Electronic Communications of the EASST
Volume 83 Year 2025

deRSE24 - Selected Contributions of the 4th Conference for Research Software Engineering in Germany

Edited by: Jan Bernoth, Florian Goth, Anna-Lena Lamprecht and Jan Linxweiler

Collaborative software development to disentangle counterproductive incentives

Fynn Freyer, Paul Wolk, Marie Bittiehn, Berit Haldemann, Marie Lataretu, Stephan Fuchs, Piotr Wojciech Dabrowski

DOI: 10.14279/eceasst.v83.2612

Collaborative software development to disentangle counterproductive incentives

Fynn Freyer[1*], Paul Wolk[2],[*] Marie Bittiehn[1*], Berit Haldemann[2], Marie Lataretu[2], Stephan Fuchs[2+], Piotr Wojciech Dabrowski[1+]

School of Computing, Communication and Business (Faculty 4), HTW Berlin University of Applied Sciences[1]
Genome Competence Center (MF1), Robert Koch Institute[2]
* These authors contributed equally
+ These authors also contributed equally

Abstract:
Background: We propose a collaborative approach to address the conflicting incentives between software development (generating high-quality code) and research (rapidly generating scientifically interpretable results).
Approach: A data analysis pipeline has been developed driven by current research needs and following software engineering best practices in a pilot collaboration between scientists from the Genome Competence Center at the Robert Koch Institute (RKI) and the computer science programs at HTW University of Applied Sciences. The challenges and benefits of such collaborations are discussed in the context of increasing the quality of research software.
Conclusion: This approach has led to the development of a new quality control pipeline following software engineering best practices: Coding conventions, an extensive test suite, version control with branching guidelines and gated commits, and automated reproducible builds. Our experiences highlight the benefits of such collaborations, and we hope to encourage others to follow similar approaches in order to facilitate the development of high-quality research software. The newly developed QC pipeline is available online on Azure [qcu] and v.1.1.3 is available on zenodo [FWH+24].

Keywords: Collaborative development, research software engineering, bioinformatics, computer science education, synergy, quality control pipeline

1 Introduction

As the world in general and research specifically largely depends on software, that software should be developed in a sustainable manner, meaning that it "continues to meet its purpose over time, which includes that the software adds new capabilities as needed by its users, responds to bugs and other problems that are discovered, and is ported to work with new versions of the underlying layers, including software as well as new hardware" [KM21]. However, pure software development and research are naturally driven by different incentives. While the main aim of software development should be the generation of a reliable, easily maintainable and sustainable product, the daily life of a researcher is dominated by the quest for accurate, reproducible state-of-the-art analysis results and by publication deadlines. This discrepancy is one of the driving forces behind the Research Software Engineering (RSE) community and has been described in detail in [ABD+21]. While sustainable long-term solutions are presented in that paper, their implementation requires changes to such complex systems as funding programmes or the evaluation of the value of scientific contributions. As such changes take a long time to implement, there is still a need for approaches that can help increase the quality of research software in the short to medium term.

The workplace of the Research Software Engineer is exactly in this area of conflict: Developing sustainable software for rapidly moving and publication-focused research. Depending on the institution, this is usually achieved by either having RSEs directly integrated in the research groups to maximize the expertise in the field where the software will be applied, or by creating central RSE groups that provide services to the entire institution and allow synergies from many RSEs working in a shared environment [CW20]. Additionally, building communities where RSEs share their knowledge can blur the lines between these approaches and combine the benefits of both [SKM+18].

As an extension of a central RSE group, that group can provide services to collaborating partners beyond its own institution. Successful examples include the Innovative Software and Data Analysis group at the University of Illinois or the Center for Research Computing Software Development Group at the University of Notre Dame [KMRH19].

We present a possible path towards seeding such an RSE group that is focused on external collaborations in order to isolate the software engineering process from the volatility of day-to-day research work. To this end, two different entities driven by different incentives work closely together: The team at the Genome Competence Center of the Robert Koch Institute (RKI) and the applied computer science bachelor's and master's study programs of the HTW Berlin University of Applied Sciences.

