DOI: 10.17199/NOBUGS2016.42

Community driven scientific software projects: lessons learned on tools and practices

C Pascual-Izarra *, C Falcón-Torres, Z Reszela, G Cuní, D Fernández-Carreiras, G Jover-Manas and M Rosanes-Siscart
ALBA-CELLS Synchrotron, Cerdanyola del Vallés, Spain
E-mail: cpascual@cells.es, ctgensoft@cells.es

Abstract. On the one hand, science is about openness and collaboration and, on the other hand, open and community-driven software projects have demonstrated to yield more generic and resilient solutions thanks to the diverse users' needs and feedback. Yet, the development of many scientific software projects (even free/open-source ones) is still managed in a closed way, typically by one or few people from a single institution. In some cases this is just due to the lack of knowledge or confidence on the already available tools and practices for collaborative development. In this work we share our experience on coordinating the transition of the Taurus and Sardana projects from being in-house developments to being driven by an international and diverse community. The initially established rules are constantly evolving aiming to reach continuous delivery while maintaining good software quality. The selected contribution workflows, testing strategies as well as the software and documentation delivery tools are described in detail, and the benefits of this kind of organization as well as potential pitfalls and lessons learned are discussed.

1. Introduction

It is generally accepted that software projects whose development is driven by a community (as opposed to a closed group of developers) tend to become more resilient (e.g. against the eventual leave of a main author), have better code quality (more bugs spotted and corrected), attract more contributions (making them more generic) and, as a consequence, get a wider exposure [1]. But this comes at some cost: first, one needs to provide tools for coordinating a geographically dispersed community of developers; second, time must be devoted to document procedures and policies and for supporting new developers in order to keep a coherent code quality and style; third, the contributions require to be peer-reviewed before being incorporated and, finally, a much stricter policy regarding the Application Programming Interface (API) is required to avoid breaking other people's developments.

While these are general good practices, closed-team development is more forgiving in taking shortcuts and exceptions to them in the name of short-term benefits. Also, some authors may fear the loss of control over their "pet" projects, or may not be familiar with the tools and practices available for collaborative development. This may explain why so many scientific software is developed in a non-open way (even if it uses free/open-source licenses).

In the rest of this paper we will try to encourage scientific software authors to consider a collaborative development model by showing the example of the Sardana [2, 3] and Taurus [4, 5] projects, which successfully transitioned from in-house to community-driven development.

2. Background

Sardana and Taurus are projects initially developed to satisfy the needs for the ALBA synchrotron beamline and accelerator control system [6]. Since their inception around 2009 (or earlier) they were licensed as Free/Open Source software, but their development was internal to ALBA and the code repository was only accessible from within the site.

In 2011, the code repositories for both Sardana and Taurus were moved to SourceForge [7] in order to facilitate adoption by third parties, but the development model continued to be essentially in-house and the support to other facilities was considered only a last priority.

Still, some other laboratories expressed their interest in using Taurus and Sardana, and as a consequence the move towards a community-driven project was proposed in order to share both the decision-making and the load of the development.

3. Opening the development

3.1. Seeding the Sardana Community

Following the example of the Tango Community [8, 9], which was initially based on a Memorandum of Understanding (MoU), some attempts were done for signing a formal agreement between the interested facilities. But the review and discussion on this MoU got delayed and the community still has no institutional agreement (nor seems to require one so far).

On technical grounds, the first efforts were put to formalize the processes on how to contribute to the project. For shared design decisions, the Sardana Enhancement Proposal (SEP) [10] process was agreed, inspired on Debian Enhancement Proposal (DEP) [11]. The SEP process defines the discussion workflow for promoting ideas into actual Sardana or Taurus features or policies. Regarding code contributions, a process for *public* code review was discussed and agreed [12]. It not only promotes the quality of the code but it also benefits the developers who can easily share their knowledge. Following the example of the Linux kernel project, we based the review process on patches sent and discussed by email (which only requires an email client, and does not require to log into any external service). But, in retrospective, we found two main issues: for the integrators it is cumbersome, especially when the contributions accumulate in the lists; for the contributors, it requires to learn and follow some conventions when emailing or discussing their contributions.

A biannual release cycle was selected according to the needs of the involved institutes. Each of the releases, preceded by extensive manual tests on various platforms, ensure a periodic checkup of the project. In addition, the pending issues may be assigned to a given release, improving the project transparency.

