Automatic Reproduction of Workflows in the Snakemake Workflow Catalog and nf-core Registries

Samuel Grayson
grayson5@illinois.edu
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, Illinois, USA

Darko Marinov
marinov@illinois.edu
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, Illinois, USA

Daniel S. Katz
d.katz@ieee.org
NCSA & CS & ECE & iSchool
University of Illinois Urbana-Champaign
Urbana, Illinois, USA

Reed Milewicz
rmilewi@sandia.gov
Department of Software Engineering and Research
Sandia National Laboratories
Albuquerque, New Mexico, US

ABSTRACT

Workflows make it easier for scientists to assemble computational experiments consisting of many disparate components. However, those disparate components also increase the probability that the computational experiment fails to be reproducible. Even if software is reproducible today, it may become irreproducible tomorrow without the software itself changing at all, because of the constantly changing software environment in which the software is run.

To alleviate irreproducibility, workflow engines integrate with container engines. Additionally, communities that sprung up around workflow engines started to host registries for workflows that follow standards. These standards reduce the effort needed to make workflows automatically reproducible.

In this paper, we study automatic reproduction of workflows from two registries, focusing on non-crashing executions. The experimental data lets us analyze the upper bound to which workflow engines could achieve reproducibility. We identify lessons learned in achieving reproducibility in practice.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; • **Information systems** → **Information systems applications**; • **Applied computing** → *Digital libraries and archives.*

KEYWORDS

reproducibility, workflow engines, research software engineering

ACM Reference Format:
Samuel Grayson, Darko Marinov, Daniel S. Katz, and Reed Milewicz. 2023. Automatic Reproduction of Workflows in the Snakemake Workflow Catalog and nf-core Registries. In *2023 ACM Conference on Reproducibility and Replicability (ACM REP '23), June 27–29, 2023, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3589806.3600037

ACM REP '23, June 27–29, 2023, Santa Cruz, CA, USA

ACM ISBN 979-8-4007-0176-4/23/06…$15.00
https://doi.org/10.1145/3589806.3600037

1 INTRODUCTION

In recent years, scientific workflows have become a lingua franca for expressing computational experiments [16]. Workflows offer portability, extensibility, reusability, and machine-readability, enabling automated tooling. This success has led to a growing population of workflows and workflow management systems on the web [15].

However, workflows are often irreproducible[1] [19]. They may have always been irreproducible, or they may have initially been reproducible but decayed into irreproducibility later due to changes in computational environments [42]. Science is only self-correcting because scientists can scrutinize and build on each others' work [29], so irreproducibility hinders scientific progress. Scrutiny is hindered when readers need help to re-execute the workflow on their computer, which in turn harms the communal practice of science, and requires researcher to independently re-develop each others' work.

Even outside of basic research, the reproducibility of workflows is essential. Suppose engineers use workflows to simulate the behavior of a physical part. Simulations are rapidly improving, so they may want to rerun a simulation done in the past with newer techniques or with different parameters. The physical part may have a lifetime measured in decades, but the software simulation is much more fragile, lasting only years. If the computation is not reproducible, engineers cannot easily rerun the simulation; they must either attempt time-consuming digital archaeology or rewrite the simulation from scratch.

A roadmap for workflow technologies by Deelman et al. notes an urgent need for innovative approaches, methods, and tools to ensure workflow reproducibility [9]. If archived and made discoverable, workflows could eventually become an enduring resource for the scientific community — enabling researchers to reproduce and build upon each others' work rapidly and credibly.

Current data on the frequency and causes of workflow failures is crucial to building workflow archival and sustainment solutions. A 2012 study by Zhao et al. was among the first to examine such failure causes, specifically among Taverna workflows from the myExperiment workflow registry [42]. Unfortunately, Taverna is no longer actively maintained. The landscape of workflow technologies

[1] We use the ACM definition of reproducibility: a measurement is *reproducible* if a different team can use the same experimental setup to make a concurring observation [36].

has changed significantly, and newer tools have displaced Taverna (see Table 1). In short, several positive developments have happened, and we need a refreshed perspective on workflow reproducibility.

To explore this topic further, we collected workflows from two workflow registries, Snakemake Workflow Catalog [38] and nf-core [14], and attempted to reproduce crash-free executions for them. We address the following research questions:

- **RQ0.** What are the characteristics of workflows and revisions in the selected registries? The answer tells us about the external validity of the study.
- **RQ1.** How many workflows (and for how many revisions of those workflows) in each selected registry are crash-free reproducible? This question quantitatively assesses the level of reproducibility in practice for those registries.
- **RQ2.** For workflows that we were unable to reproduce crash-free executions, what are the most common failure modes? These modes inform future work of workflow engine developers for what to fix, researchers on automatic reproducibility on what to focus on inferring, and workflow users of what to watch out for.
- **RQ3.** What is the survival rate of crash-free reproducibility of workflows over time? While we cannot wait for a specific workflow to break, which may take months or years, we can assume that software in the future will behave similarly to software in the past and make population-level inferences.
- **RQ4.** For crash-free reproductions, how much and what kinds of outputs do they produce? Future research seeking to compare subsequent revisions semantically will need to develop a handler for each kind of output. This research question tells them what kinds of outputs to focus on.

The main differences between our work and prior large-scale studies on automatic reproducibility [32, 39, 41, 41, 42] are:

- We study workflows[2], not arbitrary computational experiments (c.f. [32, 39, 41]). Workflows specifically aim to be reproducible, and the workflows we study containerize each step, for example, so they stand a better chance of being reproduced than arbitrary computational experiments.
- We analyze the "survival rate" of workflows over time. To the best of our knowledge, prior work [39, 42] used time as a categorical rather than a continuous variable (informally, "so many workflows from that year still work") or did not analyze time [32, 41].
- We analyze not only one but two registries and contrast their results. To our knowledge, prior work has not examined the similarities and differences in reproducibility from different workflow registries.

The remainder of this paper is structured as follows. Sections 2 and 3 provide background and related work in curating and sustaining scientific workflows. Section 4 describes our data collection and analysis methodology. Section 5 presents the findings of our study. Section 6 provides a detailed discussion of those findings, including the limitations of our study. Finally, Section 7 summarizes the key results of our study and describes directions for future work.

[2]For our purposes, a workflow language is a programming language where one assembles an explicit dataflow graph, where each node is an existing program.

2 PRELIMINARIES

The Association for Computing Machinery defines *reproducibility* and *replicability* as follows:

Reproducibility means "The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts." [36]

Replicability means "The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently." [36]

Both definitions use "measurement". For our study on reproducing scientific workflows, we define the following as "measurement":

Crash-free execution refers to whether the computational experiment runs to completion without crashing (specifically, terminating with a non-zero exit code for POSIX programs).

