numerous Tango [3] device servers and applications where, among others, we can find PyAlarm [4] and Panic [5], and specific experiment procedures and hardware controllers. A study has commenced on how to improve the delivery process of our software from the hands of developers to laboratories, by making this process more reliable, predictable and risk-controlled. Automated unit and acceptance tests combined with continuous integration, have been introduced, providing valuable and fast feedback to the developers. In order to renew and automate our legacy packaging and deployment system we have evaluated modern alternatives. The above practices were brought together into a design of the continuous delivery pipelines which were validated on a set of diverse software. This paper presents this study, its results and a proposal of the cost-effective implementation.

INTRODUCTION

The ACS designed, constructed and maintain the control systems required to run the facility. The core software components of the Alba control systems, like for example Tango – the middleware framework for building distributed control systems, Sardana – the scientific SCADA oriented to the experiment control and Taurus – the GUI library, are fruits of a collaborative effort of many institutes including Alba [6]. Peripheral components, like Tango device servers, Sardana controllers and macros or Taurus GUIs were either developed in-house or reused from the public repositories. Most of them are developed in Python.

Previously, the software development processes were mainly organized around one-person projects that made it difficult to conduct design questioning discussions and limited the knowledge flow. The software testing was neither formalized nor tried to be automated. This made the project transfers between the developers more difficult, for example on a developer leave. Newcomers, without the complete knowledge of the project, could not feel confident when introducing a change in the code without a way of testing it at the unit or at the system level. In some extreme cases this lead to development downtime periods, to abandoned projects or simply to buggy releases. Subversion (SVN) was the standard version control system (VCS) which did not encourage working branch-per-feature mode. Untraceable commit histories were not helping to enter into the project dynamics. The software packaging and deployment were done manually what is neither interesting nor motivating for the engineers. All the above problems and difficulties were not helping in reducing the long lead time – from the scientist request to the successful use of the software in the experiment.

DEVELOPERS COLLABORATION

Almost two years ago the ACS decided to introduce changes in the software development organization. The in-house projects were transformed from the individual to the group-based efforts. Furthermore, in the case of the two core and initially ACS's internal projects Sardana and Taurus (started at Alba in the previous decade) community-driven development and organization models were introduced. This had an impact on the following parts of the software development process.

Code Design

Internally, developers were organized in groups of 4-6 members – the *Scrum* teams [7]. A mixture of senior and junior developers were selected to build each team. The knowledge transfer activities became a second plan and continuous process. Information about the projects, previously restricted to the privileged project owners, quickly equalized among the team. Agile design practices were introduced: avoidance of upfront designs and plans and promotion of iterative and incremental developments. The Scrum activities brought many interesting design discussions that helped to achieve a better quality of the released products.

In parallel, the Sardana and Taurus project development and decision-taking was opened to a community composed mainly by synchrotrons similar to Alba (DESY in Germany, MaxIV in Sweden Solaris in Poland and ESRF in France), as well as by other institutions, companies and individuals (mostly within the Tango collaboration) who are basing their own developments in them. All these entities actively participate in the community activities. The community model requires remote collaboration tools. Sardana and Taurus are hosted on the Souceforge [8] platform and use a number of provided tools (e.g. issue tracker, mailing lists, wikis, etc.) in the code design processes. Discussions about the critical improvements and modifications are organized and formalized around public processes called Sardana Enhancement Proposal (SEP) and Taurus Enhancement

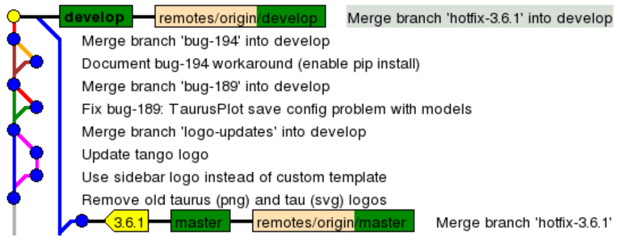
ISBN 978-3-95450-148-9

Proposal (TEP) inspired in similar workflows from the Python [9] and Debian [10] projects.

