Provenance and the Different Flavors of Computational Reproducibility

Juliana Freire
New York University
juliana.freire@nyu.edu

Fernando Chirigati
New York University
fchirigati@nyu.edu

Abstract

While reproducibility has been a requirement in natural sciences for centuries, computational experiments have not followed the same standard. Often, there is insufficient information to reproduce computational results described in publications, and in the recent past, this has led to many retractions. Although scientists are aware of the numerous benefits of reproducibility, the perceived amount of work to make results reproducible is a significant disincentive. Fortunately, much of the information needed to reproduce an experiment can be obtained by systematically capturing its provenance. In this paper, we give an overview of different types of provenance and how they can be used to support reproducibility. We also describe a representative set of provenance tools and approaches that make it easy to create reproducible experiments.

1 Introduction

The need to reproduce experiments to verify and extend them is not new in science. Revisiting and reusing past results – or as Newton once said, "standing on the shoulders of giants" – is the standard paradigm of all sciences. Unfortunately, achieving reproducibility has proved elusive for computational experiments, which, due to the explosion in the volume of available data and widely accessible computing infrastructure, have become an integral component of science in many different domains.

Scientific papers published in conferences and journals present a large number of tables, plots, and beautiful pictures that summarize the obtained results, but that loosely describe the steps taken to derive them [16,38]. Not only can the methods and the implementation be complex, but their configuration may require setting myriad parameters. Consequently, reproducing the results from scratch is both time-consuming and error-prone at best, and sometimes impossible.

Reproducibility of computational experiments across platforms and time brings a range of benefits to science. First, reproducibility enables reviewers to test the outcomes presented in papers. This is specially important given the growing concern that many spurious research findings are published in respected venues [5,12,29], which is reflected in the increasing number of paper retractions [44, 58]. Second, it allows new methods to be objectively compared against methods presented in reproducible publications. Third, researchers are able to build on top of previous work directly. Last but not least, recent studies indicate that reproducibility increases impact, visibility, and research quality [3, 7, 26, 36, 50, 59] and helps defeat self-deception [46].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

While many scientists recognize the importance of reproducibility, they are often held back by the complexities involved in putting it into practice. They must describe and encapsulate the entire experiment, i.e., the set of steps followed to obtain a result, including data, parameters, source code, dependencies, and environment, so that the results can be properly verified and analyzed. If the experiment is not systematically documented and made reproducible from the start, it may be difficult and time-consuming to retrospectively track all the necessary components, and important aspects may be mistakenly omitted. As an example, some numerical models, if not fully described, may lead to different implementations that are mathematically equivalent, but numerically disparate, thus hampering reproducibility [14]. Even when a complete description is available, others may have difficulties to reproduce the results: there may be no instructions about how to execute the code and explore it further; the experiment may not run on a certain operating system; there may be missing libraries; library versions may be different; and several issues may arise while trying to install all the required chains of dependencies.

Fortunately, much of the information needed to reproduce an experiment can be obtained by capturing its *provenance*. For a given experiment, computational provenance provides information about how and where that experiment was carried out, what input data was used, and which outputs were produced. In other words, provenance *embodies* the association between computation and results [17]: it helps determine both the data and the sequence of steps that generated the findings, which is essential to make results reproducible. Note, however, that different *types of provenance* can be captured, and these enable different *levels of reproducibility*. For instance, some systems, such as VisTrails [19] and Kepler [39], capture workflow provenance, i.e., the dataflow of an experiment, including its main programs and all the data used and derived. But these systems do not detect the library dependencies for the different programs. As a result, they only attain reproducibility given that the computational environment is unchanged (e.g., machine and version of dependencies are the same). Other systems, such as ReproZip [51], are able to capture provenance at the operating system level, including library dependencies, which allows experiments to be reproduced even in different machines. Nonetheless, different from workflow systems, ReproZip may not capture the dataflow in an human-interpretable format, which may make it harder to extend and modify the original dataflow for other purposes.