While replicable research conclusions are the end goal, assessing that goal in practice requires expert case-by-case analysis. Assessing reproducible crash-free executions, on the other hand, is possible to do automatically and is a vital stepping-stone for replicable research conclusions. If an experiment has a reproducible crash-free execution, the workflow can be scrutinized, extended, and reused in future inquiries.

One salient question about reproducibility is how it relates to time. A computational experiment may be reproducible only up to some point in time but become irreproducible after that point. This change could be due to several reasons. For example, the software environment may not be fully specified, so retrieving the "latest" dependency may stop working at some point. It could also be because the software depends on some network resource that is no longer available. This phenomenon is often called *software collapse* [21] because software with an unstable foundation is analogous to a building with an unstable foundation. Software collapse for workflows manifests itself as irreproducible computational experiments.

3 PRIOR WORK

Prior works on large-scale quantitative reproducibility studies can be split into those whose reproduction is assessed by automatic means versus a manual effort.

Zhao et al. [42] evaluate automatic reproducibility of Taverna workflows from the myExperiment registry. However, Taverna is now defunct, and there have been many changes since 2012 (see Table 1), so we should expect the results to change. Furthermore, Zhao et al. do not examine the correlation of crashes with time or the kinds of outputs when the execution is crash-free.

Trisovic et al. [39] evaluate automatic reproducibility of R code from the Harvard Dataverse repository. While Trisovic et al. propose to study reproducibility based on R version and time (in their RQ8), they treat time as a categorical variable and do not perform a statistical analysis to generalize their data. Furthermore, Trisovic et al.'s reproduction of R code does not include the order in which the

scripts in a single project were originally run, so it incurs failures that may be simply due to a wrong order; our work studies workflows, which avoid the ordering problem because the workflow specifies dependencies between tasks.

Pimentel et al. [32] and Wang et al. [41] automatically run Jupyter Notebooks from GitHub. Jupyter Notebooks have different strengths and use-cases than workflows. Jupyter Notebooks are usually used for small, interactive jobs, whereas workflows are used for large, batch-processing jobs [12, 27]. For example, Snakemake and Nextflow *at the language-level* both provide facilities to run jobs on a cluster. Snakemake and Nextflow, *by default*, write intermediate results to disk so that workflows can be resumed if the node halts or needs to be restarted. While both batch-scheduling submission, crash-recovery, and containerization can be implemented in Python, workflow engines are more specialized for analyzing data at a large scale. Therefore, we expect that the reproducibility characteristics can be quite different. For example, Wang et al. find that using one set of Python packages, namely those in the default Anaconda distribution[3], was sufficient for running their evaluation; in contrast, workflows in Snakemake and Nextflow often provide a distinct set of Python packages *for each task*! Finding the correct set of packages is non-trivial, as we will see in RQ2.

As an example of manual reproduction, Krafczyk et al. execute an in-depth case study on a small set of computational experiments [25]. Stodden et al. [37] perform case studies with specific attention to journal policies. The case-study methodology is helpful for in-depth results but has difficulty generalizing the results to an entire population. Our work attempts an automatic reproduction of a large set of experiments to address population-level questions but does not perform an in-depth analysis of a small subset.

Continuous integration [20] seeks to run tests at every change. However, software can fail not just by changes to the code itself but also by changes to the environment (see "software collapse" above). Continuous integration usually does not seek to cover the case of static code under an evolving environment. Beaulieu-Jones and Greene [4] propose "continuous analysis" to maintain reproducibility. That approach is complementary to ours; future work could combine techniques with our work to continuously evaluate large-scale reproductions.

Provenance is also an important research direction. Pouchard et al. showed how collecting provenance data and performance metrics can aid in confirming the reproducibility of extreme-scale application workflows [33]. Meng and Thain developed a framework for capturing execution environments of workflows at a task-by-task level of granularity [28]. Large-scale reproduction tells provenance researchers where to start looking for examples of working workflows, examples of common errors, and other data. On the other hand, provenance systems improve the reproducibility of workflow engines, which large-scale reproductions can evaluate.

Functional package managers such as Nix and Guix [6, 7] treat building and installing a package as a pure function. To enforce purity, a functional package manager builds packages inside a sandboxed environment that *only* contains the declared inputs. One can use symlinks to link together built aritfacts into a project-specific environment. This approach solves dependency issues but leaves other sources of irreproducibility open, such as applications that access network resources at run-time (at build-time the network is unavailable). A subset of the problems we are studying here would also be problems for Nix and Guix. Guix Workflow Language [40] takes this idea a step further by creating a workflow where each step runs in a Guix-specified environment. While these are promising tools for future development, this work focuses on current, popular workflow engines such as Snakemake and Nextflow to capture an image of reproducibility in the real-world.

Besides research literature, the community has been developing new policies, organizations, and tools to encourage reproducibility (see Table 1).

Table 1: A sample of tools, organizations, and policies regarding reproducibility since 2010.

Year	Kind	Description
2012	Tool	Snakemake paper [27]
2013	Policy	Geoscientific Model Development (GMD) journal requires code sharing [1]
2013	Policy	Office of Science and Technology Policy memorandum (Holden et al.) [22]
2015	Tool	Spack paper [18]
2015	Org	Volume 1 of ReScienceC published [35]
2015	Tool	Guix for HPC paper [7]
2017	Tool	Nextflow paper [12]
2017	Tool	Singularity paper [26]
2017	Tool	Nix for HPC paper [6]
2020	Org	Nextflow community curates nf-core [14]
2022	Policy	Office of Science and Technology Policy memorandum (Nelson et al.) [30]
2022	Policy	NASA Science Mission Directorate Science Policy Document 41a [43]

4 METHODOLOGY

The Workflow Community Initiative[4] lists four registries: Dockstore [31], Snakemake workflow catalog [38] (here on, *"SWC"*), WorkflowHub [17], and nf-core [14]. Dockstore and WorkflowHub contain workflows of many different workflow languages, and they overlap as the same workflow can be in both registries. For this study, we chose SWC and nf-core because they contain only one workflow language each but are still well-populated. The Pegasus Workflow Engine also has a workflow hub called PegasusHub[5], but at the time of this writing, it had only twelve workflows, most of which were examples. Future work could extend our experiment to more workflow registries.

Each entry in SWC and nf-core refers to a specific project on GitHub. These registries are in machine-readable formats.

- The SWC registry includes any project on GitHub that satisfies its requirements, of which the chief is to have a `Snakefile` or `workflows/Snakefile` in the root directory. Users can optionally include a `.snakemake-workflow-catalog.yml`, with a machine-readable description of how to run the workflow.

