Longitudinal Study of Software Environments Produced by Dockerfiles from Research Artifacts: Initial Design

Quentin Guilloteau, Antoine Waehren, Florina M. Ciorba

SHORT: Longitudinal Study of Software Environments Produced by Dockerfiles from Research Artifacts: Initial Design

Quentin Guilloteau
Quentin.Guilloteau@unibas.ch
University of Basel
Basel, Switzerland

Antoine Waehren
A.Waehren@stud.unibas.ch
University of Basel
Basel, Switzerland

Florina M. Ciorba
Florina.Ciorba@unibas.ch
University of Basel
Basel, Switzerland

ABSTRACT

The reproducibility crisis has affected all scientific disciplines, including computer science (CS). To address this issue, the CS community has established artifact evaluation processes at conferences and in journals to evaluate the reproducibility of the results shared in publications. Authors are therefore required to share their artifacts with reviewers, including code, data, and the software environment necessary to reproduce the results. One method for sharing the software environment proposed by conferences and journals is to utilize container technologies such as Docker and Apptainer. However, these tools rely on non-reproducible tools, resulting in non-reproducible containers.

In this paper, we present a tool and methodology to evaluate variations over time in software environments of container images derived from research artifacts. We also present initial results on a small set of `Dockerfiles` from the Euro-Par 2024 conference.

CCS CONCEPTS

• **Software and its engineering** → *Software post-development issues.*

KEYWORDS

Artifact Evaluation, Containers, Longevity, Sustainability

ACM Reference Format:
Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. SHORT: Longitudinal Study of Software Environments Produced by Dockerfiles from Research Artifacts: Initial Design. In *ACM Conference on Reproducibility and Replicability (ACM REP '25), July 29–31, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3736731.3746146

1 INTRODUCTION

The scientific community has been hit by the "Reproducibility Crisis" [2], and computer science makes no exception [4]. In order to improve the reproducibility of articles in Computer Science, the community set up, in conferences and journals, Artifact Description/Artifact Evaluation processes [23] where authors would *voluntary* submit the artifact required to reproduce the main results from their article. Then, artifact reviewers would try to reproduce those main results given the artifact from the authors. Based on the result of the reproduction attempt, the article is awarded *badges* to promote its reproducibility aspect [1]. The link to the artifact can be found in the published version of the article, sometimes with an appendix giving more information about the artifact (*e.g.,* download instructions, hardware requirements, workflow) [21, 22, 31].

One difficult aspect of creating a truly reproducible artifact is to properly control the software environment: packages and libraries versions. Several works already shed light on the possible variability of experimental results given a badly captured software environment [28, 32]. To address this issue, the conference Artifact Description/Artifact Evaluation committees have advised authors to provide containers or virtual machines to share their artifacts in order to "freeze" their software environment [7, 9, 30]. In practice, about 12% of the artifacts use such tools [10]. In the case of usual virtualization techniques to generate containers or virtual machines, authors have to write the "recipe" (for example, `Dockerfile` for Docker containers) made of a list of commands to execute from a "base image" (the `FROM` instruction in `Dockerfiles`). This approach has several limitations. First, these base images can vary, either because of bad practices from the users (*e.g.,* using the `latest` version of the images) or from the administrators of the image (*e.g.,* pushing a different image to a previously defined tag). Administrators can also remove these base images, making all recipes un-buildable [29]. Another limitation of this approach is the heavy dependency on non-reproducible tools such as classical package managers (*e.g.,* `apt`, `dpkg`, `yum`). Hence, rebuilding a recipe of a container or a virtual machine can yield different results from the authors' image, and thus endangers the reproducibility of the results. One would be for the authors to store and then provide the built container itself (*i.e.,*tarball) on some long-term storage (*e.g.,* Zenodo [34]). But this comes with the downside of requiring more storage space than just storing the recipe (binary vs. text file), and thus has some **severe scalability and sustainability issues in terms of longevity** [10, 27]. Moreover, storing the built binary image exhibits limitations when needed to *inspect* the actual content of the image, and when a modification of the software environment capture by this image is needed [26]. Having the opportunity to extend previous scientific work is a crucial aspect in any scientific field. To the best of our knowledge, no previous work has exhibited the variations in software environment build from `Dockerfiles` through time.