In this paper, we provide an overview of the different types of provenance and how they influence reproducibility. Our main goal is to help researchers understand how the different types of provenance affect the level of reproducibility and their corresponding trade-offs, to guide them in selecting a suitable approach for their experiments. In Section 2, we start by defining *computational reproducibility*, and use this definition to detail the different levels of reproducibility one can achieve. Next, we present all the different types of provenance and how they related to reproducibility in Section 3, where we also describe tools that can be used to capture these different types. We conclude in Section 4, where we discuss challenges and open problems.

2 What is Reproducibility?

There are different definitions for the term *reproducibility*, which have been used inconsistently across scientific disciplines [2]. In this paper, we focus on the *computational aspects* of reproducibility and introduce definitions that capture the necessary components to implement computational reproducibility in practice.

2.1 Computational Reproducibility

To understand what is needed to reproduce an experiment, we must first define what a *scientific experiment* is. We borrow the definition from Rokem and Chirigati [52]:

Definition 1 (Scientific Experiment): A scientific experiment, or simply experiment, is any procedure carried out to validate or refute a hypothesis, which involves the use of computational assets, including computer programs and digital data that is consumed (input data) and produced (output data).

Such procedure can be represented as a *dataflow*: a sequence of steps that are connected by the flow of data, where the output data of a step is used as input data for the following step. In this case, a step is represented by a computer program or a sequence of programs, and it transforms the data it consumes as part of the procedure. Below, we define what it means for an experiment to be reproducible [17].

Definition 2 (Reproducible Experiment): An experiment composed by a sequence of steps S that has been developed at time T, on environment (hardware and operating system) E, and on data D is *reproducible* if it can be executed with a sequence of steps S' (different or the same as S) at time $T' \geq T$, on environment E' (different or the same as E), and on data D' (different or the same as D) with consistent results.

Consistency here implies that the results can validate the original claims of the research, i.e., the same conclusions derived from the original results can be obtained from the new results. Clearly, this depends on the problem being addressed and the overall goal. For instance, it is unlikely that an experiment that compares the performance of two database systems will produce the exact running times across runs, in particular, if different machines are used (i.e., $E' \neq E$). However, if the new numbers reflect the same trends originally observed and validated, then we say that the results are consistent, and therefore, reproducible.

Note that our definition includes both *exact reproducibility* and *approximate reproducibility* [52]. Exact reproducibility, also known as *repeatability*, entails reproducing the exact same results (meaning, the exact same numbers) as originally published, which requires having $S' = S$, $E' = E$, and $D' = D$. Approximate reproducibility, on the other hand, involves producing results that are similar to (but not that same as) the original ones, which entails having different data, a modified sequence of steps, a different environment, or a combination of these. In this case, the reproduced results may not necessarily be the same, but they must be consistent with the original results. For experiments that are intrinsically difficult to replicate (e.g., experiments that require a specific hardware), guaranteeing approximate reproducibility is essential.

The PRIMAD Model. The PRIMAD model [18] extends our definition of reproducibility to include non-computational aspects as well. It provides a flexible model to define reproducibility: different levels (or modes) of reproducibility can be defined by modifying a set of variables while reproducing an experiment. We consider the following variables:

- **(P) Platform:** the computational environment, including operating system, hardware architecture, and library dependencies.
- **(R) Research Objectives:** the main goal or purpose of the experiment, i.e., the problem that the experiment is trying to address.
- **(I) Implementation:** how the experiment and its corresponding sequence of steps is implemented (e.g., source code and binaries).
- **(M) Methods:** the methods and algorithms that the experiment implements and uses to achieve the research goals.
- **(A) Actors:** the main users of the experiment.
- **(D) Data:** the input data files, intermediate data, and parameters for the experiment.

These variables are used to describe which aspects of the experiment can be changed while still attaining reproducible results. The conditions under which the experiment is reproducible are defined by qualifying the different variables: we tag a variable X with the prime symbol to indicate that X can be changed and the experiment is still reproducible. If untagged variables are changed, reproducibility cannot be guaranteed. For instance, if an experiment is claimed to be $P'RIMA'D'$, it means that, if researchers, who are not the original

Level	Data		Platform	Implementation	
	Parameters	Data Files		Binaries	Source Code
Repeatable	–	–	–	–	–
Re-runnable	X	X	–	–	–
Portable	–	–	X	–	–
Extendable	–	–	–	X	X
Modifiable	–	–	–	–	X

Table 2: Levels of reproducibility based on the different variables. The symbol "X" denotes a change in the corresponding variable, while "–" denotes no change from the original setting. Adapted from [18].

authors of the experiment (A'), reproduce the experiment with different data (D') on a different platform (P'), they will obtain results that are consistent with the ones originally published.