[3] See https://www.anaconda.com/

[4] https://workflows.community/registries

[5] https://pegasushub.io

- The nf-core registry is a community-curated set of analysis pipelines built using Nextflow. The authors follow the convention of having `main.nf` and `nextflow.config` files in the root directory. Also, `nextflow.config` must define a profile for `test` and `docker`, `singularity`, `podman`, and other virtualization providers. These workflows have many users and contributors.

Snakemake and Nextflow are **interpreters** for domain-specific language that researchers use to write **workflow scripts**. The workflow scripts construct a directed acyclic graph (DAG) of **tasks** based on the input and configuration. The Snakemake and Nextflow languages contain directives to encapsulate each task in a specific **container**.

We use the appropriate workflow engine for each revision of each workflow in the registry. When we run the workflow, we are a different team using the same measuring system (experiment); therefore, we are checking its reproducibility [36]. Testing if the research result is consistent with a specific claim requires data from the original run and expert knowledge, so instead, we just test if the default command with default parameters has a *non-crashing execution*. We run the experiments in a Spack environment (see Appendix A for the exact environment) that has the workflow managers and their dependencies: Snakemake, Nextflow, Conda, Singularity, and others.

We also had to install a few dependencies that workflows assume to exist on the system. Snakemake and Nextflow both allow the workflow to specify a container image to run the tasks in, but they do not provide a way to specify the environment of the program that generates the DAG. In some cases, the environment is determined by convention, but these conventions are neither universally used nor automatically consumed by tools. These dependencies include domain-general processing tools such as Numpy, Pandas, and Peppy. We discovered the exact set of dependencies through trial-and-error. We recognize that the computational experiments may rely on other unspecified dependencies or internet-accessible resources that no longer exist. We expect these to fail, and in fact, this research aims to count how many fail that way.

While most workflows finish within 30 minutes, some can take multiple hours (see Figure 1). In total, we spent over 5,600 CPU hours executing workflows. We use Parsl [3] to run different experiments on a parallel cluster to reduce the waiting time. We used Microsoft Azure to provide the parallel cluster, but Parsl supports a wide range of parallel cluster providers or even a single node. Each worker node runs a specific set of workflow revisions, writes the output to storage, and sends the success indication back to the main node. At the end of execution, we have one or more "execution records" for every revision of every workflow.

We initially ran our setup on a small random sample of revisions. Then we looked through every crash; some were due to the underlying workflow crashing, but some were artificial and caused by our experimental setup. We spent thirty minutes per crash debugging it; if no leads were pointing to our experimental setup after that, we assumed the problem was with the workflow under test. We iteratively improved our experimental setup and repeated all experiments on the latest version.

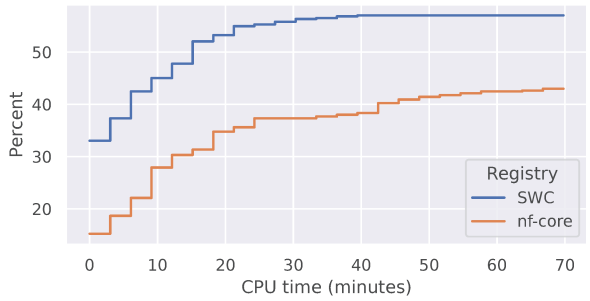

Figure 1: A cumulative frequency histogram of CPU time per revision (crashing and crash-free).

5 RESULTS

See Appendix A for the raw and processed results.

RQ0. We only attempt to reproduce revisions the developers mark as "releases" in GitHub. These revisions are more likely to have worked on the developer's original machine. SWC has 2,045 workflows but only 53 with some revision marked as releases, and nf-core has 48 workflows with revisions marked as releases. For these 101 total workflows for both registries combined, we enumerated every revision marked as a release, yielding 589. The number of revisions per workflow follows a power-law distribution. Most workflows have just 1 revision, but a prolific few have over 20 revisions (see Figure 2). 75% of the SWC workflows have seven or fewer revisions, but one[6] has 61 at the time this paper was written. 75% of the nf-core workflows also have seven or fewer revisions, but the maximum[7] is 25 revisions.

Revisions of nf-core workflows are somewhat older, between 0 and 4.5 years old, while most revisions of SWC workflows are between 0 and 2 years old (see Figure 3). Nf-core officially began in early 2018[8], while SWC only began in late 2020[9], but older revisions are possible. Each registry only holds the URL of the GitHub repository, so a workflow with historical revisions stretching past the inception of the registry can be added to the registry, and our enumeration does include all such older revisions.

RQ0. The selected registries contain 101 workflows combined with 589 revisions, where the distribution of revisions to workflows follows a power-law distribution. The revisions are up to five years old, so **these registries are appropriate for analyzing mid-term reproducibility**.

RQ1. We automatically reproduced crash-free executions for 28% of the total 589 revisions from both registries. The nf-core workflows had a much higher crash-free reproducibility rate, 51%, compared to SWC workflows, 11%. This difference is surprising, considering that revisions of the nf-core workflows are older on

[6] https://github.com/snakemake-workflows/dna-seq-varlociraptor
[7] https://github.com/nf-core/eager
[8] See https://nf-co.re/about
[9] See https://github.com/snakemake/snakemake-workflow-catalog/commit/4d0155429dc2fb75e4916e0e938d02f8b1efcb81

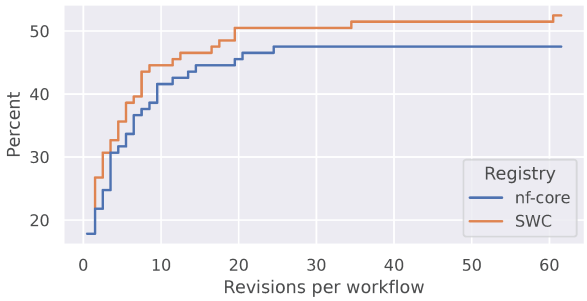

Figure 2: A cumulative frequency histogram of revisions per workflow in selected registries.