Code Control

Since many developers started contributing to the projects the SVN became a limitation in many ways. Other systems were evaluated having the following reasons in mind: easier branching and merging (necessary in scattered organizations like ours) and easier tools and workflows for the code validation. Git won this competition bringing many other benefits such as its distributed architecture, better performance and the possibility of much cleaner history of commits with less effort. Hence Sardana and Taurus projects were migrated from SVN to Git, and the ACS started using Git for the newly created internal projects.

Sardana and Taurus branching rules were formalized according to *gitflow* [11,12] (Fig. 1). Two main branches and the core for the project are: the *develop* branch – where all the developments take place or get integrated into and the *master* branch – which always represents the latest production-ready state of the project. Each commit to the master branch gets tagged following the *semantic version system* (semver) [13] consisting on three dot-separated fields: major, minor and patch (Fig 1.). Increment in the major field indicates backwards incompatible changes in the API. The minor field increments when new functionality is added in a backwards compatible manner. And finally the patch field increments with the backwards compatible bug fixes. Other supporting branches are: the *feature* branches – where development of the SEPs, TEPs or feature requests take place, the *release* branch – the intermediate branch between the develop and the master states as well as the *hotfix* branches where the critical bugs in the master branch are fixed.



```
develop  remotes/origin/develop    Merge branch 'hotfix-3.6.1' into develop
Merge branch 'bug-194' into develop
Document bug-194 workaround (enable pip install)
Merge branch 'bug-189' into develop
Fix bug-189: TaurusPlot save config problem with models
Merge branch 'logo-updates' into develop
Update tango logo
Use sidebar logo instead of custom template
Remove old taurus (png) and tau (svg) logos
3.6.1  master  remotes/origin/master    Merge branch 'hotfix-3.6.1'
```

Figure 1: Extract from the Taurus git history demonstrating use of gitflow and semver rules.

Code Review

Together with collaborative software development, systematic code review practices were introduced. They rely on examination and validation of the contribution, usually done before checking it into the repository. In the case of the internal projects, lightweight peer reviews had proven to be a great way of improving the quality of code. Apart from that they help to equalize the technical knowledge within the team and reduce the information silos across the developers. The Sardana and Taurus projects apply more formal code reviews. All the code contributions are evaluated on the public forum but only the integration managers (representatives of each of the community institutes) are allowed to push the code into the canonical repository. The only tools used in this process are just a few git commands and the developers mailing list. Based on the current experience we can affirm that the quality of these projects has improved thanks to the code reviews. At the same time we can observe that the limited time of the integration managers is a bottleneck for patch integration.

TESTING

No testing strategy existed for the software projects developed and maintained by the ACS. SEP5 [11] established the common testing strategy for Sardana and Taurus. The following best practices are based on its results and more than a year experience with software testing.

Tests should be written before developers start work on the features that they test. Together, these tests form an executable specification of the behaviour of the system, and when they pass, they demonstrate that the functionality required by the users has been implemented correctly. The automated test suite should be run by the continuous integration (CI) service every time a change is made to the application – which means that the suite also serves as a set of regression tests. This strategy applies ideally to new projects, where with prior selection of the testing technology and the CI platform, developers could start writing and applying automatic tests in the process right from the beginning. However mid or legacy projects, like for example Sardana and Taurus, require a certain variation of the approach. The best is to start automating the most common, important, and high-value use cases of the application. Based on this selection, "happy path" tests covering these high-value scenarios should be automated. The rest of the scenarios should initially be tested manually. They should be automated only when one discovers that the same function is tested manually more than a couple of times.

Jenkins [14] was selected as the general CI system for the ACS projects mainly because of its big community of users, a broad and continuously growing set of plugins, and simplicity in setting them up and running.

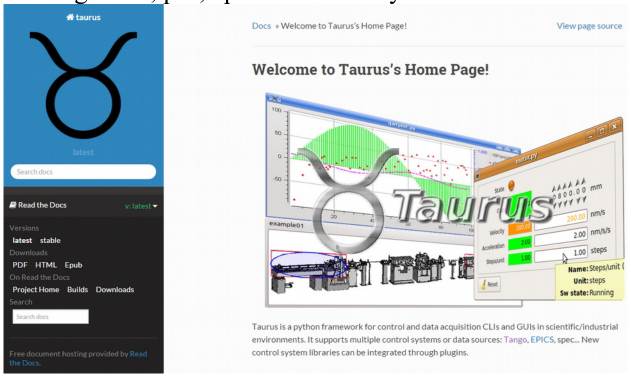
The main purpose of the CI service is to test the software on each commit, providing fast feedback to the developers. Our use of Jenkins was extended to the unique registry of all the software maintained by the group. Hence even the projects which are not actively developed by the group or the external projects have their corresponding Jenkins jobs. These jobs are not triggered at each commit but their role is to automate the build processes and the integration tests with our control system.