In this paper, we propose our **initial design of a longitudinal study of the variations of the software environments produced by `Dockerfiles` from research artifacts.** We show initial results from five `Dockerfiles` extracted from the Euro-Par 2024 conference reproducibility artifacts built every month for seven months, from October 1^{st} 2024 to April 1^{st} 2025. We focus only on Docker containers as they are the most used in artifacts [10].

ACM REP '25, July 29–31, 2025, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1958-5/2025/07
https://doi.org/10.1145/3736731.3746146

The remainder of this paper is organized as follows. Section 2 presents ecg, our software framework for this longitudinal study. The study protocol is presented in Section 3. Section 4 presents the initial results of a small-scale version of the study. Finally, we conclude and give perspectives in Section 5.

2 THE ECG FRAMEWORK

In this section, we present ecg, a tool for building Dockerfiles from research artifacts and collecting information on their resulting software environment. ecg is a 280-line Python3 script [11] under the GPLv3 license.

2.1 Information captured by ecg

This section presents the different information collected by ecg.

2.1.1 Cryptographic Hash of the Artifact. The authors can share artifacts in several ways, but not all ways have the same guarantees in terms of longevity and "stability" of the link. Hence, ecg captures the result of the download at the given link (success or error). In case of a successful download, ecg computes the cryptographic hash of the download result to capture any variation in the source of the artifact which could later explain variations in the resulting software environment.

For example, authors can link to a Zenodo archive that will *always* point to the latest artifact version. Hence, the content of the artifact behind this link can change over time.

2.1.2 Docker Building Error. In the case of Dockerfile failing to build, ecg captures the error returned by the building process (*e.g.,* missing image, failing command). In fact, ecg is only capable of capturing the *first error* of the build even if there might be several errors in the building process. The Docker error is extracted by greping the building process log.

2.1.3 List of Installed Packages and their Version. The software environment can be composed of several packages, which can have been installed by different means. The most common way is through package managers (*e.g.,* apt, dpkg, yum). But there are also packages installed for programming languages such as with pip for Python. Another way to install the package is to download the sources and build them locally. ecg is able to capture these different ways to install packages. Hence, for each package, ecg extracts the package name, the version, and the tool used to install the package.

Currently, ecg is capable of querying information from popular package managers (dkpg (and thus apt), pacman, pip, conda) and to deal with situations where packages are installed by source (package_managers in Listing 1), or the case of packages installed in virtual Python environments. In the case where there is a git clone command in the Dockerfile (git_packages in Listing 1), ecg will extract the commit of the local clone of the repository after the successful build of the container, and use this commit as a "version". In the case of a download of the sources of a package (misc_packages in Listing 1), for example with wget or curl, there is no immediate way to log the version of the package that is being downloaded. The URL of the source code might contain a version number, but this URL might in the future point to a different version of the source code. Hence, ecg will download *locally* (outside the container) the source code pointed by the URL and compute the cryptographic hash of the result to generate a "version".