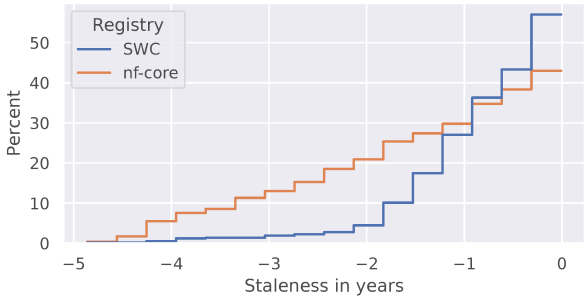

Figure 3: A cumulative frequency histogram of staleness of executions in selected registries.

average (see Figure 3). There are multiple explanations for this difference:

- The nf-core registry is a curated selection of workflows that select for popularly used pipelines.
- Of the workflows that we examined in more detail, the nf-core workflows defer most of their data processing to containerized tasks rather than the main script, which is not containerized. In contrast, SWC workflows process data in the main script *and* containerized tasks. For example, some SWC workflows require BioPython to run the main workflow script.
- The SWC standard does not identify an obvious place for a default or example configuration. In RQ2, we will see that many of these failures are due to missing example data. The nf-core registry requires workflows to have a `test` profile where this information can go.
- Snakemake uses Conda, which can fail for the reasons described later in **RQ2**. This issue affects Snakemake workflows, which use Conda to manage environments, but not nf-core workflows, which are less likely to use Conda. Even excluding these cases from the sample, we get a rate of 13%, which is still significantly less than nf-core.

Quantity	All	SWC	nf-core
# workflows	101	53	48
# revisions	584	333	251
% of revisions with no crash	28%	11%	51%
% of workflows with at least one non-crashing revision	53%	23%	88%

Table 2: Summary of data from automatic reproduction.

Zhao et al. [42] find a 20% of their experiments can be reproduced without crashing. The SWC registry has a lower reproducibility rate, due to the effects above The nf-core registry has a higher reproducibility rate, probably because it is community-curated, whereas Zhao et al. tested workflows in a self-depository called myExperiment. myExperiment is like Zenodo or GitHub in that almost anyone can upload almost anything.

Trisovic et al. [39] reported crash-free reproduction for 25% for R scripts, but they excluded codes which timed out from the denominator. When considering timeouts as failures (the computational experiment might be reproducible but require longer time), the rate falls to 12%, on par with SWC but lower than nf-core. We expect workflows to be more reproducible than a bundle of R scripts because repositories with R scripts may not specify the order to run the R scripts that may have dependencies between them; workflows explicitly encode those dependencies. However, issues similar to the SWC environment are a possible culprit. The scripts studied by Trisovic et al. are from such different domains that the crash-free reproduction rate may differ.

For both registries we study, the crash-free executions do not all come from the same workflows. Namely, workflows are not simply reproducible in either most of their revisions or none; rather, we see a diversity of crash-free reproduction rates across workflows (see Figure 4).

The reproducibility of SWC is biased by many revisions coming from workflows where we could not reproduce any revision, nine of which are due to the same kind of crash. That one cause substantially reduces the number of non-crashing SWC workflow revisions. On the other hand, three of the working workflows have just one revision. The nf-core workflows, being more evenly spread in the number of revisions, do not suffer the same way.

RQ1. In all, we reproduced non-crashing executions for 28% of all revisions of all workflows in our selected registries. Considering the prevalence of irreproducibility, **more work needs to be done on achieving reproducibility with low effort.**

RQ2. For each crashing execution, we examined the log files and standard error to find the low-level cause of the crash. Then we wrote a regular expression that could parse the information for other crashes with the same cause (e.g., if the program crashed because an exception was thrown in a Snakemake script, we wrote a regular expression to parse the traceback). Next, we repeated the process for the first crash not classified by the set of previously written regular expressions. Note that these crashes are only the *earliest* crash present in the code. If we were to fix the immediate

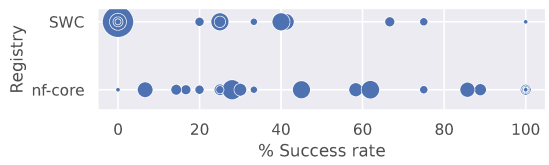

Figure 4: Each workflow is represented by a bubble: its radius is proportional to the number of revisions, its y-position corresponds to its registry, and its x-position is the percent of revisions that were reproducible.

crash, another crash of a different kind may still happen later, for which we have no information. Finally, we putatively categorized the failure cases according to their causes:

- **Missing input**: a crash due to missing data or configuration.
- **Missing dependency**: a crash due to missing a dependency.
- **Network resource changed**: a workflow expects a network resource with a behavior different than current, e.g., the workflow queries a database using an API that has since changed.
- **Timeout reached**: we limited each revision run to 2 hours.
- **Unclassified reason**: not all crash causes can be easily identified automatically. For example, two workflows may fail with the same `IndexError`, but in one case, the code may try to access a configuration file that was not passed (i.e., missing input), and in another case, the same error may be caused by a bug in the script (i.e., workflow script error).
- **Singularity error**: the workflow failed while invoking Singularity. One cause for some of these errors is due to a bug in Singularity[10].
- **Conda environment unsolvable**: Conda can fail to solve an environment for a number of reasons. One observed problem is that Conda cannot manage packages installed by other package managers. We might have a specific version of libc or other packages installed by Spack to support the experiment. If these packages conflict with the packages requested by the environment, then Conda will fail to solve. We count this as a true reproducibility failure because a user may have packages installed that conflict with the Conda environment for a specific project; that project would not be automatically reproducible on that user's system.
- **Other (workflow script)**: the workflow fails for some other reason, and the crash happens within the workflow script, i.e., the program that generates a DAG of tasks.
- **Other (containerized task)**: the workflow fails for some other reason, and the crash happens within one of the containerized tasks, i.e., the nodes of the workflow DAG.

While some of these errors, especially missing inputs, may be easy to fix, there are too many to fix manually. They make the workflows not *automatically* crash-free reproudcible.

These reasons are similar to those by Zhao et al. [42], but we allow for the timeout to be reached and "other." The "other" crashes indicate that the workflow was started correctly, had all necessary inputs, had complete software dependencies, and did not reach a timeout. Therefore, the "other" crashes are probably due to the workflow never working. This cause is consistent with what we find when manually analyzing those cases.

[10]See https://github.com/sylabs/singularity/issues/1721. The bug has since been fixed, but we were not able to re-run all of our experiments due to resource constraints.

Kind of crash	All	SWC	nf-core
Missing input	32.2%	43.8%	16.7%
Conda environment unsolvable	10.8%	18.9%	0.0%
Unclassified reason	7.9%	12.0%	2.4%
Timeout reached	7.0%	5.7%	8.8%
Singularity error	6.0%	6.6%	5.2%
Other (workflow script)	5.7%	1.5%	11.2%
Other (containerized task)	1.2%	0.0%	2.8%
Network resource changed	0.7%	0.0%	1.6%
Missing dependency	0.5%	0.9%	0.0%
No crash	28.1%	10.5%	51.4%
Total	100%	100%	100%

Table 3: Reasons for crashes in revisions we failed to reproduce. These percentages are normalized to the *total* number of executions (crashing and crash-free).

RQ2. Among workflows that crashed, the leading cause of crashes was missing input data or configuration files. Missing is more prevalent in SWC because **`.snakemake-workflow-catalog.yaml` has no place to specify an example invocation**.