Note that P, I, and D are equivalent to E, S, and D, respectively, from Definition 2. The other variables from the PRIMAD model correspond to the non-computational aspects of reproducibility. While we do not consider these aspects in this paper, we note that they are vastly important and useful to complement the computational components.

Additional Dimensions of Reproducibility. Besides the PRIMAD variables, there are other dimensions that are important for qualifying the level of reproducibility of an experiment. One of these dimensions is *coverage* [17], which takes into account how much of the experiment can be reproduced, i.e., if the experiment can be partially or fully reproduced. Many experiments cannot be fully reproduced, e.g., experiments that rely on data derived by third-party Web services or special hardware. But such experiments can, sometimes, be partially reproduced. For example, if an experiment uses data that is derived by special, proprietary hardware, the data derivation may not be reproducible. However, the downstream analyses that use these data may be reproduced by others if the data is made available.

Another important dimension is *transparency*, which considers *how much information* is made available for an experiment. This dimension affects the variables in different ways. For instance, in terms of implementation, one can provide the original binaries used in the experiment. While this allows the results to be reproduced, it limits reusability: these binaries can only be executed in a compatible platform, and the implementation cannot be modified. On the other hand, if the source code is provided, the implementation can be better inspected and reused. In terms of data, if the data cleaning process is made available, in addition to the final, cleaned input files, it is easier to clean other datasets and explore how the experiment behaves with these.

Finally, *longevity* relates to the ability to reproduce experiments (long) after they were created. Supporting longevity is challenging because software environments evolve, i.e., libraries, operating systems, and data used change over time.

2.2 Levels of Reproducibility

To assess the reproducibility of an experiment, we need to understand which of its different components are made available. In what follows, we categorize the different *levels of reproducibility* based on Definition 2 and the PRIMAD model. Table 2 summarizes the different levels and the corresponding variables that can be changed while attaining that level of reproducibility.

Repeatable. An experiment is *repeatable* if the same results can be re-generated within the same computational environment, with no changes to code or data, i.e., if the results can be repeated. This is the lowest level of reproducibility (exact reproducibility), and can be used to determine if the experiment is deterministically consistent. Note that some experiments may have non-deterministic steps (e.g., a random number generator, or third-party services that are accessed remotely). In such cases, achieving the exact same results may not be possible, but they must be consistent with the original ones, i.e., the same conclusions must be reached.

Re-runnable. We say an experiment is *re-runnable* if we can vary the input data (either the data files or the input parameters) and still get results that are consistent. Note that this allows one to determine how the experiment behaves and how robust the results are for different inputs. For instance, if the input data changes significantly and the results are still consistent, the experiment may have a broader scope worth investigating. This also allows one to evaluate whether the data originally used is representative for a given domain.

Portable. An experiment is *portable* if it can be re-executed on platforms that are different from the platform where the original results were generated. In this case, the results may be reproduced on similar environments (i.e., compatible operating system but different machines), or on different environments (i.e., different operating systems and machines). The level of portability will be higher if the experiments can be run on completely different environments. Note that, by changing platforms, library dependencies may also change, and this also affects reproducibility. For instance, the version of a software library may be available on Ubuntu (the original environment) but not on Windows (the new environment).

Extendable. An experiment is *extendable* if we can reuse its original dataflow and structure, i.e., its original sequence of steps, for other experiments. Examples include integrating the original pipeline into an existing dataflow, or even *extending* the original one to include new pre- or post-processing steps, e.g., performing additional data cleaning to the input. This is possible by having access to the implementation of the experiment, either binaries or source code. Note, however, that the experiment must be portable so that the binaries can be run in different platforms.

Modifiable. We say an experiment is *modifiable* if we can change its implementation for reuse purposes, and this is achievable by having the source code of the experiment. It is worth noting that changing the implementation also allows others to verify the correctness of the original implementation. In addition, if the experiment is not yet portable, others may modify the source code to make it runnable in different platforms.