Two mailing lists, one dedicated to the developers and the other to the users were created. They are the main discussion and support channel, with lots of participants. As a negative note, the traffic generated by the code review process tends to bloat the developers list.

Additionally to the written communication, the community members organize open annual Sardana Meetings, where the technical aspects and roadmaps are discussed. Initially, video conferences were considered as a complement to the presential meetings but they are not so common for now.

3.2. Tools and methodologies adopted for community development

The Git [13] distributed version control system facilitates workflows for collaboration between disperse groups of developers. For this reason, both Taurus and Sardana migrated their code from a SVN repository into a Git one within the SourceForge platform and adopted the gitflow rules [14], which fit well with the biannual release cycle.

In the last years Github [15] became increasingly popular. Sardana and Taurus, similarly to the Tango community projects, will soon migrate from SourceForge to GitHub, attracted by its user-friendliness and the possibilities enabled by its tools and related services to better

coordinate a community. The Github pull-request and the code-review workflows will make the contribution process more agile and hopefully bring more participants.

Performing manual tests of the overall project on each incoming contribution resulted to be time consuming and error prone. Therefore, the SEP5 [16] was proposed, defining some basic rules on how to develop consistent automated tests. Since providing full test coverage for already-existing projects such as ours is not practical [17], we focused on developing happy path tests for existing code, while encouraging Test Driven Development (TDD) for new features. Also, when working in a team, we try to alternate the person writing the tests and the one implementing the functionality.

The automated tests deliver their full benefits when using them together with the Continuous Integration (CI) practice. Employing disposable Docker [18] containers as the underpinning technology, facilitates the design and implementation of the testing infrastructure and helps with test isolation. Having a *public* CI service is crucial in collaborative projects. This could be achieved by exposing an internal CI server to the external users (in ALBA we use Mr. Jenkins [19] internally). However, we opted to set up the Taurus and Sardana CI on external platforms instead (Travis [20] and Appveyor [21]), mainly because: a) we reduce maintenance efforts related to supporting account creation, security, etc; b) it makes the communitary infrastructure less dependent on one specific facility; and c) they integrate smoothly with Github and Docker.

The up-to-date and verbose documentation is very important when it comes to sharing the project among many institutes. The Sardana and Taurus documentation is written using Sphinx [22] and was originally hosted on ALBA's internet servers. This setup required manual builds and deployments on every release, which was tedious and impractical. Consequently, we migrated the documentation to the Read the Docs (RTD) platform [23] which builds new docs after each commit to the Git repository and, at the same time liberated us from the infrastructure maintenance. However, while the benefits of the continuous documentation practice are unquestionable, the adequacy of RTD for projects such as Sardana or Taurus is being reconsidered for the following reasons: a) it forces us to implement and maintain mocks for the various non-pure-python dependencies; b) it is prone to spurious false-positive build failures and c) its environment is difficult to replicate locally when debugging is required. Alternatively, a Travis based solution may be implemented in the future for deploying the documentation.

3.3. Coming next

While employing CI brings benefits for the developers, it does not add a direct benefit for the user. The road of the software towards the production stage does not end after a successful build or unit test execution [24, 17]. In continuation of the CI step, the rest of the Continuous Delivery (CD) pipeline needs to be implemented providing a fully automated, reliable, repeatable and constantly improving process ended with a ready-to-deploy software package. GitHub features such as the automatic releases or staging areas together with Travis CI (with Docker) and Appveyor will be very useful in building such pipelines. However, since not all parts of the code are automatically tested, the less common use cases will still need to be tested manually. Optimally, these tests should be based on the user documentation, explaining how to interact with the system.

The user could benefit even more if the developers collaboration is extrapolated into the packagers collaboration i.e. the software would become available as high quality packages for the most popular platforms. Currently Sardana and Taurus projects are present in the official Debian [25] repositories, mostly thanks to the effort of a single packager. We believe that all the benefits of the collaborative development could be achieved in the packaging processes as well. Platforms such as Alioth [26] or OBS [27] are some examples on how to implement collaborative packaging. The artifacts produced by the CI step could indeed be distribution packages that could be used in the subsequent steps of the CD pipeline and finally become a software update.