RQ3. While we cannot easily look back in time to see when any individual revision stopped working, we can instead reason about the aggregate population of revisions. We assume that the probability that a workflow revision published some time ago (e.g., two years ago) works is a good estimate for the probability that a workflow revision published today will work in the same amount of time (e.g., two years). We attempt to find a trend between crash-free reproduction rate and "staleness," the difference in time between when the revision was published and when we execute it in 2023.

We make the simplifying assumption that all revisions of all workflows in our selected registries were automatically reproducible when that revision was initially uploaded. The assumption may not hold for arbitrary workflow revisions, but our focus is precisely on the revisions that were marked as releases because such releases are highly likely to have worked for the original developers. With each passing day, a change may cause a workflow to break. We also assume that, as workflows age, the risk that they will break increases over time. We fitted a Weibull survival function on our data to model this behavior. The following parameterized formula describes a Weibull probability distribution:

$$f_X(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{k}\right)^{k-1} \exp\left(-(x/\lambda)^k\right) & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (1)$$

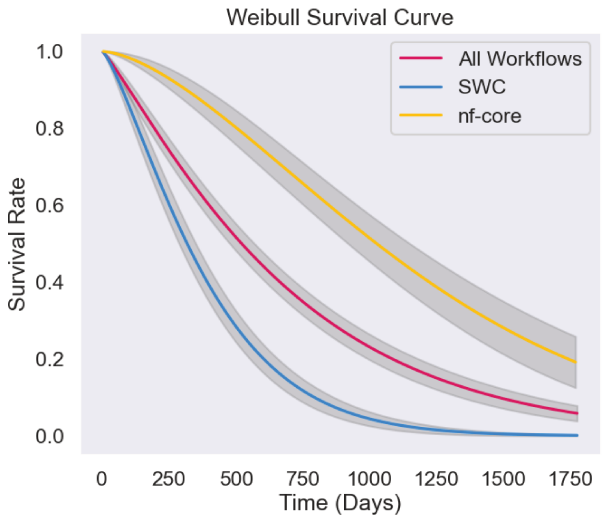

Figure 5: Estimates for expected workflow reproducibility over time modeled using a Weibull decay function. The estimated median survival time across all workflows in either registry is 1.42 years, which means half of the workflows remain crash-free reproducible after that. As the graph indicates, however, the nf-core workflows are much longer-lived (median 2.81 years) than the SWC workflows (median 0.81 years).

	λ (95% CI)	k (95% CI)	Med. Surv. Time
All	717 (657–777)	1.15 (1.06–1.24)	521d
SWC	420 (382–458)	1.31 (1.20–1.43)	317d
nf-core	1292 (1144–1439)	1.59 (1.36–1.82)	1026d

Table 4: Parameters used in Weibull estimates in Figure 5.

The parameter x represents the time-to-failure for a workflow. The parameter k is the shape parameter; a value $k < 1$ means that the failure rate decreases over time, $k = 1$ means that the failure rate is constant, and $k > 1$ means that the failure rate increases over time. Finally, λ is a scale parameter; it can be interpreted as how much time must pass until 63.2% of workflows have failed. Using the Python package *lifelines* (version 0.27.4), we fit curves on our workflow reproducibility data for SWC workflows, nf-core workflows, and all workflows taken together. We provide a summary table of the curves in Figure 5.

The difference in outcomes between the SWC and nf-core workflows calls attention to characteristics we are not directly measuring. The nf-core registry is a carefully curated, community-driven effort to build and sustain nf-core genomics workflows, and most of the failing cases are old revisions that are no longer officially supported. Meanwhile, the SWC workflows are drawn from a much larger corpus across GitHub. The fact that curated revisions of workflows survive three times longer than those in the wild is not surprising, but it does show that a sharp decline in reproducibility is not inevitable.

Type	All	SWC	nf-core
ASCII text	85%	33%	100%
HTML document	59%	0%	76%
SVG image	26%	0%	33%
Zip archive data	7%	0%	10%
XML 1.0 document	4%	0%	5%
CSV text	19%	0%	24%
JSON text data	15%	0%	19%
very short file (no magic)	4%	0%	5%
gzip compressed data	11%	0%	14%
PDF document	7%	0%	10%
Blocked GNU Zip Format	7%	17%	5%
PNG image data	4%	0%	5%
LaTeX 2e document	4%	0%	5%
Total	27	6	21

Table 5: Each row shows what proportion of workflows with multiple revisions with crash-free executions have a *common output* of that file type. Note that these need not add to 100%; one workflow might have a common output of ASCII text and a common output of PNG images; this would increment the count in both rows.

RQ3. Aging workflows are more likely, all things being equal, to crash. However, different populations of workflows (such as those drawn from different registries) can **decay at different rates**. Moreover, **biological survival analysis is a useful tool** to study software collapse and plan ahead for it.

RQ4. This work examines just the reproducibility of crash-free executions. While full reproducibility of research results requires expert knowledge, some intermediate levels of reproducibility can be automatically assessed. An automated tool might look at the outputs produced and compute their variation if they have the same structure. "Structure" here refers to both the location of the files (e.g., the directory and filename of specific output datasets) and the content within the files (e.g., the order of columns in CSV). Such a tool would need to know what types of files are common outputs between the two executions.

We call a file path relative to the experiment's working directory an **output** if it does not match a list of known intermediate outputs, log files, or temporary data. This notion is biased towards assuming a file is an output because that is the default for workflows. The list of known non-output files includes `work/`, `pipeline_info`, `.snakemake` and `logs` for Snakemake, and `.nextflow.log` and `.nextflow` for Nextflow. We also added directives to store the Singularity container file systems and Conda environments in separate file paths, so they would not be considered outputs. We call an output **common** to a workflow if the workflow has more than one revision with crash-free executions and the output is present with the same file type in at least two-thirds of the executions. We used `file`[11], a "file type guesser," to guess the file type.

[11] See https://www.darwinsys.com/file. We used `file -brief`.

The result (Table 5) shows that when there are common outputs, at least one of them is usually ASCII text with no "higher level" structure like CSV or JSON. Only 25% of the nf-core workflows can be compared through CSV.

If one cannot deduce any structure of plain text files, the only choice may be to treat them as strings or a list of strings (for each line). Future work may investigate methods for quantifying the difference between ASCII text files; e.g., one could use the edit distance at a line-level (similar to what `diff` does). If there are a lot of sub-line changes, one might use edit-distance on characters, although this comparison would be time-consuming.

RQ4. The most common output across revisions of a workflow is usually unstructured text.

6 DISCUSSION

6.1 Lessons learned for reproducers

Superuser is required to reproduce normal workflows!

