Project Skeletons for Scientific Software

Michael Riesch, Michael Haider, and Christian Jirauschek
Department of Electrical and Computer Engineering
Technical University of Munich
Arcisstraße 21, 80333 Munich, Germany
Email: michael.haider@tum.de

***Abstract*—Although research relies heavily on software packages such as mathematical libraries or data analysis tools, efforts to provide high-quality scientific software are hardly rewarded. As a possible way out of this dilemma, project skeletons can be employed to accelerate software development while ensuring code quality. In this work, we review existing project skeletons and present a skeleton template for a C++ library with Python bindings as an example.**

I. Introduction

A common dilemma in science is that research relies heavily on software packages such as mathematical libraries or data analysis tools, but it is hard to get funding for developing and maintaining such packages [1]. While the latter efforts are hardly acknowledged, they are prone to consume a lot of time. This is underlined by the amount of published literature on best practices in software engineering in general (for example, the influential "Pragmatic Programmer" [2]) and in scientific contexts in particular (e.g. [3]–[5]). Naturally, apart from time the tasks require knowledge in software engineering which may constitute a considerable barrier for scientists from other fields than computer science [1], [5]. There are development guidelines and best practices guides (e.g., those published by the German Aerospace Center (DLR) [6] and the Netherlands eScience Center [7]), which give scientists valuable recommendations. However, they are intentionally kept general to be applicable to as many software projects as possible. Therefore, if one wants to create a certain project, the implementation of best practices must be carried out from scratch for the given programming language, which is tedious and time-consuming. It would be more advantageous if those repetitive setup tasks were automatized to reduce the time and knowledge required. In this context, ready-to-use project skeletons would allow scientists to focus on the actual implementation, which is typically a formidable challenge on its own.

II. Review of Existing Project Skeletons

We focus on three exemplary projects: a C++ project representing software projects that base on a compiled language, a Python project as an example of an interpreted programming language, and a LaTeX project that stands for miscellaneous software. Due to the complex compilation and linking process of C++, a quite intricate project skeleton is required. Several approaches are publicly available, out of which we consider the work in [8] as the most complete solution. For Python projects, we found the skeleton in [9] very helpful. Finally, we could not find a skeleton for LaTeX projects yet. Table I gives an overview of the implemented best practices and possible implementation candidates for the three projects.

This work was supported by the German Research Foundation (DFG) within the Heisenberg program (JI 115/4-2).

TABLE I
Overview of best practices in software engineering for scientific projects. For each best practice and project type, one possible implementation is listed. The asterisk denotes our suggested additions. Adapted from previous work [10].

Best practice	**C++** [8]	**Python** [9]	**LaTeX***
Version control system	git		
Collaboration platform	GitLab, GitHub		
Workflow	GitLab Flow, GitHub Flow		
Code formatting tool	clang-format	black*	latexindent
Static code analysis	clang-tidy	pylint*	lacheck
Use open file formats	e.g., JSON, CSV, HDF5		
Open-source libraries	e.g., FFTW, GNU Scientific Library		
Continuous integration	gitlab-ci, Travis CI		
Build automation	CMake	not req.	CMake
Function reference	Doxygen	Sphinx	docstrip
Documentation	Markdown	reStructuredText	LaTeX
Unit test framework	Catch2	PyTest	not req.
Code coverage report	gcov*	Coverage.py*	not req.
Deployment	conda*	PyPI	ctan
Online documentation	GitLab Pages, GitHub Pages		

III. C++ Library/Python Interface Skeleton

Rather than considering C++ and Python projects separately, we turn our attention towards a combination of the two. In scientific computing, such a combination is quite common as it combines the computational performance of C++ with the flexibility and brevity of Python. Basically, there are two ways to realize such a combination: either Python code is annotated (e.g., using Cython) in order to create a C/C++ extension automatically, or a Python interface is generated automatically from a C++ library. In recent work, we chose to do the latter with the help of the SWIG project and found that the required build steps are quite complex. As no project skeleton was available, we decided to develop bertha, an open-source skeleton for C++ libraries with Python interface [10]–[12]. Currently, this skeleton implements all key elements in Tab. I using the candidates for C++. This alone would make bertha the most comprehensive project skeleton for a C++ project. Additionally, the required steps to build and install a Python interface module have been implemented. The deployment is enabled with the help of a conda feedstock [13].