Plenty of other ideas are being evaluated or planned to be implemented in our collaboration. The most interesting ones are the automatic code-style checks for the contributions, the unit test coverage metrics or the voting for the pending issues for the priority assignment. Due to the lack of space they are skipped in this paper.

4. Beyond the technologies

While the technologies and procedures are important, the success in building and coordinating a community of developers is also based on other factors. In our case, the following were determinant:

- Being responsive and encouraging towards users and contributors. Even prioritizing support to external members over those from our own facility (thinking in the longer-term goal of building a strong community).
- Having a "gradual strictness policy": we expect contributors to follow certain conventions, but we try not to scare first-timers by being too strict with their contributions, and rather educating them as they become more involved in the community. In the meanwhile, the integrators accept the burden of adapting the contributions to keep the quality standards.
- Using standard tools and services: when deciding on alternative tools or services, value what the potential contributors are already using and know. Avoid requiring them to join many online services, and of course make sure that all the applications that we recommend are freely available (i.e. Free/OpenSource software).
- Using well-known and documented workflows: just as with the tools, prioritize what the potential contributors already may be familiar with. Even to the point of preferring sub-optimal but well-known and well-documented workflows over perfectly-customized but unique ones that would require contributors to learn one more specific convention and also require ourselves to invest time documenting it.
- Making code modular: this is a good practice in general, but even more for distributed development since it allows less experienced contributors to collaborate without fear of breaking critical parts. It also enables parallel developments.
- Being transparent: use public channels (such as mailing lists, tickets, Enhancement Proposal documents, etc) for all development-related discussions. Even when the participants are on the same site or exchange private emails regularly. In this way, others may join the discussion or at least understand the motivations of previous decisions.
- Document everything. From the APIs (obvious) to the design goals and agreed roadmaps (see previous point). This facilitates outsiders to join the community.
- Ensuring the agreements are being followed: one or more developers get the sheriff role to keep an eye on CI tests status, code static analysis, unattended queues, etc., and remind the rest of the community to act in case of deficiencies.

5. Conclusions

Moving from an in-house development model into a community-driven one was a key for the success of the Sardana and Taurus projects, whose user base grew in the last three years from a few institutions to dozens of laboratories and several companies offering support for them [4]. The contributors community is also starting to grow (see 1) and we expect it to be further expanded with the adoption of more agile contribution workflows in the near future.

The transition required ALBA to invest a lot of effort into tasks whose results were not immediate or which were competing with ALBA's most urgent issues, but which were, nevertheless, deemed necessary on a longer-term perspective.

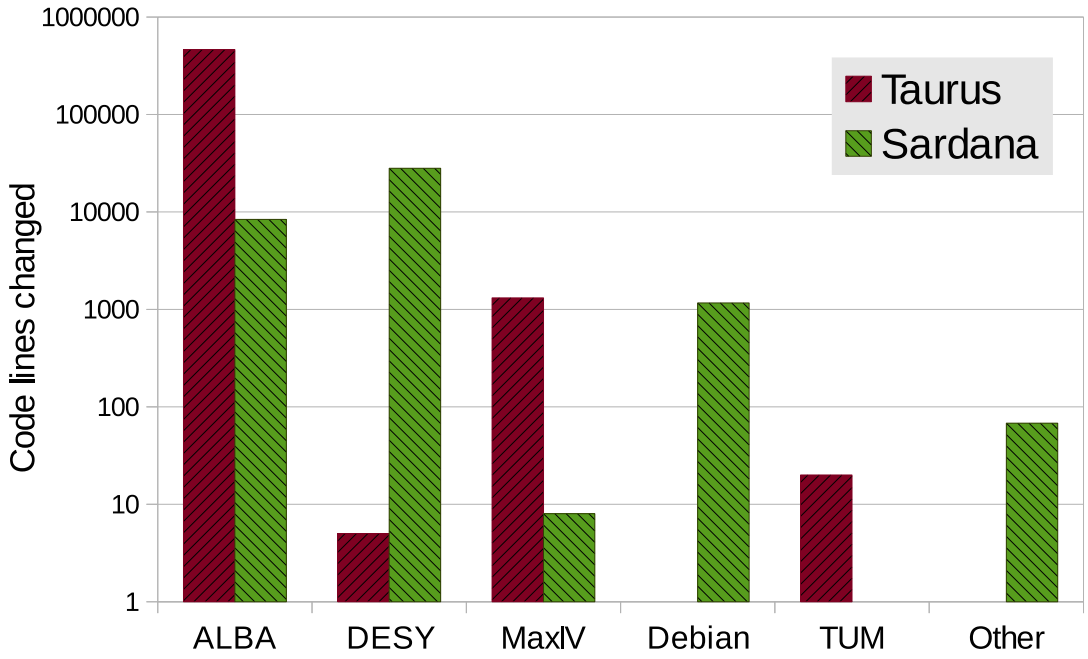

Figure 1. Contributions (measured as number of lines changed) in Taurus and Sardana projects for the period between Jul15 and Jul16 releases, grouped by author's institution