Big supercomputers are often shared among a whole department or multiple institutions, so most users do not have superuser access. Ideally, one should not need superuser access just to reproduce someone else's computational experiment. However, some functionality in container engines currently requires superuser access, which did affect this work. While a normal user *can* install Singularity and its successor Apptainer, that installation do not support full set of features and experience worse filesystem performance than when installed by a superuser [11, 13] in "setuid mode." We noticed several failures due to inability to mount the right paths in a Singularity container, which we fixed by installing Singularity as superuser. While one does not need to run the workflow engine itself as a superuser, it calls Singularity, which calls a setuid binary that escalates into superuser privileges, so a superuser has to install that binary. Also note that setuid Singularity cannot nest within another Singularity (setuid or not) [13], so we had to run our experiment on bare-metal such that the workflow engine could start a setuid Singularity container.

Ongoing developments in the "unpriveleged user namespace" feature of the Linux Kernel open the door to container engines that do not run as root, but old versions of the kernel hinder this use-case. For example, CentOS 6 uses the 2.6 Linux Kernel, but user namespaces were not available until 3.8 or later [10]. Even in later kernels, user namespaces may be disabled. Enabling user namespaces opens a much larger surface for attacks (e.g., see CVE-2020-14386 in Linux Kernel 5.9 [8]), so many security standards recommend disabling them [2]. Still, Linux developers are making progress in securing user namespaces, and old supercomputers are being retired, so eventually, reproducibility can be improved through unpriveleged user namespaces. Given a recent enough kernel, Charliecloud [34] provides precisely that. However, Snakemake has yet to integrate with Charliecloud (see ongoing issue[12]). Future work could quantify how the choice of container engine and root-user privilege changes the non-crashing reproducibility rate.

[12] https://github.com/snakemake/snakemake/issues/44

Continuous integration (CI) scripts do not help much.

Often, a human could glean how to run a computational experiment given the CI scripts. However, selecting the right target is difficult to automate because the CI scripts contain instructions for many different goals besides the goal of testing the software. When looking at GitHub Action scripts in SWC, we found scripts that lint, generate reports (without running), and test the Conda environment; these would have to be excluded by automatic reproducibility software. If the computational experiment has a rather long running time, users will exclude it from CI testing, so we have no guarantee that any CI action actually tests the code.

There can be more than one way to test the workflow.

The CI discussion also raises another point: what should the "test configuration" be? Should it be a scaled-down execution or a full-fledged one? What if the experiment supports multiple different modes; which should be used? In practice, the nf-core repositories specify one configuration as the "default" test configuration, but they often contain multiple `test_*` configurations, providing for test variants. An open ontology could describe what knobs to turn in each test. Such configurability would open the door to many automatic testing applications, such as autotuning configuration parameters, outcome-preserving input reduction, or other kinds of parameter searching if the system knows what knobs it can turn without breaking the experiment's semantics.

6.2 Recommendations for workflow engine designers

The presence and rigor of community standards greatly affect reproducibility.

The nf-core repositories usually have a configuration profile in the root called `test` that runs whatever the workflow author defines as a test. Other tools choose conventions to make their tools easier to use (e.g., `make all`). Other Nextflow workflows outside of nf-core do not usually follow this convention[13], so it would be much harder to test them automatically.

SWC does have a similar convention, but it is not rigorous enough. While SWC workflows have a place for "mandatory flags" in the `.snakemake-workflow-catalog.yml`, there is no place for an example invocation[14]. As such, many of the workflows fail because our default command does not provide them with any example data.

We should use metadata to link the publication, funding, and authors to the workflow.

We could not find a machine-readable link between the workflow and the publication, funding, and authors. Linking the workflow would allow us to study the impact of policies on reproducibility. Git stores a history of the authors who touch the code, but these do

[13] For an example, see https://github.com/marcodelapierre/toy-gpu-nf

[14] See https://github.com/snakemake-workflows/dna-seq-varlociraptor/pull/204 for discussion

not include all collaborators (e.g., a university professors when the student does the work) or other facilitators. Transitive Credit aims to solve this problem with JSON-LD [23], but it is not yet widely used.

> Workflow engines should report resource requirements in a machine-readable format.

Each of these different test variants may have different resource requirements too. In batch compute systems, such as supercomputers, on which many computational scientists work, the users request a compute resource allocation (such as the number of CPUs, GPUs, peak disk utilization, peak memory utilization, or total time). In practice, the users guess the request using rules-of-thumb; if the guess is wrong, their job may fail, and they will have to retry with a larger resource request. While not strictly necessary for reproducibility, it may be easier if the original authors publish the resources needed to run their experiment. Modern retrospective provenance systems [5, 24] do not yet provide a way of capturing or storing this information, although it would be straightforward to add. Knowing at least the total time the computational experiment takes helps future users to know if the run got "stuck" in a deadlock or infinite loop. We **do** report resource utilization requirements for the workflows in our dataset, which can be found in Appendix A.

> Opaque container images may be reproducible but are not ideal.

Workflow programs supply a container image for each step of their execution. This gives a high level of reproducibility for the task which runs in the container, but the dozens of container images used in an experiment become another digital artifact which need to either be archived (heavy storage cost) or reproducibly built (pushes the buck to another tool). Functional package managers have the potential to fill this gap by either building container images reproducibly or managing the environment natively for each step in the experiment.

6.3 Threats to Validity

The workflows we selected may not be representative of all workflows. We worked with two large registries and ran every workflow in each registry uniformly, but there may be a selectivity bias for workflows submitted to workflow registries. Still, problems for the community's most publicized workflows are likely also problems for the other workflows.

We only test for reproducible crash-free execution. We cannot test research reproducibility because we do not have access to the original results, and we would need the expertise to compare results from two runs to see if they are equivalent. However, reproducible crash-free execution is a necessary condition for reproducible research results, and right now, only 51% of nf-core workflows and 11% of SWC workflows have even crash-free reproducibility.

Our system's packages may also be conflicting with the packages that the experiment wants to install. While the exact symptom is specific to our system, the pathology is a problem for reproducibility more generally.

The workflows we test may be reproducible, but our automated system could not figure out the correct command to run. These would be manually reproducible but not automatically reproducible. However, automatic reproducibility confers benefits that manual reproducibility does not. For example, automatic reproducibility makes it easy to set up continuous integration.

On the other hand, one might argue that in designing our automated system, we encoded too much information discovered from manually debugging failed workflows. For example, we found that the SWC workflows often require Peppy and Pandas, just to subselect the data for input to the tasks. Because it is reasonable to expect these packages might be installed on the user's machine, we added them into our software environment. One might argue that experiments which depend on a package without declaring a dependency on that package should be marked as not automatically reproducible; we considered this position, but then so many workflows would be not reproducible that we would not have much data left to work with for the rest of the RQs.

