The Software Version Control Procedure for the Array Control and Data Acquisition Software of the Cherenkov Telescope Array Observatory

Conforti V.*[a], Gasparyan H.[b], Lopez B.[c], Neise D [c], Oya I [c] for the CTA Observatory [d].
[a]INAF INAF/OAS Bologna, Via Gobetti 93/3, 40129 Bologna, Italy.
[b]Deutsches Elektronen-Synchrotron DESY, Platanenallee 6, 15738 Zeuthen, Germany.
[c]Cherenkov Telescope Array Observatory, Saupfercheckweg 1, 69117 Heidelberg, Germany.
[d]http://www.cta-observatory.org.

ABSTRACT

The Cherenkov Telescope Array Observatory (CTAO) is the next-generation ground-based instrument for gamma-ray astronomy. CTAO will be located at two sites, one in the Northern (La Palma, Spain) and the other in the Southern Hemisphere (Paranal, Chile), with telescopes in three different sizes to cover different energy ranges. The commissioning of the first CTAO Large-Sized Telescope (LST-1) is being finalized at the CTAO-North site. The Array Control and Data Acquisition (ACADA) software is a central element of on-site CTAO operations. ACADA comprises subsystems for central control, the short-term scheduler, monitoring systems, and data handling at rates of GB/s. Consequently, it is a very complex software that requires many developers with different expertise, such as control software, data acquisition, data analysis, scheduling, configuration, and human interfaces. To implement such complex software, ACADA has been broken down into subsystems, which CTAO delegates to expert developer teams around the world through in-kind contributions. All the software is under version control exploiting a dedicated installation of GitLab. We have created at least one repository for each subsystem and a final one for the integration. We have defined the software development and integration procedures so that all phases of the Software Development Life Cycle (SDLC) are supported. Particular attention has been paid to the critical time when a software version is in operation on site and, bug-fixing and new features need to be kept under version control in parallel. The goal is to manage bug fixes without adding new features out of the scope of the release, but at the same time to guarantee the distribution of bug fixes for future releases. This contribution presents our strategy to manage multiple software versions according to the CTAO development plan.

Keywords: Cherenkov Telescope Array Observatory, Software, Control Software, Data Acquisition, version control.

1. INTRODUCTION

The Cherenkov Telescope Array Observatory (CTAO) [1] exploits the Array Control and Data Acquisition (ACADA) [2] software for managing the CTAO operations in the two observing sites: La Palma in Spain and Paranal in Chile. The ACADA system comprises all software responsible for the control and data acquisition of telescopes and the additional devices responsible for array calibration and environment monitoring at each of the CTAO sites. ACADA is also responsible for the efficient execution of pre-scheduled observations and those triggered by science alerts – which allows the CTAO to respond on sub-minute timescales and observe interesting transient phenomena such as gamma-ray bursts [3]. These science alerts can be triggered externally by other scientific installations, or internally within ACADA thanks to a dedicated analysis running in real-time. The ACADA system also provides the user interface for the site operators and astronomers. To implement such complex software, ACADA has been broken down into subsystems, which CTAO delegates to expert developer teams around the world through in-kind contributions. All the software is under version control exploiting a dedicated installation of GitLab. We have defined the software development and integration procedures so that all phases of the Software Development Life Cycle (SDLC) are supported. Particular attention has been paid to the critical time when a software version is in operation on site and, bug-fixing and new features need to be kept under version control in parallel. The goal is to manage bug fixes without adding new features out of the scope of the release, but at the same time to guarantee the distribution of bug fixes for future releases. This contribution presents our strategy to manage multiple software versions according to the CTAO development plan.

*vito.conforti**@inaf.it;**

Software and Cyberinfrastructure for Astronomy VIII, edited by Jorge Ibsen, Gianluca Chiozzi, Proc. of SPIE Vol. 13101, 1310127 · © 2024 SPIE
0277-786X · doi: 10.1117/12.3017802

2. SOFTWARE MANAGEMENT

The ACADA software is a set of subsystems that work together to implement the ACADA requirements and use cases. The ACADA system, with a well-described architecture design [2], will be deployed at each CTAO site as a whole product, and therefore we need a repository which is the main access point to the software. The array-control-and-data-acquisition repository is the main ACADA access point. It includes an ACADA software version (including subsystems as git-submodules), user manuals, integration tests, instructions, and scripts to build, test and run the ACADA software. The ACADA team is composed of several teams in charge of producing subsystems. We have created at least one repository for each subsystem. In addition to ACADA subsystems, the following repositories have been created as well to support the development and test activities:

- common/ to collect the software (also libraries) used by many components.
- icd/ to collect all the interface control documents and "glue code" and application programming interfaces (APIs) between components.
- operation-scripts/ to implement a sequence of tasks foreseen for the operations.
- sde/to define the development environment common for all the sub-systems.