2.2 Artifact representation

The information captured by ecg cannot be automatically extracted from a Dockerfile. In some cases, authors of artifact can have shell scripts installing packages (*e.g.,* calling apt install), which would be invisible at the Dockerfile level. With ecg, a Dockerfile from an artifact is represented as a Nickel configuration file [33], which is similar to JSON or YAML. One of Nickel's features is the possibility to *verify* that a configuration file is valid with respect to a *contract*. The Nickel file representing the artifact is then *verified* against the contract [12] and translated to a JSON file given as input to ecg. Listing 1 shows an example of a Nickel representation. The contract is versioned (line 2 of Listing 1) to take into account potential future changes to the descriptions and the contract. If the Nickel file does not respect the contract, an error is reported to the user.

```
1  {
2    version = "1.0",
3    artifact_url = "https://zenodo.org/.../artifact.zip",
4    type = "zip",
5    doi = "10.5281/zenodo.XXXXXXXX",
6    conf_date = 2024,
7    virtualization = "docker",
8    buildfile_dir = "artifact",
9    package_managers = [ "dpkg", "pip" ]
10   git_packages = [
11     { name = "spack", location = "/home/user/spack" }
12   ],
13   misc_packages = [
14     {
15       name = "cmake-3.22.2-linux",
16       url = "https://github.com/Kitware/CMake/releases/
       download/v3.22.2/cmake-3.22.2-linux-x86_64.sh"
17     }
18   ]
19 }
```

Listing 1: Example of Artifact representation in Nickel

2.3 Workflow

The workflow of ecg is shown in Figure 1. The workflow itself [13] is governed by the Snakemake workflow manager [24], and its software environment [14] is managed by the Nix functional package manager [8]. These tools provide reproducibility and robustness to the workflow setup to ensure that we will be able to execute the exact same workflow in the exact same conditions for all the future builds of the artifacts.

3 PROTOCOL

This section presents the initial protocol of our longitudinal study.

(1) For each conference with an artifact evaluation process, we examine all artifacts and consider only those that share their software environment using a Dockerfile.
(2) From the artifact and the Dockerfile we generate the associated Nickel file (Listing 1) which contains information such as the artifact's location, how to extract the Dockerfile from the artifact and the package managers used.
(3) Once all Nickel descriptions for the artifacts of a conference have been completed, we use ecg to construct the download, build, and extract information from each of them.

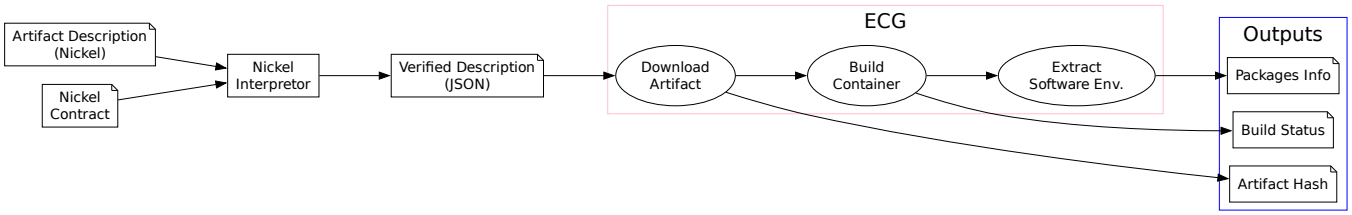

Figure 1: Workflow of ecg. Each description of an artifact is verified with the Nickel contract and then converted in a JSON representation. This JSON representation is then read by ecg to download the artifact, compute the hash of its content, build the container from the `Dockerfile`, and extract the software environment information from the built container. ecg outputs files containing the information about the artifact and its `Dockerfile` .

(4) Prior to building each container, the local Docker cache is cleared on these machines.
(5) We will execute ecg on each artifact every month, as close to the beginning of the month as possible, for a full year (*i.e.,* 13 months).

Details.

- We ensure that an empty local Docker cache is used before building the containers.
- Each container is allocated a 10-minute time frame to build.
- ecg is executed on the x86-64 machines of the *Grid'5000* platform [3] on the dahu cluster of the Grenoble site.
- We do not try to reproduce the experiment results from the artifacts, but only to build the container. The impact of software environment variations on the experiments results has been studied for example in [28, 32].