3 Provenance for Reproducibility

The variables that correspond to the computational aspects of reproducibility, i.e., P, I, and D, can be systematically captured by keeping track of the experiment's *provenance*. Provenance refers to the record trail that accounts for the origin of a piece of data [20]. These trails enable scientists to: (i) obtain insights into the chain of reasoning used in the production of a result; (ii) verify the sequence of steps that led to the experiment findings (by capturing I – implementation); (iii) identify the inputs to an experiment and where they came from (by capturing D – data); and (iv) determine where the experiment was originally executed (by capturing P – platform).

Provenance is *essential* for attaining reproducibility [10, 11]. But different *types of provenance* can be captured, and these impact reproducibility in various ways. Thus, to choose the most appropriate provenance capture method for a scientific experiment, it is important to understand how each of these types influence the different levels of reproducibility. Below, we describe different types of provenance and how they affect reproducibility. We also present a set of tools that support automatic provenance capture.

3.1 Database Provenance

Database provenance provides a description of the derivation of a piece of data that results from executing a database query against a source database [8]. It is fine-grained, in the sense that provenance is captured for individual data items: it can be used to determine which parts of the source dataset were used to generate a piece of data in the resulting dataset. Often, database provenance is captured by reasoning about the algebraic form of the query and the underlying data model of both the source and resulting datasets. Different notions of database provenance have been defined, notably *why*-, *where*-, and *how-provenance* [9]. Given a tuple t in the result of a query, why-provenance describes all the source tuples that contributed to t or helped produce t.

Where-provenance determines where the source tuples were copied from (e.g., which cell of the relation the data comes from). How-provenance, as the name suggests, describes how the source tuples derived t (e.g., how many times each source tuple contributed to t and how they were combined).

Since databases operate in a stateful mode, database provenace is important for reproducibility. Every time a transaction commits, there is a new state that reflects the changes applied within the transaction; executing a query over different database states is likely to lead to different results.

Database provenance is *descriptive*: it provides an explanation for the derivation of results. Note that this type of provenance only captures variable D: no information about the computational environment or the database management system is available.

Levels of Reproducibility. Because database provenance captures the different states of a database (D), it can help make an experiment *repeatable*. How-provenance can make an experiment *re-runnable* by re-executing the operations with different data and assessing whether the new results are consistent with the original ones.

Tools. Systems that capture database provenance include Trio [4], Orchestra [24], Perm [22], GProM [1], and ProQL [32]. Trio is a database management system based on an extended relational algebra called ULDB, which supports uncertainty and provenance. The source data is annotated with probabilities, and the captured provenance is used to compute probabilities associated with the derived results. Orchestra is a collaborative sharing system that uses how-provenance to filter data based on user-specified trust conditions (provenance is used for data quality and reliability). In Perm, different notions of database provenance, including why-, where-, and how-provenance, are supported. Perm is implemented as a modified PostgreSQL engine, where data and its provenance are represented together in a single relation; provenance queries are then rewritten to standard SQL queries, leveraging existing query optimization techniques. GProM uses the query rewriting approach from Perm in a generic provenance middleware, in addition to supporting updates, transactions, and operation-spanning transactions. In the ProQL system, SQL is extended to support how-provenance queries.

Transaction temporal databases keep track of the different states of the database as tuples are added, deleted, or updated. Therefore, they also support fine-grained provenance and reproducibility: users can revisit old states and maintain provenance of query results even when data changes [28]. Large database vendors, such as Oracle [31] and DB2 [53], support version control for their database systems by using temporal models.

3.2 Workflow Provenance

Workflow provenance consists of the record of the derivation of a result (e.g., a dataset, an image, a plot, etc.) by a computational process represented as a scientific workflow [15]. Scientific workflows are often represented as dataflows, directed acyclic graphs (DAG) whose vertices are modules (functions) that perform computations, and data flows through the edges which connect modules. They adopt a functional, deterministic model where each module receives some input data and generates a new output: the workflow structure and inputs uniquely identify the outputs.

