Track 2 Lightning Talk: Software Development Best Practices at the Netherlands eScience Center

Jason Maassen Niels Drost Willem van Hage Rob van Nieuwpoort
Netherlands eScience Center, Amsterdam, The Netherlands
Email: j.maassen,n.drost,w.vanhage,r.vannieuwpoort@esciencecenter.nl

***Abstract*—In this talk we present the software development best practices that the Netherlands eScience Center has developed during the 90+ projects it has participated in, in the 5 years of its existence.**

I. Introduction

The Netherlands eScience Center (NLeSC) [1] is a Dutch funding agency and scientific software development expertise center that supplies specialized manpower (eScience Research Engineers) along with research funding for innovative eScience projects to all of research in the Netherlands.

The main goal of the NLeSC is to apply technological innovation developed in computer and data science research to enhance other research fields (ranging from climate science to archeology). Software sustainability plays an important role in achieving this goal. We are only successful if the software created in our projects has a lasting impact, and to do so the software needs to be (at least) reliable, reusable, maintainable and well documented.

One of the problems faced by the NLeSC is the sheer diversity of our projects. Project topics range from fields that are traditionally strong in adapting computer and data science, such as climate science, astronomy or high-energy physics, to fields only just discovering the possibilities, such as archeology, ecology or law. As a result of this diversity, there is no one-size-fits-all solution to software development and sustainability.

We do, however, see a set of best-practices which can be applied to most software development we perform in our projects. We have gathered these in a guide [2], which we take as a staring point in all of our projects. This guide is a living document, created by and for the eScience Research Engineers working in our center and the project partners we cooperate with. It slowly changes with our increasing experience and the insights we gain from our projects, and due to the increasing number of software development tools available on line. Like for most of our software, we use GitHub [3] to cooperatively develop the guide.

In the remainder of this document we will highlight a few of the best practices described in our guide. Although some of these may initially appear to be stating the obvious, we often see that explicitly discussing them helps our project partners understand why we work in a certain way, and it often stimulates them to think about software development issues they originally have not given much thought. We would like to stress that we do not claim our guide to be a one-size-fits-all solution. It works well for us, however, and we feel other institutes may learn from our experiences.

II. Best practices

In this section we highlight a few of the software development best practices we apply in our projects.

A. Use and create open source software whenever possible.

Open source is important part of reproducible research, next to open access and open data. Although the latter two receive a lot of attention from funding agencies and publishers, we feel open source is just as important, as it is often the scientific instrument used to produce the results. When creating open source software, it needs to be findable, accessible and usable. Claiming anyone may have a copy of your research software after they send you an email is not enough to make it open source. It need to be easy to find on-line, anyone should be able to download it, it needs proper instructions on how to get it to run, and most importantly, contain a proper open source license. At the NLeSC, we use GitHub [4] for our projects. Practically all of our software is in public repositories and uses the Apache-2 license [5]. There are several alternatives to both.

B. Use version control from the start.

Although the complexity of version control systems can be daunting to less experienced developers, they have clear benefits when developing software in a team. Although it is possible to host version management systems on the institutes private servers, it significantly reduces the findability and accessibility of the code. We prefer to use public services instead, such as GitHub [4], BitBucket [6], etc.

C. Branching model, coding style, contributor guide, code of conduct, ...

A branching model is an agreement on how (and how often) to branch and merge code in a version control system. Explicitly picking a branching model helps to ensure that multiple developers can smoothly cooperate on a single piece of software. We prefer to use the GitHub Flow model [7], as it is a lightweight approach also suitable for smaller teams.

Picking a coding style improves the readability and maintainability of the code, as it ensures all developers use the same style. A code style can easily be configured via editor plugins such as EditorConfig [8].

A contributor guide explains how developers can contribute to the code, while the code of conduct sets the rules on how developers should behave when communicating with each other. Although these seem to be minor details, thinking about them at the start of a project and clearly communicating them can save a lot of discussion afterwards.

D. Testing, code quality tools, code reviews.

Unit and integration testing and code reviews are well known ways to improve software quality. If possible, we try to include code reviews into your version management. For testing we use several of the cloud based continuous integration services available. These integrate into version management, compiling and running test automatically whenever code is committed. At NLeSC we mainly use Travis CI [9], AppVeyor [10], and Scrutinizer CI [11]. Each offers different features, and a single software project often needs to use a combination of them to fully cover all expected user environments. For example, we often use a combination of Travis CI to test in Linux and MacOS environments, and AppVeyor to test in Windows.