The artifacts above are defined before starting the implementation of a release. Then, the subsystem development teams provide a software version that respects the interfaces governed by the interface control documents, also using the common software agreed upon and provided in the software development environment. For the subsystem development, the feature-branch workflow is encouraged where the default branch (named master/main) is protected, and new features enter the default branch via merge requests (MR). Before merging an MR into the target branch, it is subjected to a set of steps, whose goal is to maintain and improve the quality of ACADA (test pipeline and quality gates pass, code adheres to ACADA quality standards, the new version is sufficiently tested). The formal process of creating a release is done via the GitLab release page of the repository. Here the Coordinator of the subsystem creates a new git tag, release title as well as release notes. A released version is a special commit, pointed to by a git tag. We develop one ACADA release, through several internal use-case-oriented mini-releases. During the mini-release development, more than one use case is likely to be worked in parallel. The integration team prepares a branch for each use case. After the integration is successful, the resulting code is tagged as an ACADA "mini-release". The process is repeated for all use cases associated with the release. The formal process of creating an ACADA release is done via the GitLab release page of the ACADA repository such as for the ACADA subsystems.

3. RELEASE STABILIZATION

In the realm of software development, particularly in complex systems such as the ACADA software, ensuring the stability of a software version is paramount to its successful deployment and operation. The concept of stabilizing a software version represents a critical phase in the software development life cycle, where the software undergoes rigorous testing and refinement to achieve a state of reliability and robustness suitable for deployment in production environments. Stabilization of a software version involves addressing various aspects, mainly bugs but also optimizing performance. This process is essential to mitigate risks associated with deploying unstable software versions that may result in failure during the operations. In case of a new failure created by a newly deployed patch or hot patch, it is always possible to roll back to the previous stable version (which is ready and tested). In general, a bug fix applies to the latest released version. No other functionalities should be provided in a release that fixes a bug. Therefore, the initial point to solve the issue of the subsystems affected by the bug shall be the same version used in the affected release. The stabilization workflow has been defined to stabilize a release while at the same time allowing for proceeding with the development of functionality for the next ACADA release.

3.1 Stabilization workflow during the ACADA verification

When a new ACADA version is under the verification phase, the following procedure, depicted in Figure 1, is applied:

1. The Subsystem teams finalize the development and the resolution of open issues, culminating with the verification of the subsystem. The "commit" used for verification/delivery shall be tagged, following the Semantic Versioning convention (e.g., "3.0.0").

2. Upon successful verification and delivery, subsystem teams can proceed with the development for the next release on the subsystem's "master" branch, as usual.
3. As subsystems get verified and delivered, and to facilitate internal communications (e.g. deployment of test versions), ACADA release candidate tags can be created (e.g., "1.0.0-rc1") on the ACADA "master". Related information shall be included in the tag message, e.g. which subsystems have been finalized, which issues have been fixed, etc.
4. When all subsystems are verified and delivered, the AIV (Assembly Integration and Verification) team creates a branch "release/<version>", (e.g. "release/1.0.0") from the ACADA "master", and tests the release correspondingly. If bugs/issues are detected during this process, subsystem teams are requested to provide corresponding fixes on a branch that starts from the latest delivery (or bug-fix) tag, e.g. from "3.0.0". When delivering the fix, the "commit" used for delivery of the fix shall be tagged again (e.g. "3.0.1"). It is suggested to include notes about the related issues in the tag message.
5. New tags related to bug fixes shall be updated per submodule in the corresponding ACADA "release" branch. When the "release" branch is stable, its tip shall be tagged with the release version (e.g. "1.0.0") and the branch be merged into the "master".