The Genome Competence Center provides consulting and analyses to other research units within the RKI. The bioinformaticians working at the unit are fully qualified and experienced computer scientists with the knowledge and skills necessary for the development of high-quality software. However, the primary focus is on the reliable generation, processing, and visualization of analysis results, typically through prototypical workflows. Due to capacity constraints, best practices from software engineering often receive insufficient attention.

On the other hand, the HTW has access to a large number of computer science students with a high interest in software development, such as Applied Computer Science, Computer Engineering, Computer Science in Culture and Health or Health Electronics. As such, they are a perfect resource for developing research software that is focused on reproducibility, scalability and sustainability. However, there are two challenges to overcome when working with computer science students who want to develop research software: Firstly, access to realistic questions and data is hard to come by, and developing software that solves a self-posed problem using simulated data is not very satisfying. Secondly, students are only available during the short time frames defined by the duration of their projects.

Given that those challenges and opportunities show high potential for synergy, RKI and HTW have established a cooperation for research software engineering. In short, the daily challenges and requirements concerning analysis software are discussed in semi-regular joint meetings.

Work packages and research questions (such as the comparison of different algorithms) are jointly defined and prioritized. Then, those are given out to students at the HTW in the form of projects or final theses. The results are continuously combined and evaluated at the RKI, and flow back into the work package definitions. In order to create a sustainable and continuous environment in which those work packages can be integrated, a single position is co-financed by the RKI and the HTW to supervise and coordinate the entire process.

In this way, a new bioinformatics pipeline to evaluate the quality of Illumina sequencing data (QCurchin) was developed that combines state-of-the-art bioinformatics methods with essential principles of good software engineering. QCurchin is currently productively used at the RKI. We hope that our experiences, both positive and negative, will encourage others to enter and profit from similar synergetic relationships. The details of the approach and the resulting quality control pipeline are described on the following pages, in the hope that this will motivate and facilitate other similar cooperations.

2 Case description

The RKI is the national Public Health Institute in Germany. As such, research on and surveillance of infectious diseases belong to its main objectives. Next Generation Sequencing (NGS), which was initially introduced at the RKI in 2006 in the unit focused on highly pathogenic viruses (ZBS1), is playing an increasingly important role in this context. Given the growing importance and application of NGS for public health-related tasks, a central sequencing core facility was established at the RKI in 2017. In 2022, both sequencing and bioinformatics core facilities were merged to form the Genome Competence Center, providing integrated sequencing data and analyses.Currently, tens of thousands of samples are processed annually using various technologies in the fields of genome, expression, metagenome, and single-cell analyses. Since analysing each of these datasets manually would require a prohibitive amount of hands-on time, the bioinformatics team depends heavily on automated workflows to perform often-recurring types of analyses in routine settings.

One such type of analysis that needs to be performed on virtually every dataset of sequencing raw data is quality control (QC) [EGFF16]. The task of such a pipeline is to run several tools, as outlined below, to assess, improve and report on the quality of the raw data. The assessment and automated removal of known types of artifacts ensures that follow-up analyses work with high-quality data and can create reliable results. Also, in case of unexpected results in the follow-up analyses, the quality report generated by the pipeline can be used to quickly identify potential quality-related causes for those unexpected results. The quality report is also used by the sequencing core facility in order to monitor the overall sequencing quality level and as an additional factor in monitoring the health of the sequencing machines. This means that a QC pipeline used at the service unit has to meet especially high standards in regard to robustness and maintainability. Thus, a QC pipeline was chosen as the use case for the collaboration described here.

The requirements for the pipeline were defined based on the equipment and IT infrastructure available within the institute and previous experience. In brief, they are as follows:

- Analyze data from Illumina sequencers

- Extract run metadata from paths and filenames based on flexible naming patterns
- Perform read trimming (see e.g. [CKHW14] and [Che23])
- Perform (mapping-based and/or kmer-based) taxonomic assignment of reads to facilitate identification of contamination or sample swaps (see e.g. [MWX+24])
- Generate customizeable end-user and internal reports
- Provide both sensible defaults and extensive parametrization
- Run both on HPC environments and locally

While these requirements are specifically geared toward the situation at the RKI, they are also somewhat typical for core facilities offering NGS data analysis in general.

3 Technical solution

In the description of the technologies and processes, we focus on those aspects that we believe to be transferable to other similar projects since this is a case study rather than a software note. An extensive description of the pipeline itself, QCUrchin, is present in the repository under https://dev.azure.com/RKI-HTW/NGS-QC/_git/QCurchin.