4 PRELIMINARY RESULTS

We wrote the descriptions in Nickel (see Section 2.2) for the five artifacts of the Euro-Par 2024 conference that used a Docker container to share their software environments, and used ecg to build their associated `Dockerfiles` once every month for seven months. The initial build was October 1^{st} 2024, then the following builds were on November 4^{th} 2024, December 2^{nd} 2024, January 2^{nd} 2025, February 5^{th} 2025, March 12^{th} 2025, and April 1^{st} 2025. The deadline for the authors to submit their artifacts to Euro-Par 2024 was May 15^{th} 2024. Hence, our initial build took place almost 5 months after the artifacts submission. The following sections follow the dimensions captured by ecg (see Section 2.1). The data collected and presented in this section is available on Zenodo [20].

4.1 Hash of Artifacts

Four out of the five articles provided a Zenodo archive to the artifact, the other provided a GitHub link with a specified `git` tag.

No variation of the artifact sources was observed, via computation of their cryptographic hash.

4.2 Build Status

All of the artifacts managed to successfully produce a container.

4.3 Software Environment

Table 1 summarizes the information about the studied `Dockerfiles`. We can see that most of the container used the `ubuntu` base image and that *all* the `Dockerfiles` specified the version of the base image to use, thus limiting the variations in terms of the software environment produced. However, all `Dockerfiles` called the `apt-get update` command, which fetches the latest versions of the packages every time and thus introduces variations.

Artifact	Docker Base Image used		Calling `apt update`?
	Name	Version	
`canon_solving` [15]	`ubuntu`	`22.04`	Yes
`geijer_how` [16]	`ubuntu`	`22.04`	Yes
`hiraga_peanuts` [17]	`devcontainers/cpp`	`1-debian-12`	Yes
`munoz_fault` [18]	`ubuntu`	`22.04`	Yes
`wolff_fast` [19]	`ubuntu`	`22.04`	Yes

Table 1: Information about the `Dockerfiles` from the study.

Figure 2 depicts the evolution through time of the software environment produced by `Dockerfile` of each artifact studied. We can see that even after *a single month* of the initial build, the software environment is already different for *all* of the artifacts studied. This is a severe threat to reproducibility as one month is the time frame for an artifact evaluation process in conferences. This means that the `Dockerfile` submitted by the authors can generate a different software environment between the submission of the artifact and the reproduction attempt of the reviewers.

We can see that every month new versions of the packages are being introduced in the software environments of the containers.

Figure 3 shows the evolution of the packages' versions managed by each tool present in the studied artifact: `dpkg`, `git`, `pip`, and manual download and build (`misc`). We can see that the tools that introduce variation in the software environment are `dpkg` and `pip`.

The packages that changed the most in terms of versions are:

(1) `linux-libc-dev:amd64` (installed via `dpkg`) with 7 different versions for each artifact (1 per new build attempt), except for `hiraga_peanuts` where there were 4 different versions.
(2) `numpy` (installed via `pip`) with 6 different versions for the artifact `wolff_fast`
(3) `fonttools` (installed via `pip`) with 5 different versions for the artifacts `geijer_how` and `wolff_fast`

5 CONCLUSION AND PERSPECTIVES

This paper presented the initial design of our longitudinal study of the evolution of the software environments produced by `Dockerfiles`.

