

The Challenges of Reproducible Computations in Pharmacometrics

Best Practices and Solutions

Stella Belin, Rikard Nordgren
Department of Pharmacy, Uppsala University, Sweden

Introduction

For a computation to be reproducible it needs to generate exactly the same or similar results given the same inputs and methods. Reviews of the current situation have painted a grim picture of the state of reproducibility, showing that a range from 0% to 26% of studies could be reproduced [1,2,3,4]. Previously, attempts have been made to formalize what is needed to reach higher levels of reproducibility [5], and we would like to highlight one abstraction: the five pillars of reproducible computations [6]. This abstraction is founded in a fully scripted workflow, control over the

dependencies, documentation and persistent sharing of code and data. Reproducibility of computations is a hard problem, but there currently exist solutions that can solve the problem. We propose some solutions with different levels of effort, as reproducibility is a scale (i.e. doing something is better than nothing). From the proposed solutions, it follows that reproducibility is something that is important to keep in mind in the beginning of a project and follow throughout, since doing so later in the process can be hard, if not impossible.

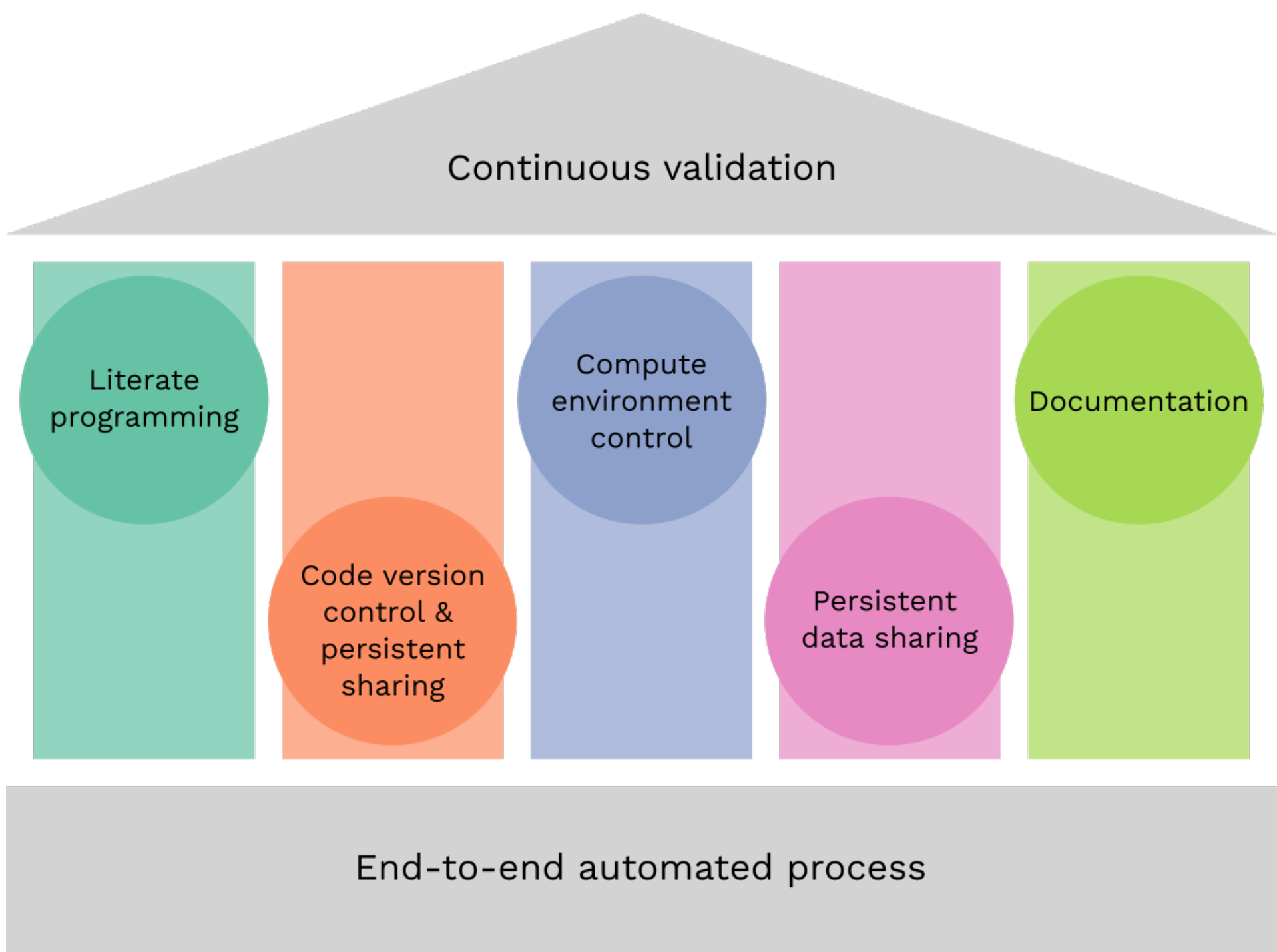


Figure 1. The five pillars of reproducibility (recreated from [6])

