Common software lifecycle management in external projects

C. Duma[1], A. Costantini[1], D. Michelotto[1], P. Orviz[2] and D. Salomoni[1]

[1]INFN Division CNAF, Bologna, Italy
[2]IFCA, Consejo Superior de Investigaciones Cientificas-CSIC, Santander, Spain

E-mail: `ds@cnaf.infn.it`

Abstract.
This paper describes the common procedure defined and adopted in the field of Software Lifecycle Management and Continuous Integration and Delivery to manage the new releases, as a first step to ensure the quality of the provided solutions, services and components, while strengthening the collaboration between developers and operations teams among different external projects. In particular, the paper analyses the common software lifecycle management procedure developed during the INDIO-DataCloud project and recently improved and adopted in two EC funded projects: eXtreme DataCloud and DEEP Hybrid DataCloud.

1. Introduction

The eXtreme-DataCloud (XDC) [1] and DEEP-HybridDataCloud (DEEP-HDC) [2] projects are aimed at addressing requirements from a wide range of User Communities belonging to several disciplines and test the developed software solutions against the real life use cases. The software solutions carried out by the both projects are released as Open Source and are based on already existing components (TRL8+) that the projects will enrich with new functionalities and plugins.

The use of standards and protocols widely available on the state-of-the-art distributed computing ecosystems may be not enough to guarantee that the released components can be easily plugged into the European e-Infrastructures and in general on cloud based computing environments and the definition and implementation of the entire Software Lifecycle Management process becames mandatory in such projects.

As the software components envisaged by both projects have a history of development in previous successful European projects (such as the INDIGO-DataCloud [3] project) implementing different types of modern software development techniques, the natural choice was to complement the previous, individual, Continuous Development and Integration services with a Continuous Testing, Deployment and Monitoring as part of a DevOps approach:

- Continuous Testing - the activity of continuously testing the developed software in order to identify issues in the early phases of the development. For Continuous testing, automation tools will be used. These tools enable the QA's for testing multiple code-bases and in parallel, to ensure that there are no flaws in the functionality. In this activity the use of Docker containers for simulating testing environments on the fly, is also a preferred choice. Once the code is tested, it is continuously integrated with the existing code.

- Continuous Deployment - the activity of continuously updating production environment once new code is made available. Here we ensure that the code is correctly deployed on all the servers. If there is any addition of functionality or a new feature is introduced, then one should be ready to add resources according to needs. Therefore, it is also the responsibility of the SysAdmin to scale up the servers. Since the new code is deployed on a continuous basis, automation tools play an important role for executing tasks quickly and frequently. Puppet, Chef, SaltStack and Ansible are some popular tools that could be used at this step. This activity represents the Configuration Management - the process of standardising the resources configurations and enforcing their state across infrastructures in an automated manner. The extensive use of containerisation techniques will provide an entire runtime environment: application/service, all its dependencies, libraries and binaries, and configuration files needed to run it, bundled in one package - container. T3.1 will also manage the scalability testing, being able to manage the configurations and do the deployments of any number of nodes automatically.
- Continuous Monitoring - very crucial activity in the DevOps model of managing software lifecycle, which is aimed at improving the quality of the software by monitoring its performance. This practice involves the participation of the Operations team who will monitor the users' activity to discover bugs or improper behavior of the system. This can also be achieved by making use of dedicated monitoring tools, which will continuously monitor the application performance and highlight issues. Some popular tools useful in this step are Nagios [4], NewRelic [5] and Sensu [6]. These tools help to monitor the health of the system proactively and improve productivity and increase the reliability of the systems, reducing IT support costs. Any major issues found could be reported to the Development team to be fixed in the continuous development phase.

These DevOps activities are carried out on loop continuously until the desired product quality is achieved. Automation will play a central role in all the activities in order to achieve a complete release automation, moving the software from the developers through build and quality assurance checks, to deployment into integration testbeds and finally to production sites part of the Pilot Infrastructures. In the following sections, an overview of the recentli defined best practices that have been adopted in both XDC and DEEP-XDC projects for the Software Lifecycle Management and Continuous Integration and Delivery are presented and described.

2. Software Quality Assurance and Control

Software Quality Assurance (SQA) covers the set of software engineering processes that foster the quality and reliability in the software produced. The activities involved in this task are mainly focused on:

- Defining and maintaining a common SQA procedure to guide the software development efforts throughout its life cycle.
- Formulating a representative set of metrics for the software quality control to follow up on the behavior of the software produced, aiming to detect and fix early deviations in the software produced.
- Enabling a continuous integration process, eventually complemented by a continuous delivery scenario, promoting the automation adoption for the testing, building, deployment and release activities.