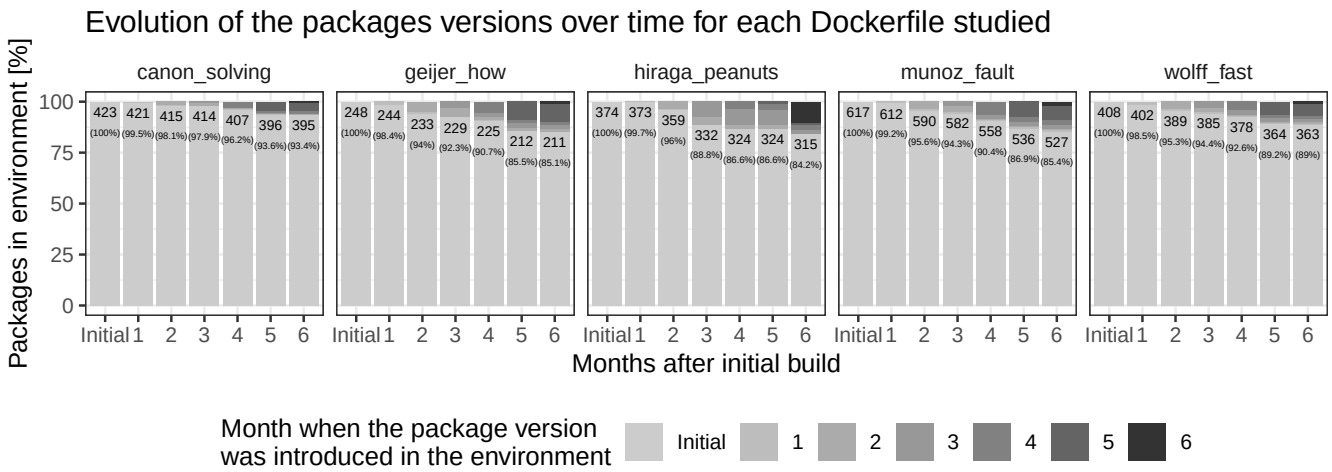

Figure 2: Evolution of the packages in the software environment of each container over time. The color of the bar corresponds to the month when a specific version of a package has been introduced in the software environment. We can see that the decrease of the proportion of package versions similar to the versions in the initial build over time.

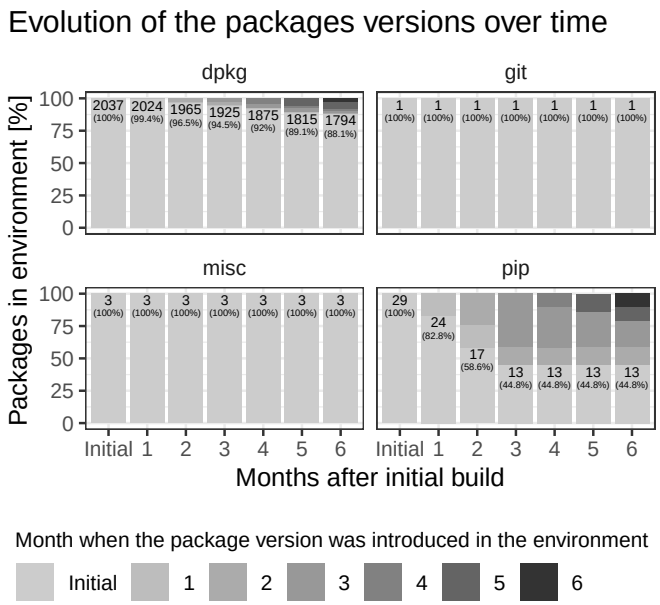

Figure 3: Evolution through time of the packages in the software environments managed by each tool present in the studied artifacts.

We presented in Section 2 `ecg`, and its workflow (Section 2.3), a tool to download, build, and extract information from research artifacts `Dockerfiles`. Section 3 presented the protocol to observe the potential variations in the software environment over time. We showed initial results in Section 4 on five artifacts from the Euro-Par 2024 conference built every month for seven months.

While this initial study only considered five `Dockerfiles`, it revealed that **the software environments produced by the considered `Dockerfiles` vary *every month*** which highlights the lack of reproducibility of such tools and methods to share software environments. The question of the results' significance on such a small sample will need to be answered in a larger-scale study.

A practice to deal with the lack of reproducibility of `Dockerfiles` is that the authors store the built container on long-term storage such as Zenodo. However, this practice raises serious questions in terms of scalability and sustainability in terms of storage [27]. The lack of reproducibility of `Dockerfiles` should not be hidden behind long-term storage: the practices of the authors and of Artifact Description/Artifact Evaluation need to be re-evaluated.