7 CONCLUSION

Reproducibility allows science to be self-correcting and helps us build on each other's results. While it intuitively seems that computational experiments should be perfectly reproducible, especially compared to bench work, computational experiments are often the root of irreproducible research.

In this work, we investigate how reproducible workflows are in practice by looking at workflows from two specific registries, SWC and nf-core. The fact that our experiment on reproducibility is possible is a testament to the improvements in tooling and community practices. The nf-core registry could be used an example of how communities standardize around standard conventions and tooling. However, the current practice needs to be improved for a higher degree of reproducibility. In particular, workflow authors should incorporate example data that runs "out of the box." More work needs to be done on standardizing how to specify the means to reproduce a computational experiment.

A CODE & DATA AVAILABILITY

A snapshot of the latest state of this code can be found at: https://doi.org/10.5281/zenodo.7996835.

A rolling release of the code can be found at: https://github.com/charmoniumQ/wf-reg-test.

In the rolling release or snapshot:

- `data` holds a machine-readable view of the data, split across several files.
- `data/results.html` is a human-readable HTML view of the data.
- `REPRODUCING.md` contains instructions on how to reproduce the results in this paper from various steps.
- `spack/spack.lock` contains the Spack environment in which this experiment was run.

ACKNOWLEDGMENTS

This work was partially supported by NSF grants CCF-1763788 and CCF-1956374. We acknowledge support for research on flaky tests from Google and Meta.

REFERENCES

[1] J. Annan, D. Hargreaves, D. Lunt, A. Ridgwell, I. Rutt, and R. Sander. 2013. Editorial: The publication of geoscientific model developments v1.0. *Geoscientific Model Development* 6, 4 (Aug. 2013), 1233–1242. https://doi.org/10.5194/gmd-6-1233-2013 Publisher: Copernicus GmbH.

[2] STIG Authors. 2020. Red Hat Enterprise Linux 8 Security Technical Implementation Guide. https://www.stigviewer.com/stig/red_hat_enterprise_linux_8/2020-11-25/

[3] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M. Wozniak, Ian Foster, Michael Wilde, and Kyle Chard. 2019. Parsl: Pervasive Parallel Programming in Python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19)*. Association for Computing Machinery, New York, NY, USA, 25–36. https://doi.org/10.1145/3307681.3325400

[4] Brett K. Beaulieu-Jones and Casey S. Greene. 2017. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology* 35, 4 (April 2017), 342–346. https://doi.org/10.1038/nbt.3780 Number: 4 Publisher: Nature Publishing Group.

[5] Anila Sahar Butt and Peter Fitch. 2020. ProvONE+: A Provenance Model for Scientific Workflows. In *Web Information Systems Engineering – WISE 2020 (Lecture Notes in Computer Science)*, Zhisheng Huang, Wouter Beek, Hua Wang, Rui Zhou, and Yanchun Zhang (Eds.). Springer International Publishing, Cham, 431–444. https://doi.org/10.1007/978-3-030-62008-0_30

[6] Bruno Bzeznik, Oliver Henriot, Valentin Reis, Olivier Richard, and Laure Tavard. 2017. Nix as HPC package management system. In *Proceedings of the Fourth International Workshop on HPC User Support Tools (HUST'17)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3152493.3152556 interest: 99.

[7] Ludovic Courtès and Ricardo Wurmus. 2015. Reproducible and User-Controlled Software Environments in HPC with Guix. In *Euro-Par 2015: Parallel Processing Workshops (Lecture Notes in Computer Science)*, Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander (Eds.). Springer International Publishing, Cham, 579–591. https://doi.org/10.1007/978-3-319-27308-2_47 interest: 99.

[8] CVE database. 2020. CVE - CVE-2020-14386. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-14386

[9] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. 2018. The future of scientific workflows. *The International Journal of High Performance Computing Applications* 32, 1 (Jan. 2018), 159–175. https://doi.org/10.1177/1094342017704893 Publisher: SAGE Publications Ltd STM.

[10] Linux Developers. 2021. user_namespaces(7) - Linux manual page. https://www.man7.org/linux/man-pages/man7/user_namespaces.7.html

[11] Singularity Developers. 2023. Security in SingularityCE — SingularityCE Admin Guide 3.11 documentation. https://docs.sylabs.io/guides/latest/admin-guide/security.html

[12] Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature Biotechnology* 35, 4 (April 2017), 316–319. https://doi.org/10.1038/nbt.3820 Number: 4 Publisher: Nature Publishing Group.

[13] Dave Dykstra. 2022. Apptainer Without Setuid. https://doi.org/10.48550/arXiv.2208.12106 arXiv:2208.12106 [cs].

[14] Philip A. Ewels, Alexander Peltzer, Sven Fillinger, Harshil Patel, Johannes Alneberg, Andreas Wilm, Maxime Ulysse Garcia, Paolo Di Tommaso, and Sven Nahnsen. 2020. The nf-core framework for community-curated bioinformatics pipelines. *Nature Biotechnology* 38, 3 (March 2020), 276–278. https://doi.org/10.1038/s41587-020-0439-x Number: 3 Publisher: Nature Publishing Group.

[15] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, Bartosz Balis, Taina Coleman, Frederik Coppens, Frank Di Natale, Bjoern Enders, Thomas Fahringer, Rosa Filgueira, Grigori Fursin, Daniel Garijo, Carole Goble, Dorran Howell, Shantenu Jha, Daniel S. Katz, Daniel Laney, Ulf Leser, Maciej Malawski, Kshitij Mehta, Loic Pottier, Jonathan Ozik, J. Luc Peterson, Lavanya Ramakrishnan, Stian Soiland-Reyes, Douglas Thain, and Matthew Wolf. 2021. A Community Roadmap for Scientific Workflows Research and Development. In *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, St. Louis, MO, USA, 81–90. https://doi.org/10.1109/WORKS54523.2021.00016 arXiv:2110.02168 [cs] interest: 90.

[16] Rafael Ferreira da Silva, Kyle Chard, Henri Casanova, Dan Laney, Dong Ahn, Shantenu Jha, William E. Allcock, Gregory Bauer, Dmitry Duplyakin, Bjoern Enders, Todd M. Heer, Eric Lançon, Sergiu Sanielevici, and Kevin Sayers. 2021. *Workflows Community Summit: Tightening the Integration between Computing Facilities and Scientific Workflows.* Technical Report ORNL/TM-2022/1832. Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States). https://doi.org/10.2172/1842590

[17] Rafael Ferreira da Silva, Loïc Pottier, Tainã Coleman, Ewa Deelman, and Henri Casanova. 2020. WorkflowHub: Community Framework for Enabling Scientific Workflow Research and Development. In *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, Georgia, USA, 49–56. https://doi.org/10.1109/WORKS51914.2020.00012

[18] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. 2015. The Spack package manager: bringing order to HPC software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2807591.2807623 interest: 80.