CONTINUOUS DOCUMENTATION

Sardana and Taurus projects use Sphinx [15] to create documentation. Sardana and Taurus documentation was originally hosted on the Alba's internet servers. This setup required manual builds and deployments on every update. Mainly due to the work overheads the documentation update frequency gradually decreased to twice per year, concurring with the biannual releases.

The documentation build processes of both projects were adapted to make them independent of specific libraries e.g. PyTango (Python language binding to Tango). This was achieved via a custom Python modules *mock* generator. Finally, both projects migrated thir documentation to the Read The Docs (RTD) [16] platform (Fig. 2), bringing the following benefits:

- Documentation gets built on every commit, early validating its correctness and notifying developers about any errors.
- Maintenance of the servers and the necessary software is outsourced to RTD.
- Several versions of the documentation e.g. *stable* or *latest* are available in a unique place.
- The documentation is available in different formats e.g. html, pdf, epub and is easily searchable.



Free document hosting provided by Read the Docs.

Welcome to Taurus's Home Page!

Taurus is a python framework for control and data acquisition CLIs and GUIs in scientific/industrial environments. It supports multiple control systems or data sources: Tango, EPICS, spec... New control system libraries can be integrated through plugins.

Figure 2: Taurus latest documentation available on RTD.

CONFIGURATION MANAGEMENT

The ACS manages all the software under maintenance with the bliss system [17]. The bliss system, developed by the ESRF, is a rpm-based packaging and Software Configuration Management (SCM) tool. It comprises two applications: the blissbuilder and the blissinstaller, and centralizes information about the packages and the hosts in a mysql database. Its main advantages are: the offline access to hosts' configurations and an intuitive to "non-packaging experts" graphical way of defining and creating packages. While bliss has served very well over its many years of use, it shows limitations when it comes to automatic package creation and deployment. First, all the package definition is spread in the bliss database tables and it is not possible to maintain it in the project code repository. This limits the creation of the project rpm package to the bliss system users only. The automatic package creation is not fully customizable, and requires modification of the package metadata in the database. However the biggest limitation of bliss comes with the configuration management and the automatic deployments (bliss does not provide a way to configure a group of hosts neither supports the Windows platform). While it seems possible to implement all the missing features in the bliss system (it is written in Python) or develop the complementary scripts, we decided to evaluate alternative public and widely used products.

SCM Tools

Many SCM tools exist, with the most popular choices being Puppet, Chef, Ansible [18] and Salt [19], and all of them are successfully used in many different organizations. In the process of comparison we had in mind the following aspects: precedence was given to free and open source projects, ideally developed in Python – the widest spread programming language in the group. Apart from that the ideal candidate was expected to support both Linux and Windows platforms in the most seamless way possible. Definition of the hosts configuration, and the possibility of applying them to different groups of hosts was also considered as an asset. Finally, the simplicity and the smoother learning curve for the ACS were also considered.

A closer look was given to the two Python based candidates: Salt and Ansible. They give a possibility to use the repository integration modules like the ones for apt, zypper or yum and allow operating system agnostic definitions of the hosts configurations. Salt architecture is based on a single master and distributed minions which exchange messages when necessary. However the use of minions is optional, and Salt could fallback to execute commands via ssh if necessary (a mixture of both solutions in the same system is possible). Ansible and Salt offer very similar features. Salt could eventually bring benefits in the future thanks to its higher scalability and the agent based architecture. In its favour speaks that the Alba IT Systems Section uses Salt for SCM of the High Performance Computing Centre of the Alba Synchrotron.