In order to define the SQA process, the specific context of the software developed in the project has to be taken into account. The following particularities characterize the corresponding development teams:

- Heterogeneous developer profiles: different backgrounds and different degrees of expertise.

- Geographically distributed.
- Different home institutes which implies different cultures, different development technologies, process and methods.
- High turnover due to the limited duration of the projects where the grid software has been developed so far.
- More focus on development activities, with limited resources, if any, available for quality assurance activities.

The Quality Assurance process has to take all above described factors into account to define the Software Quality Assurance Plan (SQAP). A set of "QA Policies" have also to be defined to guide the development teams towards uniform practices and processes. These QA Policies define the main activities of the software lifecycle, such as releasing, tracking, packaging and documenting the software carried out by the project. This is done in collaboration with development teams, making sure they are flexible enough to co-exist as much as possible with current development methods. The SQA activities have to be monitored and controlled to track their evolution and put in place corrective countermeasures in case of deviations.

Moreover, a quality model have to be defined to help in evaluating the software products and process quality. It helps to set quality goals for software products and processes. The Quality Model has to follow the ISO/IEC 25010:2011 "Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models" [7] to identify a set of characteristics (criteria) that need to be present in software products and processes to be able to meet the quality requirements.

Those SQA criteria **??** have the goal to

- Enhance the visibility, accessibility and distribution of the produced source code through the alignment with to the Open Source Definition [9].
- Promote code style standards to deliver good quality source code emphasizing its readability and reusability.
- Improve the quality and reliability of software by covering different testing methods at development and pre-production stages.
- Propose a change-based driven scenario where all new updates in the source code are continuously evaluated by the automated execution of the relevant tests.
- Adopt an agile approach to effectively produce timely and audience-specific documentation.
- Lower the barriers of software adoption by delivering quality documentation and the utilization of automated deployment solutions.
- Encourage secure coding practices and security static analysis at the development phase while providing recommendations on external security assessment.

3. Software Maintenance and Support

Regarding the software maintenance and support area of the software lifecycle management, the main objectives that should be and described in the Maintenance plan are:

- To increase the quality levels of the software by contributing to the implementation and automation of the Quality Assurance (QA) and Control procedures defined by the project.
- To boost the software delivery process, relying on automation.
- To emphasize the communication and feedback with/from end users, in order to guarantee adequate requirements gathering and support.
- To guarantee the stability of services already deployed in production and the increase of their readiness levels, where needed.

Moreover the common practices deal with the definition of those processes and procedures related to the software maintenance and support and their continuous execution:

- Software Maintenance - regarding software preparation & transition from the developers to production repositories and final users.
- Problem Management - providing the analysis & documentation of problems.
- Change Management - control code, configuration changes, retirement calendars.
- Coordination the provisioning of adequate support to released software.
- Responsible for the release management and coordination and the maintenance of the artifacts repositories, defining policies and release cycles.

The plan regarding the software maintenance and support management have to follow the guidelines of the ISO/IEC 14764:2006 standard [10], and includes a set of organizational and administrative roles to handle maintenance implementation, change management and validation, software release, migration and retirement, support and helpdesk activities. Component releases are classified in major, minor, revision and emergency, based on the impact of the changes on the component interface and behavior. Requests for Change (RfC) are managed adopting a priority-driven approach, so that the risk of compromising the stability of the software deployed in a production environment is minimized. The User Support activity deals, instead, with the coordination of the support to the users that make use of the software components (developed within the project activities) and included in the main project software distributions.

4. Services for continuous integration and SQA

To support the Software Quality Assurance, the Continuous Integration and the software release and maintenance activities, a set of tools and services are needed. Usually, those tools and services are provided by using publicly available cloud services due to the following reasons:

- Higher public visibility and in line with project objectives for open source software,
- Provides a path to further development, support and exploitation beyond the end of the project,
- Smaller effort needed inside the project to operate and manage those services.