There are several advantages of describing computations as workflows. Notably: they provide a simple programming model where a sequence of tasks is composed by connecting the outputs of one task to the inputs of another; they support useful manipulations, such as the ability to query workflows and update them in a programmatic fashion [54]; through abstraction, they provide a high-level description of the experiment, making the specification easier to understand and more amenable for publication; and by providing a unified environment to run computations, they facilitate *provenance capture*.

Different from database provenance, workflow provenance is *coarse-grained*. Modules are black boxes and provenance captures input data they consume, the function they execute and associated parameters, and data they output; operations inside the function are not visible and no information is available about how the function manipulates the input data.

There are different types of workflow provenance [20]. *Prospective provenance* corresponds to the description of the experiment and captures the specification of the workflow structure, including its modules, connections, and inputs. *Retrospective provenance*, on the other hand, captures information about the execution of the workflow, i.e., what actually happened when the workflow was run. *Workflow evolution provenance* captures the history of the workflow, i.e., the changes that were applied to a workflow over time. If we consider the PRIMAD variables, workflow provenance contains information about the data D (input files and parameters) and the implementation I (binaries or source code for each computational step). While information about the platform P is available, it is often not captured by workflow systems.

Levels of Reproducibility. Workflow provenance supports *repeatable* and *re-runnable* experiments, as parameters and data files are systematically captured and can be varied. Workflows are also *extendable*, as scientists can change the structure of the dataflow (i.e., the prospective provenance) or incorporate it in their own pipelines. A workflow is *modifiable* if the source code for the modules is available. But if a module relies on a binary executable or a Web service to process the data, it is not possible to change the implementation. Note that, if the workflow system does not capture the platform P, portability cannot be supported.

Tools. Many scientific workflow systems are available. Taverna [41] was developed to stitch together Web and third party services: a module invokes either a Web service or a local service (e.g., R scripts, or Java API classes and methods). If a workflow is only composed of Web services, it is *portable* assuming these services are available (live and accessible) and running, but not *modifiable*. Note that using third-party resources (i.e., services provided by external hosts) may impact both repeatability and longevity because they may be interrupted or changed without notice, causing the workflow to fail.

Kepler [39] and VisTrails [19] capture both prospective and retrospective provenance. A new concept introduced by VisTrails was the notion of *provenance of workflow evolution* [21]. VisTrails treats the workflows as first-class data items and also captures their provenance. The utility of workflow-evolution provenance goes beyond reproducibility. It supports reflective reasoning [45]: users can explore multiple chains of reasoning without losing any results, and because the system stores intermediate results, users can reason about and make inferences from this information. It also enables a series of operations which simplify the exploratory process that is common for scientific experiments. For example, users can easily navigate through the space of workflows created for a given task, visually compare the workflows and their results, and explore (large) parameter spaces [21]. In addition, users can query the provenance information [55] and modify workflows by analogy [54]. VisTrails manages data manipulated by the workflows and their versions by storing input, output, and intermediate data in a versioned repository. This ensures that different versions of the input data can be recovered in the future. Using this approach, the workflow does not depend on hardcoded filenames [33]. With respect to longevity, VisTrails has a mechanism that detects when upgrades are necessary by using provenance to compare a module in a given workflow with its currently available version [34].

Galaxy [23] is a platform used to perform computational analysis on genomic data. Similar to VisTrails, Galaxy can capture information about workflow evolution, keeping multiple versions of the analysis steps so that scientists can re-run previous computations. Because the system provides a Web-based interface to create and run workflows, allowing them to be accessed through the Web from any platform, *portability* is easily attainable.

3.3 Script Provenance

Script provenance is obtained by analyzing the source code of experiments represented as scripts. Similar to workflows, we can have *prospective* and *retrospective* provenance for scripts. While the former corresponds to the script specification, the latter records the actual execution of the code (i.e., which execution branches were taken, which values were used and processed, etc.). It is also possible to capture provenance for the evolution of scripts, for example, using version control systems. To capture script provenance, code is instrumented either automatically or manually through user-defined annotations. Automatic capture has the advantage of not requiring user intervention. On the other hand, the resulting provenance data may be voluminous if provenance is captured at a fine level of detail. While annotations can be intrusive and time-consuming to add, they allow users to precisely define what should be captured.