Packaging

Migration from the bliss system to Salt would eventually require an alternative way of creating software packages. This seems quite simple in case of the Python based projects since the standard packaging modules allow creation of rpm and deb packages for the Linux platform and exe and msi installers for the Windows platform. Sardana and Taurus already use distutils [20], and as a proof of concept creation of the rpm packages was successfully tested. In case of the Windows platform the msi format proved to be a better choice than the exe, since it allows the unattended installations necessary by the SCM. The migration from bliss would also require setting up package repositories where the packages would be uploaded and from where the SCM tool would pull the packages for installation.

CONTINUOUS DELIVERY

Agile software development together with the continuous delivery aims to transform a concept into working software as fast as possible. Continuous delivery is based on fully automated, reliable, repeatable and constantly improving software delivery pipelines. Each change in the project code should trigger the pipeline execution and in case of success deliver a deployment-ready product. The negative result of an intermediate pipeline stage must break the pipeline execution and be immediately reported to developers who should stop the current work and fix the breaking change. Each project actively developed by the ACS could have its continuous delivery pipeline, initially comprising three stages: commit, acceptance and user acceptance. These stages should be executed sequentially only if the previous stage ended successfully. Each stage could be divided into parallel run jobs if necessary.

Based on the experimental implementation of the pipelines for Taurus (Fig. 3) and Sardana the following tools and setups are recommended. Jenkins works as the pipeline orchestrator where each job represents one pipeline stage. Jobs are interconnected and trigger the downstream jobs while the pipeline advances.

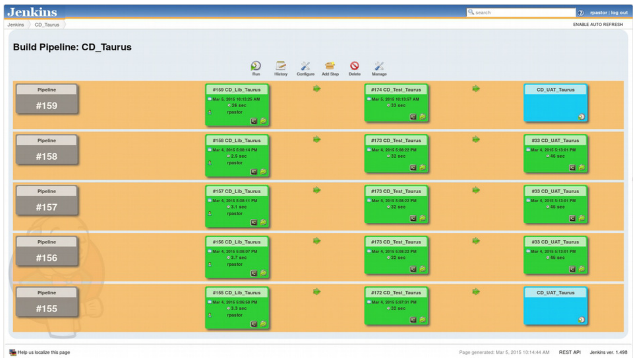

Jenkins
search
rpastor | log out
Jenkins
CD_Taurus
ENABLE AUTO REFRESH
Build Pipeline: CD_Taurus
Run
History
Configure
Add Step
Delete
Manage
Pipeline #159
#159 CD_Lib_Taurus Mar 5, 2015 10:13:25 AM 26 sec rpastor
#174 CD_Test_Taurus Mar 5, 2015 10:13:57 AM 33 sec
CD_UAT_Taurus
Pipeline #158
#158 CD_Lib_Taurus Mar 4, 2015 5:08:14 PM 2.5 sec rpastor
#173 CD_Test_Taurus Mar 4, 2015 5:08:22 PM 32 sec
#33 CD_UAT_Taurus Mar 4, 2015 5:13:01 PM 46 sec
Pipeline #157
#157 CD_Lib_Taurus Mar 4, 2015 5:08:11 PM 3.1 sec rpastor
#173 CD_Test_Taurus Mar 4, 2015 5:08:22 PM 32 sec
#33 CD_UAT_Taurus Mar 4, 2015 5:13:01 PM 46 sec
Pipeline #156
#156 CD_Lib_Taurus Mar 4, 2015 5:08:07 PM 3.7 sec rpastor
#173 CD_Test_Taurus Mar 4, 2015 5:08:22 PM 32 sec
#33 CD_UAT_Taurus Mar 4, 2015 5:13:01 PM 46 sec
Pipeline #155
#155 CD_Lib_Taurus Mar 4, 2015 5:06:58 PM 3.3 sec rpastor
#172 CD_Test_Taurus Mar 4, 2015 5:07:31 PM 32 sec
CD_UAT_Taurus
Help us localize this page
Page generated: Mar 5, 2015 10:14:44 AM
REST API
Jenkins ver. 1.498

Figure 3: A proof-of-concept continuous delivery pipeline of Taurus project.

The commit stage is triggered on each change in the VCS. The commit stage should run the unit tests of the project and execute the static code analysis and in case of success build the software packages. Finally, all the packages get uploaded to the repository. It is very important that from now on, all the subsequent stages always use the same package created in the commit stage.