In order to make complex workflows consisting of many tools manageable, the best practice is to use a workflow management system [WWG21]. Many such systems are available, and the comparison of their features is beyond the scope of this work. For this specific case, we have decided to use Nextflow [DCF+17]. To ensure reproducibility and portability across different compute environments, we use apptainer [KCB+21]: Every Nextflow process defines an apptainer image that the process is executed in. This also makes deployment easier, as the only requirements are Nextflow and Apptainer/Singularity.

Initially, we had created a single apptainer image containing all required tools: FastP [Che23], bowtie2 [LS12], Kraken2 [WLL19], Bracken [LBTS17], rmarkdown [rma24] for reporting, and Python environments for scripts used for intermediate steps. However, while very convenient for development and testing, this caused two challenges: (i) Changing the version of a single tool could have effects on the functioning of other tools (e.g. if the version of a library that several tools depend on changed), and (ii) it negatively impacted the performance when running on an HPC cluster according to the cluster admin, since the entire container had to be copied to each node even if the job on that node only required one of the packaged tools. This performance issue might just be an artifact of the RKI cluster, since the use of single- vs. multi-container deployment should not significantly affect performance [LG20]. Also, nf-core [EPF+19], which is a large community-based effort that provides reference nextflow pipeline implementations, follows this approach of a separate container for each process. Accordingly, the final version of the pipeline uses a separate apptainer image for each process that contains only the tools specifically required by that process.

To make reproducing results as simple as possible, QCUrchin can be started both using a configuration file (with a default configuration file present in the environment) and command line parameters that override settings from the configuration file. It then generates a new configuration file containing all options used during the run along with the analysis results, which allows re-running the same analysis using that configuration file.

4 Organisation and challenges

While the technical approach described above more or less constitutes bread-and-butter pipeline development, the organization of such a project posed several challenges that we believe others may be able to learn from.

First and foremost, requirements change over time in research software development just as they do in every other software project. Thus, an agile approach with regular meetings between the project partners is essential for success (we used a scrum-like approach adapted to the team size). However, this process needs to be mindful of both the very limited time of the scientists - which is one of the main motivations behind this whole approach - and the relatively low level of experience with the requirements of scientists that most typical software developers have. It is also vital to allocate sufficient time to allow both parties to understand each other's language. Often, the same words are used in different communities to describe different things. It is useful to allow each side to rephrase and repeat their understanding of problems, solutions and progress in their own words to reduce misunderstandings caused by those different languages.

In our experience, it is thus important to first identify a minimum viable product that will cover the most frequent use cases of the scientists and can be rapidly deployed as a first-release candidate. This allows the scientists to quickly see a benefit, as they have something that, while not perfect, makes their work a bit easier. It also motivates them to actually use early release candidates - as opposed to forcing them to use valuable time that they would rather apply towards their research for testing the software - and to provide feedback on misunderstandings and necessary improvements at an early stage in development. It is hard to overstate the importance of such a feedback loop that should feel productive and not like a burden for both sides.

A second major challenge in this model of collaboration is posed by the availability of resources. In most cases, the financial resources will not be present to outright employ software engineers who will do all the work. However, students from study programmes that involve computer science education can help in solving this problem by contributing source code in the course of internships or student projects. This is a mutually beneficial arrangement, as it not only helps the group hosting the students develop or maintain analysis software. It is also both financially attractive and useful for the involved students, as they gain insights into the inner workings of a real project and the satisfaction of generating code that will actually run in production. The challenges this poses lie in the short duration of the individual student projects, the associated high personnel turnover, and the varying quality of code produced by different students. Especially when involving students in the form of projects that run in parallel to the semester, we have observed that, as described by Katz et al. [KMRH19], students need to be consistently available to work on a project for at least about one whole day per week in order to generate a meaningful net contribution to the project.

The use of a workflow management system such as Nextflow makes coping with those challenges somewhat easier. It allows the creation of subtasks, such as the development and benchmarking of a single process, the restructuring of a sub-workflow, or the creation of a new report template that can realistically be completed within a student project. Those features can also be developed within separate branches and only merged into the production branch if they are of sufficiently high quality.