Note that script provenance often has a *finer granularity* than workflow provenance. While workflow modules are black boxes, for scripts it is possible to observe all operations in the source code.

Levels of Reproducibility. The variables D (parameters and data files) and I (source code) are often captured, allowing an experiment to be *repeatable*, *re-runnable*, *extendable*, and *modifiable*. Provenance may also be captured for the platform P, including the library dependencies explicitly used in the code. While this may not be sufficient for *portability*, it useful for analyzing and comparing execution traces.

Tools. noWorkflow [43] transparently captures retrospective provenance from Python scripts. It does not require any code instrumentation, and the captured provenance—which includes metadata, file contents, Python dependencies (i.e., environment information at the Python level), and parameters—can be analyzed by inspecting a provenance graph, comparing multiple executions, or using inference queries. Users can choose the amount of provenance information obtained by setting the depth of the capture. noWorkflow also allows scientists to analyze workflow evolution [49].

RDataTracker [37] also captures retrospective provenance from scripts, but for the R software package. Similar to noWorkflow, the tool collects and persists provenance information, provides a provenance graph that can be inspected by users, and supports querying. Users must manually annotate the code.

Tariq et al. [56] proposed an approach through which provenance instrumentation is added at compilation time using the compiler framework LLVM [35]. This works for a range of programming languages, such as C, C++, and Java. Hooks for the capture of retrospective provenance are inserted at each function entry and exit: when the framework compiles the experiment, provenance at the binary level is transparently captured, with a small overhead.

YesWorkflow [40] captures prospective, rather than retrospective provenance. Scientists can make use of language-independent annotations to make latent dataflow information from scripts explicit. There has been also work on linking the retrospective provenance collected by noWorkflow with the prospective provenance obtained by YesWorkflow [48]. ProvenanceCurious [27] is another tool for capturing prospective provenance: it builds an abstract syntax tree from Python code to generate provenance graphs.

3.4 System-Level Provenance

System-level provenance corresponds to the provenance data captured at the operating system level, which often includes the description of the platform P, providing detailed trails of how data products are derived. This provenance is often captured by monitoring system calls and tracking processes and data dependencies between these processes. Because the dependencies are recorded at the process level, the provenance data is fine-grained. Note, however, that these processes are black boxes: it is not possible to observe what happens inside them. For this reason, in terms of the implementation I, the transparency can be low.

Levels of Reproducibility. System-level provenance is often composed of D (input parameters and data files), P, and I (binaries, at least); therefore, it can help experiments to achieve all reproducibility levels: *repeatable*, *re-runnable*, *portable*, and *extendable*. It is also possible to attain *modifiability* if the source code for the processes is available.

Tools. PASS (Provenance-Aware Storage System) [42] produces audit trails for data products by monitoring the operating system kernel. These audit trails are stored in a database and can be queried. The system supports application-generated provenance to be written into the same database via library functions, generating an integrated view of application and system-level provenance. Moreover, PASS can generate scripts to reproduce an experiment in the environment in which it was originally created and executed, thus allowing it to be *repeated*.

A number of tools have been developed to *package* and preserve the computational environment. Tools like

CDE [25], CARE [30], ReproZip [11,51], and PTU [47] trace the execution path of an experiment and obtain information about all the experiment's dependencies (e.g., files read, libraries used). This information can be used to create a package that encompasses both the experiment and its dependencies, thus supporting portability and longevity.

Most of these packing tools only support reproducibility in Linux environments that are compatible with the platform where the experiment was originally captured. CARE supports *enhanced kernel compatibility*, allowing within certain limits, for an experiment to be executed in a kernel older than the one used in the original implementation. ReproZip has an option to automatically reproduce an experiment in a virtual environment or container, thus allowing the reproduction to take place in *any* platform. ReproZip also enables scientists to inspect the captured provenance and exclude certain components from the package. Scientists can also add extra files to the package, e.g., one can add the source code if it was not originally traced. ReproZip automatically derives a VisTrails workflow for the experiment, which facilitates extending and re-using the original dataflow. PTU uses CDE internally to pack the experiment and enhances its functionalities by storing a provenance graph that can be visually inspected by scientists to determine parts of the program they want to reproduce, i.e., scientists can choose to repeat subsets of the experiment.