The community may want to adopt instead solutions *tailored for long-term reproducibility* of software environment [25] (*i.e.,* Nix [8] and Guix [6]), where authors can share a textual representation of their software environment which could be precisely reproduced and with longevity guarantees [5]. Moreover, Nix and Guix allow the precise introduction of variation in the software environment, which is a crucial feature (not present in container technologies) in order to build upon previous artifact for future publications [26].

Future Perspectives. This paper serves as the foundation for the design of a larger future study on the longevity of containers. We will consider not only Docker containers, but also other containerization tools such as Apptainer. Therefore, we will write descriptions of more artifacts from more conferences, aiming at a total of approximately a hundred containers, from conferences spanning various scientific disciplines and with varying levels of computer science technical expertise.

ACKNOWLEDGMENTS

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

This research was funded in part by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957407 (DAPHNE).

REFERENCES

[1] ACM. [n. d.]. Artifact review badging. https://www.acm.org/publications/policies/artifact-review-badging. Accessed: 2023-04-04.

[2] Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature* 533, 7604 (May 2016), 452–454. https://doi.org/10.1038/533452a

[3] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. 2013. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*, Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan (Eds.). Communications in Computer and Information Science, Vol. 367. Springer International Publishing, 3–20. https://doi.org/10.1007/978-3-319-04519-1_1

[4] Christian Collberg, Todd Proebsting, and Alex M Warren. 2015. Repeatability and Benefaction in Computer Systems Research - A Study and a Modest Proposal. (2015), 68.

[5] Ludovic Courtès, Timothy Sample, Stefano Zacchiroli, and Simon Tournier. 2024. Source Code Archiving to the Rescue of Reproducible Deployment. In *Proceedings of the 2nd ACM Conference on Reproducibility and Replicability*. 36–45.

[6] Ludovic Courtès. 2013. Functional Package Management with Guix. *arXiv:1305.4584 [cs]* (May 2013). http://arxiv.org/abs/1305.4584

[7] CS Artifacts. [n. d.]. Resources on the Artifact Evaluation (AE) Process. https://github.com/csartifacts/resources?tab=readme-ov-file#packaging-platforms. Accessed: 2025-04-02.

[8] Eelco Dolstra, Merijn de Jonge, and Eelco Visser. 2004. Nix: A Safe and Policy-Free System for Software Deployment. (2004), 14.

[9] EuroSys 2025. [n. d.]. Artifact Packaging. https://sysartifacts.github.io/eurosys2025/packaging#formats. Accessed: 2025-04-02.

[10] Quentin Guilloteau, Florina M Ciorba, Millian Poquet, Dorian Goepp, and Olivier Richard. 2024. Longevity of Artifacts in Leading Parallel and Distributed Systems Conferences: a Review of the State of the Practice in 2023. In *REP 2024 - ACM Conference on Reproducibility and Replicability*. ACM, Rennes, France, 1–14. https://doi.org/10.1145/3641525.3663631

[11] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. . https://archive.softwareheritage.org/swh:1:cnt:e451b603eb672f625ec06ba34550dea1d0388a5f;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/ecg/app/ecg.py

[12] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:297fa4d4bc26d417fc6eb92c763ccf5742eefcb5;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/workflow/nickel/artifact_contract.ncl

[13] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:f7a970ff0ca3fea98829c22469fc7f353a70a91e;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/workflow/measure.smk

[14] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:9291a1d56146a7db10fd5addfb1bf7121a65e3a7;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/workflow/envs/snakemake.nix

[15] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:22fc9eb93654be563e15de863c5789c03cb5f608;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/artifacts/nickel/europar24/canon_solving.ncl

[16] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:f6c098519cb29647c10e6b1d28c21693af153ab6;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/artifacts/nickel/europar24/geijer_how.ncl

[17] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:b9e6944490f7bdf4fa34c6899baefc5049b89ad4;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/artifacts/nickel/europar24/hiraga_peanuts.ncl