However, the high turnover still makes knowledge transfer problematic. As such, a special focus needs to be put on high-quality, consistent documentation—especially concerning the overall structure of the project and the steps necessary to start development—as well as consistent code style and structure. Additionally, a robust test suite, including both unit tests for the individual processes and integration tests for the entire pipeline, is necessary to avoid regressions. In QCUrchin, we employ pytest [KOP+04] for unit tests and pytest-workflow [pyt] for end-to-end tests. A reduced test suite is used for gated commits to ensure that the main branch always contains functioning code, and the full test suite is run as part of the automated builds. To ensure that the artefacts behave consistently with the test suite, the singularity container that QCUrchin is packaged into is the same container as the one that the tests run in - with the only exception that the large test datasets used in the full internal test suite are removed from the image prior to the generation of the final artifact in order to reduce the image size.

While these measures help in dealing with the complexity introduced by the high number of students involved in the project over its duration, they are, in our experience, not sufficient in and of themselves. Rather, a person who maintains an overall overview of the project, performs on- and off-boarding and is responsible for organizing communication with the project partner is necessary at each of the project partners. This does not need to be a full-time position and, depending on the funds available, can be an additional role given to existing team members. It is, however, a position that needs to be clearly defined and present as a constant in such a dynamic environment.

5 Conclusion

Several years ago, we started the described collaboration motivated by the idea that research software engineering might profit from being performed as a joint venture of research and software engineering. While not without challenges, this has proven to be a beneficial experiment for us, leading to the creation of a new robust quality control pipeline that is being productively used on hundreds of NGS runs every year.

None of the things we have learned during that journey and described here are entirely novel in and of themselves - or indeed very surprising in hindsight. We do believe, however, that the description of our experiences in this process, its benefits, and the lessons we learned from it can serve as a motivation and blueprint to flatten the learning curve for others who want to try this approach. In our experience, doing so leads to more sustainable research software, and, as a side effect, to more computer science students entering the market ready to become full-time research software engineers.

Bibliography

[ABD$^+$21] H. Anzt, F. Bach, S. Druskat, F. Löffler, A. Loewe, B. Y. Renard, G. Seemann, A. Struck, E. Achhammer, P. Aggarwal, F. Appel, M. Bader, L. Brusch, C. Busse, G. Chourdakis, P. W. Dabrowski, P. Ebert, B. Flemisch, S. Friedl, B. Fritzsch, M. D. Funk, V. Gast, F. Goth, J.-N. Grad, J. Hegewald, S. Hermann, F. Hohmann, S. Janosch, D. Kutra, J. Linxweiler, T. Muth, W. Peters-Kottig, F. Rack, F. H. Raters, S. Rave, G. Reina, M. Reißig, T. Ropinski, J. Schaarschmidt, H. Seibold, J. P. Thiele, B. Uekermann, S. Unger, R. Weeber. An environment for sustainable research software in Germany and beyond: current state, open challenges, and call for action. *F1000Research* 9:295, Jan. 2021.
doi:10.12688/f1000research.23224.2
http://dx.doi.org/10.12688/f1000research.23224.2

[Che23] S. Chen. Ultrafast one-pass FASTQ data preprocessing, quality control, and deduplication using fastp. *iMeta* 2(2), May 2023.
doi:10.1002/imt2.107
http://dx.doi.org/10.1002/imt2.107

[CKHW14] C. Chen, S. S. Khaleel, H. Huang, C. H. Wu. Software for pre-processing Illumina next-generation sequencing short read sequences. *Source Code Biol. Med.* 9(1):8, May 2014.

[CW20] J. Cohen, M. Woodbridge. RSEs in Research? RSEs in IT?: Finding a suitable home for RSEs. 2020.
doi:10.48550/ARXIV.2010.10477
https://arxiv.org/abs/2010.10477

[DCF$^+$17] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, C. Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology* 35(4):316–319, Apr. 2017.
doi:10.1038/nbt.3820
http://dx.doi.org/10.1038/nbt.3820

[EGFF16] C. Endrullat, J. Glökler, P. Franke, M. Frohme. Standardization and quality management in next-generation sequencing. *Applied amp; Translational Genomics* 10:2–9, Sept. 2016.
doi:10.1016/j.atg.2016.06.001
http://dx.doi.org/10.1016/j.atg.2016.06.001