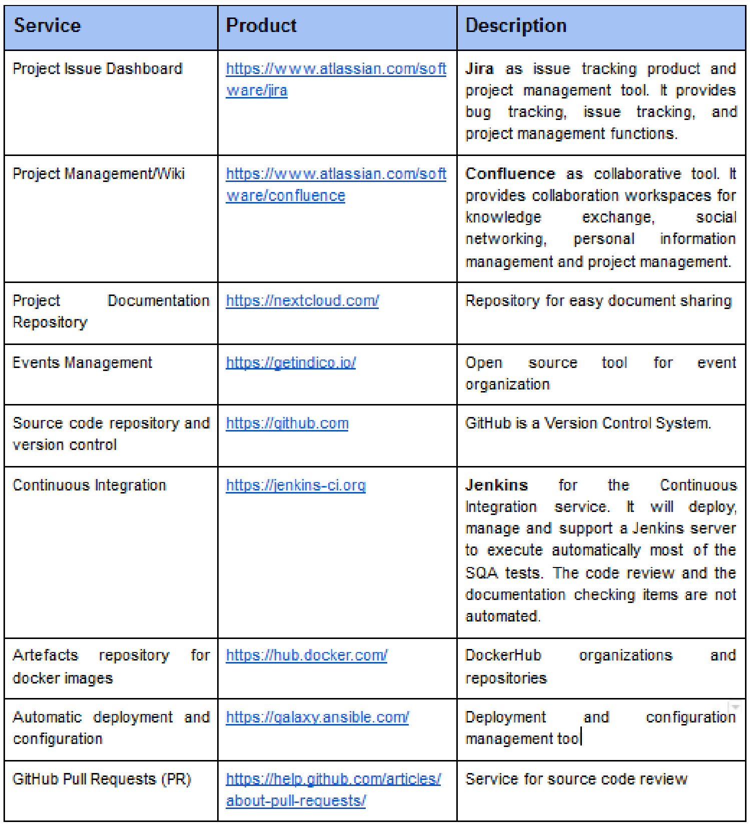
The list of services needed is given in Table 1 with a small description for each service and the related Web link.

5. Key Performance Indicators

Defining appropriate KPIs for maintenance, release and support activities, and monitor them during the project lifetime may help in highlight the project achievements and put in place the appropriate corrective actions in case of deviations. In principle, the KPIs should address the following impact areas and reflect the related goal:

- Prepare data and computing e-Infrastructures to absorb the needs of communities that push the envelope in terms of data and intensive computing
  - Goal: Extending the quality & quantity of services provided by e-infrastructures
- Promote new research possibilities in Europe
  - Goal: Increasing the capacity for innovation and production of new knowledge

Table 1: Tools and services to support DevOps.



Service	Product	Description
Project Issue Dashboard	https://www.atlassian.com/software/jira	**Jira** as issue tracking product and project management tool. It provides bug tracking, issue tracking, and project management functions.
Project Management/Wiki	https://www.atlassian.com/software/confluence	**Confluence** as collaborative tool. It provides collaboration workspaces for knowledge exchange, social networking, personal information management and project management.
Project Documentation Repository	https://nextcloud.com/	Repository for easy document sharing
Events Management	https://getindico.io/	Open source tool for event organization
Source code repository and version control	https://github.com	GitHub is a Version Control System.
Continuous Integration	https://jenkins-ci.org	**Jenkins** for the Continuous Integration service. It will deploy, manage and support a Jenkins server to execute automatically most of the SQA tests. The code review and the documentation checking items are not automated.
Artefacts repository for docker images	https://hub.docker.com/	DockerHub organizations and repositories
Automatic deployment and configuration	https://galaxy.ansible.com/	Deployment and configuration management tool
GitHub Pull Requests (PR)	https://help.github.com/articles/about-pull-requests/	Service for source code review

6. Conclusions

The paper describes the common procedures to be applied in the field of software lifecycle management, aimed at managing the new releases and ensure the quality of the provided solutions, services and components provided by the project. In particular, the paper described the best practices to adopt in order to i) foster the quality and reliability of the software produced, ii) to define the processes and procedures regarding the software maintenance and support, iii) identify the services needed to support the Software Quality Assurance, the Continuous Integration and the software release and maintenance and iv) define appropriate KPIs to monitor the project achievements.

The experience gathered throughout this activity with regards to

The adoption of different DevOps practices is becaming mandatory for software development projects, the experience gathered throughout this activity can be also applicable to the development and distribution of software products coming, for example, from the user communities and other software product activities.

Acknowledgments

DEEP-HybridDataCloud has been funded by the European Commission H2020 research and innovation program under grant agreement RIA 777435. eXtreme DataCloud has been funded by the European Commission H2020 research and innovation program under grant agreement RIA 777367.

7. References

[1] Web site: www.extreme-datacloud.eu

[2] Web site: www.deep-hybrid-datacloud.eu
[3] Web site: www.indigo-datacloud.eu
[4] Web site: https://www.nagios.org
[5] Web site: https://newrelic.com
[6] Web site: https://sensu.io
[7] ISO/IEC 25010:2011, "Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models": https://www.iso.org/standard/35733.html
[8] A set of Common Software Quality Assurance Baseline Criteria for Research Projects, http://digital.csic.es/bitstream/10261/160086/4/CommonSQA-v2.pdf
[9] The Open Source Definition, https://opensource.org/osd
[10] ISO/IEC 14764:2006 standard, https://www.iso.org/standard/39064.html