4 Discussion

Provenance is essential for the reproducibility of computational experiments. In this paper, we discussed different notions of reproducibility and gave an overview of different types of provenance and how they support the various levels of reproducibility. Over the past few years, there have been major developments in basic infrastructure and tools that, through systematic provenance capture, make it easier to create reproducible experiments. We have described some of these tools, the benefits they bring as well as their limitations.

While progress has been made, reproducibility is still not the norm for computational experiments. An important barrier to the adoption of reproducibility is the perception that it is time-consuming to create reproducible experiments. Recent studies have shown that insufficient time is one of the main reasons why scientists do not make their data and experiments available and reproducible [57], and that substantial effort is needed to make code work with the latest versions of required dependencies [13]. In addition, many authors argue that the process to make an experiment reproducible requires too much work for the benefit derived [6]. While one can interpret some of these as excuses, they do indicate that *usability* is an important requirement for a broader adoption of reproducibility: tools must be easy to use and, ideally, as automatic as possible. In the words of Vandewalle et al. [59], "an independent researcher should be able to reproduce all the results with a simple mouse click."

Some experiments require multiple types of provenance to be captured. For example, for a scientific workflow or script that queries and/or updates a database, it is important to collect both workflow or script provenance and database provenance. To support portability, it is also important to capture system-level provenance and detailed information about the platform and dependencies. The integration of these different types of provenance opens new opportunities to query and reason about an experiment at multiple levels. In particular, this information can enable explanations for potential issues encountered in an experiment, which can arise from the data, code, platform, or from interactions among them. But integrating different types of provenance is challenging because they are captured at different levels and have different granularities. In addition, domain-specific languages are needed that support queries over these data.

Acknowledgments. This work was supported in part by the NYU Moore Sloan Data Science Environment and National Science Foundation awards OAC-1640864, CNS-1229185, CNS-1405927, and IIS-1139832.

References

[1] B. Arab, D. Gawlick, V. Radhakrishnan, H. Guo, and B. Glavic. A Generic Provenance Middleware for Queries, Updates, and Transactions. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. USENIX Association, 2014.

[2] M. Baker. Muddled Meanings Hamper Efforts to Fix Reproducibility Crisis. *Nature News & Comment*, June 2016.

[3] C. G. Begley and L. M. Ellis. Drug Development: Raise Standards for Preclinical Cancer Research. *Nature*, 483(7391), 2012.

[4] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 953–964. VLDB Endowment, 2006.

[5] J. Bohannon. Who's Afraid of Peer Review? *Science*, 342(6154):60–65, 2013.

[6] P. Bonnet, S. Manegold, M. Bjørling, W. Cao, J. Gonzalez, J. Granados, N. Hall, S. Idreos, M. Ivanova, R. Johnson, D. Koop, T. Kraska, R. Müller, D. Olteanu, P. Papotti, C. Reilly, D. Tsirogiannis, C. Yu, J. Freire, and D. Shasha. Repeatability and Workability Evaluation of SIGMOD 2011. *SIGMOD Rec.*, 40(2):45–48, 2011.

[7] T. Brody. *Evaluating Research Impact through Open Access to Scholarly Communication*. PhD thesis, University of Southampton, May 2006.

[8] P. Buneman and W.-C. Tan. Provenance in Databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 1171–1173. ACM, 2007.

[9] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in Databases: Why, How, and Where. *Found. Trends databases*, 1(4):379–474, 2009.

[10] F. Chirigati and J. Freire. Provenance and Reproducibility. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 1–5. Springer New York, 2017.

[11] F. Chirigati, D. Shasha, and J. Freire. ReproZip: Using Provenance to Support Computational Reproducibility. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, TaPP '13, pages 1:1–1:4. USENIX Association, 2013.

[12] C. Collberg and T. A. Proebsting. Repeatability in Computer Systems Research. *CACM*, 59(3):62–69, Feb. 2016.

[13] C. Collberg, T. A. Proebsting, and A. M. Warren. Repeatability and Benefaction in Computer Systems Research: A Study and a Modest Proposal. Technical Report 14-04, University of Arizona, February 2015.