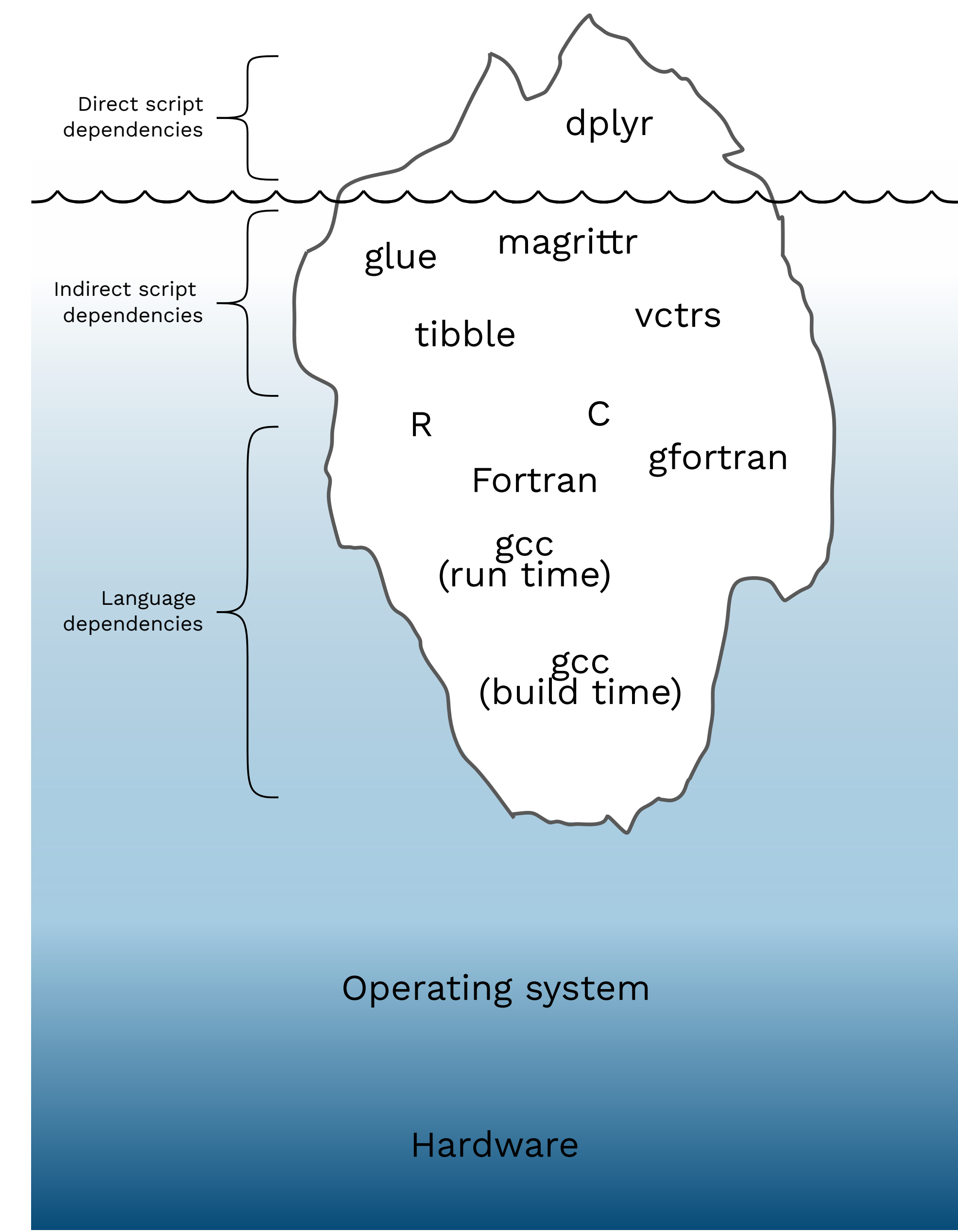


Figure 2. Iceberg representation of dependency hierarchy

Terminology	
Package manager Tool that handles package installation, removal, and updates	Run time dependencies Dependencies when executing your script/program. The versions of these may differ from compile time dependencies, potentially causing conflicts
Compiler Translates source code to generate an executable program (e.g. gfortran used to compile NONMEM, which is written in Fortran)	Direct dependencies Dependencies explicitly called by your script/program
Dependency Software package needed by another software package	Indirect/transitive dependencies Dependencies of direct dependencies, and their dependencies etc.
Compile time dependencies Dependencies when compiling a program	

Tier 1: Scripting and package management

To make reproducibility possible in practice the computations need to be scripted. The dependencies of the script should be recorded and included, with full lists with versions of all dependencies. The package managers of the programming languages could be used to simplify this, for example pip/requirements.txt for Python and renv/lock-files in R. Limiting the number of dependencies can also be useful. We also recommend deciding whether to freeze your dependencies at the start or rerunning the project at the end and then recording the dependency versions.

Tier 2: Containers

Containers [7,8] are isolated systems that can run inside of another system. After building a container the computations can be run inside it with the versions of all software being frozen. The container file can then be given to someone attempting to reproduce the computations. This approach requires more setup than the previous tier but also captures more of the complex relationship between dependencies. Using a container manager like Docker can be used to build and share containers.

Tier 3: Fully reproducible systems

The main weakness of containers is management of the compile time dependencies, i.e. all software needed to build the containers. This means that it can be difficult to reproduce the container itself. With the advent of the nix and guix functional package managers [9,10] complete solutions to the problem of reproducible computations exist. These will also take care of transitive compile time dependencies and should also be able to make NONMEM runs exactly reproducible for each version.