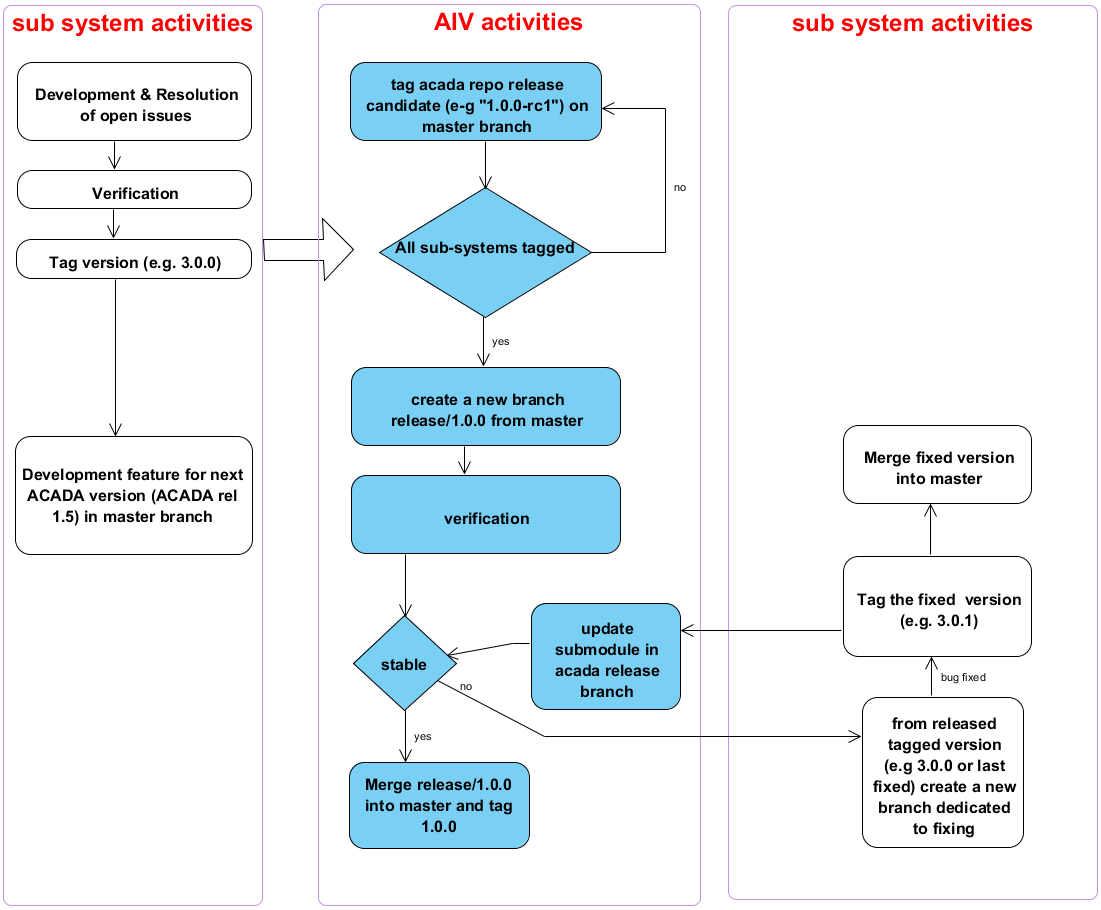

Figure 1. UML activity diagram: stabilization during ACADA verification.

3.2 Stabilization workflow during the ACADA verification

After the release, in case of a bug, a similar process followed during the ACADA verification shall be applied, but in this case, the branch shall be cut from the commit that the latest version tag points to (e.g. "1.0.0"). The branch name shall be hotfix/<version>", e.g. "hotfix/1.0.1". Like the release branch, once finished its tip shall be tagged (e.g. "1.0.1"), and the branch be merged into the "master". Figure 2 shows the UML activity diagram that depicts the process.