978-1-7281-6086-3/20/$31.00 ©2020 IEEE

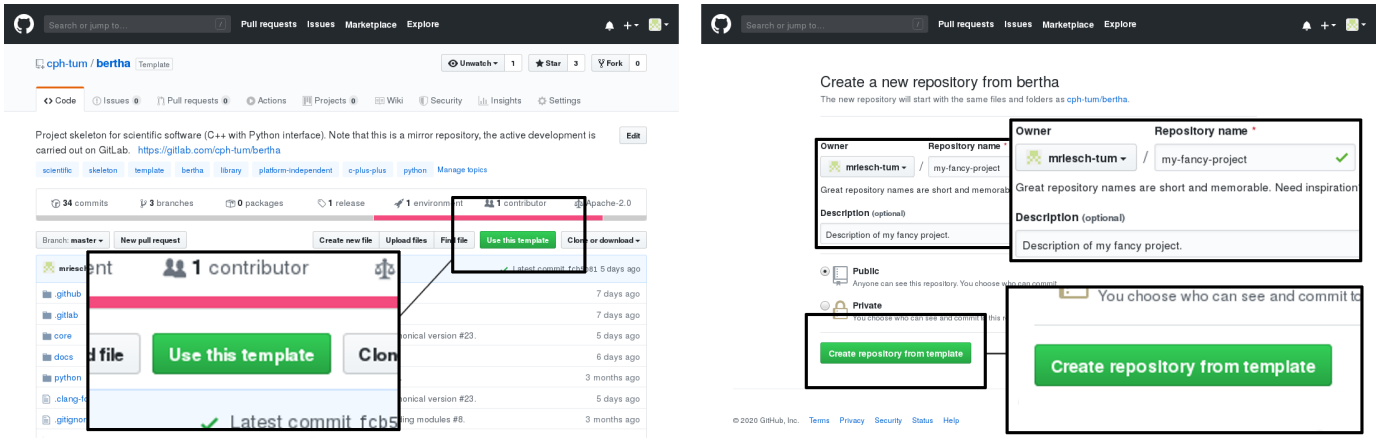

Fig. 1. Creating an instance of bertha on GitHub. On bertha's project page [11], click on "Use this template" and enter the desired owner, repository name, and project description. The button "Create repository from template" will then create the instance. Reprinted from M. Riesch et al. [10] (CC BY 4.0).

IV. Creating an Instance of the Project Skeleton

GitHub provides a simple mechanism to share a certain repository as a project template. After enabling this mechanism in the repository's settings, other users can generate a project based on the skeleton repository. Figure 1 describes the steps required to create a new instance of a skeleton. By following those steps using the bertha template, we created a software project for the simulation of rapidly tunable Fourier domain mode-locked (FDML) fiber lasers [14], [15]. It should be noted that the ongoing efforts are not going to be published in the near future. This, however, is allowed by the permissive license of the skeleton. The FDML simulation tool requires several third-party packages, e.g. scientific libraries (FFTW, GNU Scientific Library, etc.), and libraries for storing output data (HDF5). Those dependencies are conveniently installed using conda and are detected by the CMake build system. Another use case for the bertha skeleton is mbsolve [16], [17], an open-source solver for the Maxwell-Bloch equations [18]. Here, the software project existed before the skeleton, which we used as reference implementation or best practice guideline.

V. Conclusion

In the contribution at hand, we have reviewed the state of the art in project skeletons that facilitate the implementation of good software engineering practices in scientific software projects. For the important use case of a C++ library with Python bindings, we have presented our own skeleton along with two current use cases. There are, however, various types of software projects without a corresponding project skeleton. For example, a skeleton for a LaTeX project would facilitate the collaboration between researchers in the scope of scientific publications, and foster the use of so-called stock image projects [19]. In the scope of future work, such a skeleton shall be developed.

References

[1] A. Nowogrodzki, "How to support open-source software and stay sane," *Nature*, vol. 571, no. 7763, pp. 133–134, Jul. 2019.

[2] A. Hunt and H. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, 1st ed. Addison-Wesley, Boston, 1999.

[3] W. Bangerth and T. Heister, "What makes computational open source software libraries successful?" *Comp. Sci. Disc.*, vol. 6, p. 015010, Nov. 2013.

[4] A. Prlić and J. B. Procter, "Ten simple rules for the open development of scientific software," *PLoS Comput. Biol.*, vol. 8, no. 12, p. e1002802, Dec. 2012.

[5] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good enough practices in scientific computing," *PLoS Comput. Biol.*, vol. 13, no. 6, p. e1005510, Jun. 2017.

[6] T. Schlauch, M. Meinel, and C. Haupt, "DLR software engineering guidelines," https://doi.org/10.5281/zenodo.1344612, Aug. 2018.

[7] Netherlands eScience Center, "Software development guide," https://guide.esciencecenter.nl, Aug. 2019.

[8] GitHub user kracejic, "Clean C++ project for you to use," https://github.com/kracejic/cleanCppProject, Oct. 2015.

[9] A. Ioannides, "Python package template project for kick-starting new Python projects." https://github.com/AlexIoannides/py-package-template, Nov. 2018.

[10] M. Riesch, T. D. Nguyen, and C. Jirauschek, "bertha: Project skeleton for scientific software," *PLOS ONE*, vol. 15, no. 3, p. e0230557, Mar. 2020.

[11] M. Riesch and C. Jirauschek, "bertha: Project skeleton for scientific software (C++ with Python interface)," https://github.com/cph-tum/bertha, Dec. 2019.

[12] ——, "bertha: Project skeleton for scientific software (C++ with Python interface)," https://gitlab.com/cph-tum/bertha, Dec. 2019.

[13] "Conda feedstock for bertha," https://github.com/conda-forge/bertha-feedstock, Dec. 2019.

[14] C. Jirauschek and R. Huber, "Modeling and analysis of polarization effects in Fourier domain mode-locked lasers," *Opt. Lett.*, vol. 40, no. 10, pp. 2385–2388, May 2015.

[15] ——, "Efficient simulation of the swept-waveform polarization dynamics in fiber spools and Fourier domain mode-locked (FDML) lasers," *J. Opt. Soc. Am. B*, vol. 34, no. 6, pp. 1135–1146, May 2017.

[16] M. Riesch and C. Jirauschek, "mbsolve: An open-source solver tool for the Maxwell-Bloch equations," https://github.com/mriesch-tum/mbsolve, Jul. 2017.

[17] M. Riesch, N. Tchipev, S. Senninger, H.-J. Bungartz, and C. Jirauschek, "Performance evaluation of numerical methods for the Maxwell–Liouville–von Neumann equations," *Opt. Quant. Electron.*, vol. 50, no. 2, p. 112, Feb. 2018.

[18] C. Jirauschek, M. Riesch, and P. Tzenov, "Optoelectronic device simulations based on macroscopic Maxwell–Bloch equations," *Adv. Theor. Simul.*, vol. 2, no. 8, p. 1900018, Aug. 2019.

[19] M. Riesch, "qclsip: The Quantum Cascade Laser Stock Image Project," http://doi.org/10.5281/zenodo.2641239, Jan. 2019.