When we need to test our software against servers running a specific stack of services, such as a databases, webservers, or cluster resource managers (i.e., TORQUE [12] or Slurm [13]), we use Docker [14] to instantiate these services in the continuous integration environment. This also provides an easy way to test against different versions or configurations of these services by simply creating multiple docker containers, one for each version.

We use services such as Codecov [15] analyze the coverage of the tests, and code quality tools such as Codacy [16] or SonarQube [17] to detect well known bugs, vulnerabilities and code smells.

Many of the tools mentioned above can produce badges which show their results, e.g., if the software is building correctly, what the test coverage is, a grade for the code quality, etc. We publishing these badges on the GitHub pages of the software to show that we take testing and code quality seriously.

E. Software releases, package managers, citable software.

Although GitHub makes it very easy to create software releases, it is not necessarily the best way to reach users. In our experience, only very few users directly download software releases (or source for that matter) from GitHub. Instead, they rely on (often programming language specific) package managers and repositories such as PyPi [18], conda [19], npm [20], The Central Repository (Maven) [21], or JCenter [22]. Therefore, we always make our software releases available in these repositories.

To ensure our software is citable, we create a DOI for each release using Zenodo [23]. This requires very little effort due to the integration of Zenodo into GitHub. A DOI is automatically generated at the moment a release is created, and a copy of the release is stored at Zenodo. As with the code quality tools, a badge can be used to publish the DOI of the latest release on the GitHub page.

References

[1] Netherlands eScience Center, “The Netherlands eScience Center website.” [Online]. Available: https://www.esciencecenter.nl

[2] ——, “Netherlands eScience Center Guide.” [Online]. Available: https://www.gitbook.com/download/pdf/book/nlesc/guide

[3] ——, “Netherlands eScience Center Guide Repository.” [Online]. Available: https://github.com/nlesc/guide

[4] GitHub, “GitHub Source Code Hosting.” [Online]. Available: http://github.com

[5] The Apache Software Foundation, “The Apache 2 License.” [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0

[6] Atlassian, “Github Source Code Hosting.” [Online]. Available: https://bitbucket.org

[7] GitHub, “GitHub Flow branch model.” [Online]. Available: https://guides.github.com/introduction/flow/

[8] EditorConfig, “The EditorConfig file format and plugin collection.” [Online]. Available: http://editorconfig.org/

[9] Travis CI, “Travis Continuous Integration Service.” [Online]. Available: https://travis-ci.org

[10] AppVeyor Systems Inc., “AppVeyor Continuous Integration Service.” [Online]. Available: https://www.appveyor.com

[11] scrutinizer-ci, “Scrutinizer Continuous Integration Service.” [Online]. Available: https://scrutinizer-ci.com/

[12] Adaptive Computing, “TORQUE Resource Manager.” [Online]. Available: http://www.adaptivecomputing.com/products/open-source/torque/

[13] SchedMD, “Slurm Workload Manager.” [Online]. Available: https://slurm.schedmd.com/

[14] Docker Inc., “Docker Software Container Platform.” [Online]. Available: https://www.docker.com

[15] Codecov, “Code Coverage Reporting Service.” [Online]. Available: https://codecov.io

[16] Codacy, “Automated Code Reviews and Code Analytics.” [Online]. Available: https://www.codacy.com

[17] SonarQube, “SonarQube Software Quality Management Platform.” [Online]. Available: https://www.sonarqube.org

[18] The Python Community, “PyPI, the Python Package Index.” [Online]. Available: https://pypi.python.org/pypi

[19] Continuum Analytics., “Conda: Package, dependency and environment management for any language.” [Online]. Available: https://conda.io/docs/

[20] npm Inc., “The package manager for JavaScript.” [Online]. Available: https://www.npmjs.com/

[21] Sonartype Inc., “The Central Repository for Apache Maven packages.” [Online]. Available: http://search.maven.org

[22] JFrog Bintray, “JCenter repository for Apache Maven packages.” [Online]. Available: https://bintray.com/bintray/jcenter

[23] OpenAIRE, “Zenodo research data repository.” [Online]. Available: https://zenodo.org