In order to facilitate the incorporation of external developers and to underline the community orientation of the projects, we favored solutions based on external providers (SourceForge, GitHub, Read the Docs, Travis, etc.) instead of on-site infrastructure. We also invested in learning and using standard practices and tools and in engaging, supporting and encouraging contributors.

Of all the decisions regarding tools and workflows, some worked excellently so far, while others already showed some weaknesses and yet some clearly failed but allowed us to learn in the process. We hope that our experiences can also be useful for other projects considering a similar transition.

Acknowledgments

We would like to thank the Sardana and Taurus community members and the ALBA Controls Group and, specially, to T Nunez, J Kotanski and T Kracht (DESY), T Coutinho, V Valls (ESRF), V Michel and A Milan (MaxIV), S Gara (Nexeya), P Goryl and L Zytniak (Solaris), F Picca (Soleil), J Krüger (TUM) and J Andreu, S Blanch, R Homs, J Moldes and D Roldan (ALBA) for their contributions to Sardana and Taurus.

References

[1] E.S. Raymond. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media, 2001.

[2] T Coutinho, G Cuní, D Fernández-Carreiras, J Klora, C Pascual-Izarra, Z Reszela, R Suñé, A Homs, E Taurel, and V Rey. Sardana: The software for building scadas in scientific environments. *ICALEPCS2011, Grenoble, France*, 2011.

[3] Sardana website. `http://www.sardana-controls.org/`.

[4] C Pascual-Izarra, G Cuní, C Falcón-Torres, D Fernández-Carreiras, Z Reszela, and M Rosanes. Effortless creation of control & data acquisition graphical user interfaces with taurus. *ICALEPCS2015, Melbourne, Australia*, 2015.

[5] Taurus website. http://www.taurus-scada.org/.
[6] David Fernández-Carreiras et al. The design of the alba control system. a cost-effective distributed hardware and software architecture. *ICALEPS2011, Grenoble, France*, page 1318, 2011.
[7] SourceForge website. https://sourceforge.net/.
[8] Andrew Götz et al. The tango controls collaboration in 2015. *ICALEPCS2015, Melbourne, Australia*, 2015.
[9] Tango website. http://www.tango-controls.org/.
[10] C. Pascual-Izarra. SEP0 website. https://sourceforge.net/p/sardana/wiki/SEP0/, 2013.
[11] Debian Enhancement Proposal website. http://dep.debian.net/deps/dep0/.
[12] C. Pascual-Izarra. SEP7 website. https://sourceforge.net/p/sardana/wiki/SEP7/, 2013.
[13] Git website. https://git-scm.com/.
[14] GitFlow website. http://nvie.com/posts/a-successful-git-branching-model/.
[15] GitHub website. https://github.com/.
[16] M. Rosanes-Siscart. SEP5 website. https://sourceforge.net/p/sardana/wiki/SEP5/, 2013.
[17] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.
[18] Docker website. https://www.docker.com/.
[19] Jenkins website. https://jenkins.io/.
[20] Travis website. https://travis-ci.org/.
[21] AppVeyor website. https://www.appveyor.com/.
[22] Sphinx website. http://www.sphinx-doc.org/.
[23] Read the Docs website. https://readthedocs.org/.
[24] Z Reszela, G Cuni, CM Falcón Torres, D Fernandez-Carreiras, G Jover-Mañas, C Pascual-Izarra, R Pastor Ortiz, M Rosanes Siscart, and S Rubio-Manrique. Bringing quality in the controls software delivery process. *ICALEPCS2015, Melbourne, Australia*, 2015.
[25] Debian website. https://www.debian.org/.
[26] Alioth website. https://alioth.debian.org/.
[27] openSUSE Build Service website. https://build.opensuse.org/.