[18] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:d3aa33691e1b1eb6ceeae1df347271fabad8d32c;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/artifacts/nickel/europar24/munoz_fault.ncl

[19] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. https://archive.softwareheritage.org/swh:1:cnt:7c5b42e93112ebd97e14eee1a83ba0d033a4ef83;origin=https://github.com/GuilloteauQ/study-docker-repro-longevity;visit=swh:1:snp:0b338477dcb75d8801fa1a1035558957e33444e4;anchor=swh:1:rev:d685d19a4905dc59f699b508a890cd2fc282adb9;path=/artifacts/nickel/europar24/wolff_fast.ncl

[20] Quentin Guilloteau, Antoine Waehren, and Florina M. Ciorba. 2025. *Variations of software environments produced by Dockerfiles from the Euro-Par 2024 conference artifacts*. https://doi.org/10.5281/zenodo.15132467

[21] Neil Chue Hong, Brian Hole, and Samuel Moore. 2013. Software papers: improving the reusability and sustainability of scientific software. *URL* 10, 6084 (2013), m9.

[22] Donghyun Kang, TaeYoung Kang, and Junkyu Jang. 2023. Papers with code or without code? Impact of GitHub repository usability on the diffusion of machine learning research. *Information Processing & Management* 60, 6 (2023), 103477.

[23] Mallory C Kidwell, Ljiljana B Lazarević, Erica Baranski, Tom E Hardwicke, Sarah Piechowski, Lina-Sophia Falkenberg, Curtis Kennett, Agnieszka Slowik, Carina Sonnleitner, Chelsey Hess-Holden, et al. 2016. Badges to acknowledge open practices: A simple, low-cost, effective method for increasing transparency. *PLoS biology* 14, 5 (2016), e1002456.

[24] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (2012), 2520–2522.

[25] Julien Malka, Stefano Zacchiroli, and Théo Zimmermann. 2024. Reproducibility of build environments through space and time. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 97–101.

[26] Michael Mercier, Adrien Faure, and Olivier Richard. 2018. Considering the Development Workflow to Achieve Reproducibility with Variation. In *SC 2018-Workshop: ResCuE-HPC*. 1–5.

[27] John Monroe. [n. d.]. Preservation or Deletion: Archiving and Accessing the Dataverse. ([n. d.]).

[28] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F Sweeney. [n. d.]. Producing Wrong Data Without Doing Anything Obviously Wrong! ([n. d.]), 12.

[29] Nvidia. [n. d.]. Container Support Policy. https://archive.softwareheritage.org/swh:1:cnt:167b243e20f4f7e38efcd1c9d1696f291de6c5e0;origin=https://gitlab.com/nvidia/container-images/cuda;visit=swh:1:snp:62fbbe6c6441981445c19b0632f4bc0c69736d12;anchor=swh:1:rev:e3ff10eab3a1424fe394899df0e0f8ca5a410f0f;path=/doc/support-policy.md. Accessed: 2024-01-24.

[30] Rohan Padhye. [n. d.]. Artifact Evaluation: Tips for Authors. https://blog.padhye.org/Artifact-Evaluation-Tips-for-Authors/#tip-1-submit-a-friendly-package. Accessed: 2025-04-02.

[31] paperswithcode. [n. d.]. Papers with code. https://paperswithcode.com/. Accessed: 2024-01-24.

[32] Andrzej Sokołowski, Nikhil Bhagwat, Dimitrios Kirbizakis, Yohan Chatelain, Mathieu Dugré, Jean-Baptiste Poline, Madeleine Sharp, and Tristan Glatard. 2024. The impact of FreeSurfer versions on structural neuroimaging analyses of Parkinson's disease. *bioRxiv* (2024), 2024–11.

[33] Tweag. [n. d.]. Nickel. https://github.com/tweag/nickel. Accessed: 2024-03-14.

[34] zenodo. [n. d.]. Zenodo. https://zenodo.org/. Accessed: 2023-03-30.