The acceptance test stage should take place in an environment as similar to production as possible. This stage should start from the package deployment to the acceptance test environment using the SCM, as it would be deployed to production. If the software works in production on various platforms e.g. Windows and Linux, the acceptance tests should also be performed on all of them. Finally the automated acceptance tests should be executed. The acceptance tests may require a specific configuration (also maintained under the VCS) which should be applied to the acceptance test environment before the tests execution. Preparation and maintenance of the acceptance test environments may be a tedious and error prone job. Execution of the acceptance tests in the Docker [21] containers showed to be a great solution to these problems. Docker is a platform for developing, shipping, and running applications using the container virtualization technology. The idea behind it is to maintain the Docker images for each of the acceptance test environments and spin them up on demand of the pipeline execution. This solution provides lightweight, reliable and isolated testing environments occupying the resources only when needed. Docker integrates well with Jenkins via plugins and one of them allows to seamlessly use Docker containers as the Jenkins slave nodes.

Successful acceptance stage should notify developers that the package is ready for the user acceptance tests. These tests should be executed manually following a well-defined testing scenarios and it is very important to do that in an environment as similar to production as possible. While the tests must be executed manually the preparation of the testing environment should be automated thanks to the SCM. Based on the user acceptance test results a decision is taken if a package is production ready or not. Of course executing the user acceptance tests at each commit could become expensive, so this stage should be done on demand.

NEXT STEPS

Sardana and Taurus projects could already apply the continuous delivery strategy to their biannual releases. Ideally their pipelines should be accessible by the whole community of developers, both in and outside of ALBA. This may be solved by using cloud providers for the continuous delivery tools, but it has not been investigated yet. While the decision of the eventual migration to the new SCM is blocked by still very shallow knowledge about the software packaging, it also depends on the upgrade of the general platform of the Alba control system. Online code review platforms may bring new quality to the current review processes, making them more accessible to the developers and reducing the workload on the integration managers.

ACKNOWLEDGEMENT

We would like to thank the rest of the Alba Controls Section and especially to Roberto Homs, Daniel Roldan, Jordi Andreu, Fulvio Becheri and Sergi Blanch, and the Taurus and Sardana community members for their active work in this project. We would also like to thank the IT Systems Section of Alba and especially to Sergi Puso, Marc Rodriguez, Mario Diaz and Antoni Pérez as well as the MIS Section and especially to Daniel Salvat for their valuable feedback.

REFERENCES

[1] T. Coutinho et al. "Sardana, the Software for Building SCADAs in Scientific Environments", ICALEPCS2011, Grenoble, WEAAUST01.
[2] C. Pascual-Izarra et al. "Effortless Creation of Control & Data Acquisition Graphical User Interfaces with Taurus", ICALEPCS 2015, Melbourne, THHC3O03.
[3] Tango website: http://www.tango-controls.org
[4] S.Rubio-Manrique et al., "Extending Alarm Handling in Tango.", ICALEPCS 2011, Grenoble, MOMMU001.
[5] S.Rubio-Manrique et al., "PANIC, a suite for visualization, logging and notification of incidents.", PCaPAC 2014, Karlsruhe, FCO206.
[6] Alba website: http://www.albasynchrotron.es
[7] G. Cuni et al. "Introducing the SCRUM Framework as Part of the Product Development Strategy for the ALBA Control System", ICALEPCS 2015, Melbourne, MOD3O04.
[8] Sourceforge website: https://www.sourceforge.net
[9] Python Enhancement Proposals: https://www.python.org/dev/peps
[10] Python Enhancement Proposals: http://dep.debian.net/deps/dep0
[11] Sardana Enhancement Proposals: http://sf.net/p/sardana/wiki/SEP
[12] Gitflow branching model website: http://nvie.com/posts/a-successful-git-branching-model
[13] Semantic versioning system website: http://http://semver.org
[14] Jenkins website: http://jenkins-ci.org
[15] Sphinx website: http://sphinx-doc.org
[16] Read The Docs website: http://readthedocs.org
[17] The ESRF Beamline Control Unit website: http://www.esrf.eu/Instrumentation/software/beamline-control/BLISS
[18] Ansible website: http://www.ansible.com
[19] Salt website: http://saltstack.com
[20] Distutils documentation website: https://docs.python.org/2/library/distutils.html
[21] Docker website: http://www.docker.com

ISBN 978-3-95450-148-9