Highlights

- Script your workflows
- Record your dependencies
- Use a package manager
- Limit the number of dependencies
- Decide dependency strategy early

Highlights

- Use to capture dependencies outside of script
- Limit the number of dependencies in each container
- Use Docker for container management
- Record how container was built

Highlights

- Use guix or nix to capture the full dependency tree
- Does not support all software currently

List of references

[1] Ioannidis JP, Allison DB, Ball CA, Coulibaly I, Cui X, Culhane AC, Falchi M, Furlanello C, Game L, Jurman G, Mangion J, Mehta T, Nitzberg M, Page GP, Petretto E, van Noort V. Repeatability of published microarray gene expression analyses. Nat Genet. 2009 Feb;41(2):149-55. doi: 10.1038/ng.295

[2] Zaringhalam M, Federer L. Data and code for reproducible research: lessons learned from the NLM reproducibility workshop. Zenodo. 2020. 10.5281/zenodo.3818329

[3] Samuel S, Mletchen D. Computational reproducibility of Jupyter notebooks from biomedical publications. ArXiv. 2022; 2209.04308.

[4] Trisovic A, Lau MK, Pasquier T, Crosas M. A large-scale study on research code quality and execution. Sci Data. 2022;9:60

[5] van Kampen AHC, Mahamune U, Jongejan A, van Schaik BDC, Balashova D, Lashgari D, Pras-Raves M, Wever EJM, Dane AD, Garcia-Valiente R, Moerland PD. ENCORE: a practical implementation to improve reproducibility and transparency of computational research. Nat Commun. 2024 Sep 16;15(1):8117. doi: 10.1038/s41467-024-52446-8

[6] Ziemann M, Poulain P, Bora A. The five pillars of computational reproducibility: bioinformatics and beyond. Brief Bioinform. 2023 Sep 22;24(6):bbad375. doi: 10.1093/bib/bbad375

[7] Kurtzer GM, cclerget, Bauer M, Kaneshiro I, Trudgian D, Godlove D, hpcng/singularity: Singularity 3.7.3. (Apr. 06, 2021). Zenodo. doi: 10.5281/zenodo.4667718

[8] Merkel D. Docker: lightweight Linux containers for consistent development and deployment. Linux J, vol. 2014, no. 239, p. 2:2

[9] Dolstra E. The purely functional software deployment model. PhD thesis, ISBN 90-393-4130-3. 2006

[10] Courtès L, Wurmus R. Reproducible and User-Controlled Software Environments in HPC with Guix. Jul. 26, 2015, arXiv: arXiv: 1506.02822. doi: 10.48550/arXiv.1506.02822