[19] José Manuel Gómez-Pérez, Esteban García-Cuesta, Aleix Garrido, José Enrique Ruiz, Jun Zhao, and Graham Klyne. 2013. When History Matters - Assessing Reliability for the Reuse of Scientific Workflows. In *The Semantic Web – ISWC 2013 (Lecture Notes in Computer Science)*, Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz (Eds.). Springer, Berlin, Heidelberg, 81–97. https://doi.org/10.1007/978-3-642-41338-4_6

[20] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE '16)*. Association for Computing Machinery, New York, NY, USA, 426–437. https://doi.org/10.1145/2970276.2970358

[21] Konrad Hinsen. 2019. Dealing With Software Collapse. *Computing in Science & Engineering* 21, 3 (May 2019), 104–108. https://doi.org/10.1109/MCSE.2019.2900945 Conference Name: Computing in Science & Engineering.

[22] John P. Holden. 2013. Increasing Access to the Results of Federally Funded Scientific Research.

[23] Daniel S. Katz and Arfon M. Smith. 2015. Transitive Credit and JSON-LD. *Journal of Open Research Software* 3, 1 (Nov. 2015), e7. https://doi.org/10.5334/jors.by Number: 1 Publisher: Ubiquity Press.

[24] Farah Zaib Khan, Stian Soiland-Reyes, Richard O Sinnott, Andrew Lonie, Carole Goble, and Michael R Crusoe. 2019. Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv. *GigaScience* 8, 11 (Nov. 2019), giz095. https://doi.org/10.1093/gigascience/giz095 interest: 98.

[25] Matthew S. Krafczyk, August Shi, Adhithya Bhaskar, Darko Marinov, and Victoria Stodden. 2021. Learning from reproducing computational results: introducing three principles and the Reproduction Package. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379, 2197 (March 2021), 20200069. https://doi.org/10.1098/rsta.2020.0069 Publisher: Royal Society.

[26] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PLOS ONE* 12, 5 (May 2017), e0177459. https://doi.org/10.1371/journal.pone.0177459 Publisher: Public Library of Science.

[27] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (Oct. 2012), 2520–2522. https://doi.org/10.1093/bioinformatics/bts480

[28] Haiyan Meng and Douglas Thain. 2017. Facilitating the Reproducibility of Scientific Workflows with Execution Environment Specifications. *Procedia Computer Science* 108 (Jan. 2017), 705–714. https://doi.org/10.1016/j.procs.2017.05.116

[29] Robert K. Merton. 1974. *The sociology of science: theoretical and empirical investigations* (4. dr. ed.). Univ. of Chicago Pr, Chicago.

[30] Alondra Nelson. 2022. Ensuring Free, Immediate, and Equitable Access to Federally Funded Research.

[31] Brian D. O'Connor, Denis Yuen, Vincent Chung, Andrew G. Duncan, Xiang Kun Liu, Janice Patricia, Benedict Paten, Lincoln Stein, and Vincent Ferretti. 2017. The Dockstore: enabling modular, community-focused sharing of Docker-based genomics tools and workflows. *F1000Research* 6 (Jan. 2017), 52. https://doi.org/10.12688/f1000research.10137.1

[32] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR '19)*. IEEE Press, Montreal, Quebec, Canada, 507–517. https://doi.org/10.1109/MSR.2019.00077 ISSN: 2574-3864.

[33] Line Pouchard, Sterling Baldwin, Todd Elsethagen, Shantenu Jha, Bibi Raju, Eric Stephan, Li Tang, and Kerstin Kleese Van Dam. 2019. Computational reproducibility of scientific workflows at extreme scales. *International Journal of High Performance Computing Applications* 33, 5 (April 2019), 763–776. https://doi.org/10.1177/1094342019839124 Institution: Brookhaven National Lab. (BNL), Upton, NY, Number: BNL-211854-2019-JAAM Publisher: SAGE.

[34] Reid Priedhorsky and Tim Randles. 2017. Charliecloud: unprivileged containers for user-defined software stacks in HPC. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Denver Colorado, 1–10. https://doi.org/10.1145/3126908.3126925

[35] Nicolas P. Rougier and Konrad Hinsen. 2019. ReScience C: A Journal for Reproducible Replications in Computational Science. In *Reproducible Research in Pattern Recognition (Lecture Notes in Computer Science)*, Bertrand Kerautret, Miguel

Colom, Daniel Lopresti, Pascal Monasse, and Hugues Talbot (Eds.). Springer International Publishing, Cham, 150–156. https://doi.org/10.1007/978-3-030-23987-9_14

[36] ACM Inc. staff. 2020. Artifact Review and Badging. https://www.acm.org/publications/policies/artifact-review-and-badging-current

[37] Victoria Stodden and Sheila Miguez. 2014. Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. *Journal of Open Research Software* 2, 1 (July 2014), e21. https://doi.org/10.5334/jors.ay Number: 1 Publisher: Ubiquity Press.

[38] The Snakemake Team. 2023. https://snakemake.github.io/snakemake-workflow-catalog.

[39] Ana Trisovic, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. 2022. A large-scale study on research code quality and execution. *Scientific Data* 9, 1 (Feb. 2022), 60. https://doi.org/10.1038/s41597-022-01143-6 Number: 1 Publisher: Nature Publishing Group.

[40] Nicolas Vallet, David Michonneau, and Simon Tournier. 2022. Toward practical transparent verifiable and long-term reproducible research using Guix. *Scientific Data* 9, 1 (Oct. 2022), 597. https://doi.org/10.1038/s41597-022-01720-9 interest: 99 Number: 1 Publisher: Nature Publishing Group.

[41] Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. 2021. Assessing and restoring reproducibility of Jupyter notebooks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20)*. Association for Computing Machinery, New York, NY, USA, 138–149. https://doi.org/10.1145/3324884.3416585

[42] Jun Zhao, Jose-Manuel Gomez-Perez, Khalid Belhajjame, Graham Klyne, Esteban Garcia-cuesta, Aleix Garrido, Kristina Hettne, Marco Roos, David De Roure, and Carole Goble. 2012. Why workflows break — understanding and combating decay in Taverna workflows. In *2012 IEEE 8th International Conference on E-Science (e-Science)*. IEEE, Chicago, IL, 9. https://doi.org/10.1109/eScience.2012.6404482

[43] Thomas H. Zurbuchen. 2022. SMD Policy Document SPD-41a.