[EPF$^+$19] P. A. Ewels, A. Peltzer, S. Fillinger, J. Alneberg, H. Patel, A. Wilm, M. U. Garcia, P. Di Tommaso, S. Nahnsen. nf-core: Community curated bioinformatics pipelines. Apr. 2019.
doi:10.1101/610741
http://dx.doi.org/10.1101/610741

[FWH+24] F. Freyer, P. Wolk, B. Haldemann, M. Bittiehn, M. Lataretu, S. Fuchs, P. W. Dabrowski. QCUrchin. Nov. 2024.
doi:10.5281/zenodo.14092346
https://doi.org/10.5281/zenodo.14092346

[KCB+21] G. M. Kurtzer, Cclerget, M. Bauer, I. Kaneshiro, D. Trudgian, D. Godlove. hpcng/singularity: Singularity 3.7.3. 2021.
doi:10.5281/ZENODO.1310023
https://zenodo.org/record/1310023

[KM21] D. Katz, K. Mchenry. Research Software Sustainability: Lessons Learned at NCSA. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. HICSS. Hawaii International Conference on System Sciences, 2021.
doi:10.24251/hicss.2021.873
http://dx.doi.org/10.24251/HICSS.2021.873

[KMRH19] D. S. Katz, K. McHenry, C. Reinking, R. Haines. Research Software Development amp; Management in Universities: Case Studies from Manchester's RSDS Group, Illinois' NCSA, and Notre Dame's CRC. In *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*. IEEE, May 2019.
doi:10.1109/se4science.2019.00009
http://dx.doi.org/10.1109/SE4Science.2019.00009

[KOP+04] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, F. Bruhin. pytest. 2004.
https://github.com/pytest-dev/pytest

[LBTS17] J. Lu, F. P. Breitwieser, P. Thielen, S. L. Salzberg. Bracken: estimating species abundance in metagenomics data. *PeerJ Computer Science* 3:e104, Jan. 2017.
doi:10.7717/peerj-cs.104
http://dx.doi.org/10.7717/peerj-cs.104

[LG20] P. Liu, J. Guitart. Performance comparison of multi-container deployment schemes for HPC workloads: an empirical study. *The Journal of Supercomputing* 77(6):6273–6312, Nov. 2020.
doi:10.1007/s11227-020-03518-1
http://dx.doi.org/10.1007/s11227-020-03518-1

[LS12] B. Langmead, S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods* 9(4):357–359, Mar. 2012.
doi:10.1038/nmeth.1923
http://dx.doi.org/10.1038/nmeth.1923

[MWX+24] V. Mallawaarachchi, A. Wickramarachchi, H. Xue, B. Papudeshi, S. R. Grigson, G. Bouras, R. E. Prahl, A. Kaphle, A. Verich, B. Talamantes-Becerra, E. A. Dinsdale, R. A. Edwards. Solving genomic puzzles: computational methods for metagenomic binning. *Brief. Bioinform.* 25(5), July 2024.

[pyt] pytest-workflow.
https://github.com/LUMC/pytest-workflow

[qcu] QCUrchin git repository.
https://dev.azure.com/RKI-HTW/NGS-QC/_git/QCurchin

[rma24] rmarkdown: Dynamic Documents for R. 2024. R package version 2.26.2.
https://github.com/rstudio/rmarkdown

[SKM+18] S. L. R. Stevens, M. Kuzak, C. Martinez, A. Moser, P. Bleeker, M. Galland. Building a local community of practice in scientific programming for life scientists. *PLOS Biology* 16(11):e2005561, Nov. 2018.
doi:10.1371/journal.pbio.2005561
http://dx.doi.org/10.1371/journal.pbio.2005561

[WLL19] D. E. Wood, J. Lu, B. Langmead. Improved metagenomic analysis with Kraken 2. *Genome Biology* 20(1), Nov. 2019.
doi:10.1186/s13059-019-1891-0
http://dx.doi.org/10.1186/s13059-019-1891-0

[WWG21] L. Wratten, A. Wilm, J. Göke. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature Methods* 18(10):1161–1168, Sept. 2021.
doi:10.1038/s41592-021-01254-9
http://dx.doi.org/10.1038/s41592-021-01254-9