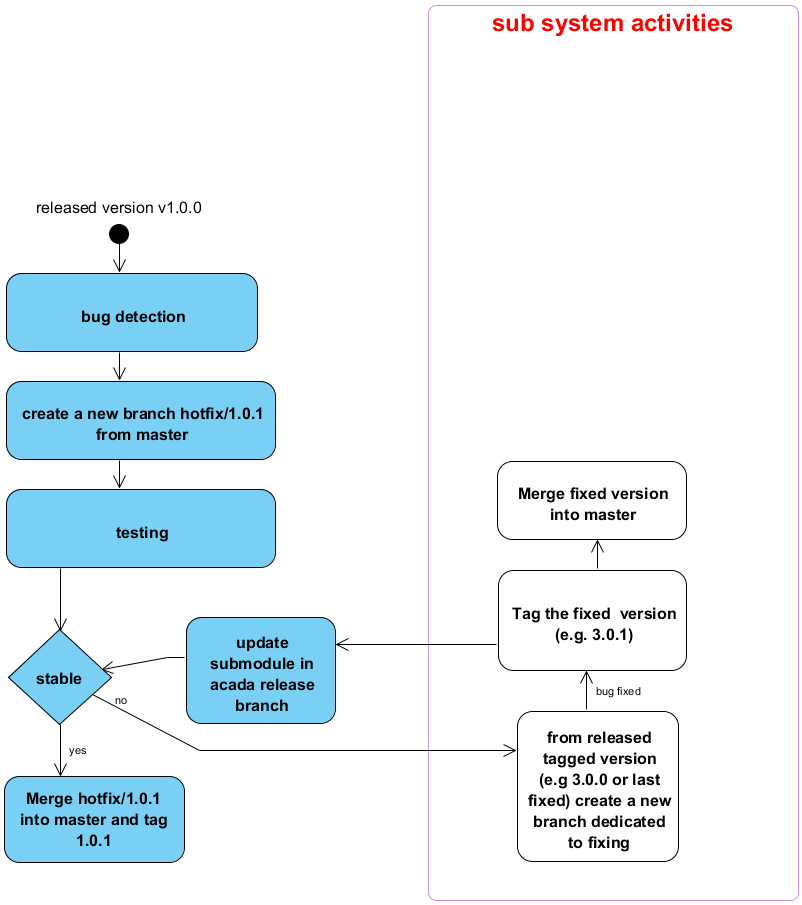

Figure 2. UML activity diagram: stabilization after ACADA release.

4. CONCLUSIONS

The procedure detailed in this paper has been successfully implemented and tested for ACADA Release 1 [4]. The iterative development model selected by the ACADA team, supported by the GitLab functionalities such as software version control, issue management, and release management, facilitating incremental specification additions and effective integration testing through mini-releases. The feature branch workflow enables the subsystems to meet the integration requirements without any look for the other subsystem teams. The stabilization processes allow the developer teams to work in parallel to the bug fixing of past releases while developing features for subsequent releases. This approach enhances transparency, traceability, and collaboration, forming a strong foundation for future ACADA software iterations.

ACKNOWLEDGEMENTS

We acknowledge the support and help of all ACADA team members, as well as CTAO Computing, Systems Engineering, and Project Management department members in establishing the management structure of ACADA. We acknowledge the fundamental commitment to ACADA, with personnel and equipment, by Deutsches Elektronen-Synchrotron (DESY), Institut de Ciències de l'Espai (ICE/CSIC), Istituto Nazionale di Astrofisica (INAF), University of Geneva – Astronomy Department, Laboratoire d'Annecy de Physique des Particules (LAPP), Nicolaus Copernicus Astronomical Center of the Polish Academy of Sciences (CAMK), Max-Planck-Institut für Kernphysik (MPIK) and the University of Potsdam.

We gratefully acknowledge financial support from the agencies and organizations listed here:
https://www.ctao.org/for-scientists/library/acknowledgments/

REFERENCES

[1] Wolfgang Wild and Federico Ferrini "Cherenkov Telescope Array Observatory (CTAO): the world's first and largest ground-based gamma-ray observatory", Proc. SPIE 12182, Ground-based and Airborne Telescopes IX, 121820P (26 August 2022); https://doi.org/10.1117/12.2630255

[2] I. Oya, E. Antolini, M. Füßling, et al, "The Array Control And Data Acquisition System of the Cherenkov Telescope Array" 17th Int. Conf. on Acc. and Large Exp. Physics Control Systems, ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR005

[3] The CTA Consortium, Science with the Cherenkov Telescope Array, World Scientific Publishing Co. Pte. Ltd., 2019. doi: 10.1142/10986

[4] I. Oya, "The first release of the Cherenkov Telescope Array Observatory Array Control and Data Acquisition software" , SPIE Telescope and Instrumentation 2024.