[14] S. M. Crook, A. P. Davison, and H. E. Plesser. Learning from the Past: Approaches for Reproducibility in Computational Neuroscience. In J. M. Bower, editor, *20 Years of Computational Neuroscience*, volume 9 of *Springer Series in Computational Neuroscience*, pages 73–102. Springer New York, 2013.

[15] S. B. Davidson and J. Freire. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1345–1350. 2008.

[16] D. Donoho, A. Maleki, I. Rahman, M. Shahram, and V. Stodden. Reproducible research in computational harmonic analysis. *Computing in Science & Engineering*, 11(1):8–18, Jan.-Feb. 2009.

[17] J. Freire, P. Bonnet, and D. Shasha. Computational Reproducibility: State-of-the-Art, Challenges, and Database Research Opportunities. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 593–596. ACM, 2012.

[18] J. Freire, N. Fuhr, and A. Rauber. Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041). *Dagstuhl Reports*, 6(1):108–159, 2016.

[19] J. Freire, D. Koop, E. Santos, C. Scheidegger, C. Silva, and H. T. Vo. Vistrails. In A. Brown and G. Wilson, editors, *The Architecture of Open Source Applications*. Lulu Publishing, Inc., 2011.

[20] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[21] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing Rapidly-evolving Scientific Workflows. In *Proceedings of the 2006 International Conference on Provenance and Annotation of Data*, IPAW'06, pages 10–18. Springer-Verlag, 2006.

[22] B. Glavic and G. Alonso. Perm: Processing Provenance and Data on the Same Data Model Through Query Rewriting. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 174–185. IEEE Computer Society, 2009.

[23] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences. *Genome Biology*, 11:1–13, 2010.

[24] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update Exchange with Mappings and Provenance. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 675–686. VLDB Endowment, 2007.

[25] P. Guo. CDE: A Tool for Creating Portable Experimental Software Packages. *Computing in Science Engineering*, 14(4):32–35, 2012.

[26] S. Hitchcock. The Effect of Open Access and Downloads ('Hits') on Citation Impact: A Bibliography of Studies. Technical report, University of Southampton, 2009.

[27] M. R. Huq, P. M. G. Apers, and A. Wombacher. ProvenanceCurious: A Tool to Infer Data Provenance from Scripts. In *Proceedings of EDBT '13*, pages 765–768. 2013.

[28] M. R. Huq, A. Wombacher, and P. M. G. Apers. Facilitating Fine Grained Data Provenance Using Temporal Data Model. In *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks*, DMSN '10, pages 8–13. ACM, 2010.

[29] J. P. A. Ioannidis. Why Most Published Research Findings Are False. *PLoS Med*, 2(8), Aug. 2005.

[30] Y. Janin, C. Vincent, and R. Duraffort. CARE, the Comprehensive Archiver for Reproducible Execution. In *Proceedings of the 1st ACM SIGPLAN Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering*, TRUST '14, pages 1:1–1:7. ACM, 2014.

[31] K. Jernigan, L. Guo, V. Krishnaswamy, V. Radhakrishnan, V. Raja, and T. Shetler. Oracle Total Recall with Oracle Database 11g Release 2. White paper, Oracle, 2009.

[32] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying Data Provenance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 951–962. ACM, 2010.

[33] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva. Bridging Workflow and Data Provenance Using Strong Links. In M. Gertz and B. Ludäscher, editors, *22nd International Conference on Scientific and Statistical Database Management, SSDBM 2010*, pages 397–415. Springer Berlin Heidelberg, 2010.

[34] D. Koop, C. E. Scheidegger, J. Freire, and C. T. Silva. The Provenance of Workflow Upgrades. In D. L. McGuinness, J. R. Michaelis, and L. Moreau, editors, *Third International Provenance and Annotation Workshop, IPAW 2010*, pages 2–16. Springer Berlin Heidelberg, 2010.

[35] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04. IEEE Computer Society, 2004.

[36] S. Lawrence. Free Online Availability Substantially Increases a Paper's Impact. *Nature*, 411(6837):521, May 2001.

[37] B. Lerner and E. Boose. RDataTracker: Collecting Provenance in an Interactive Scripting Environment. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. USENIX Association, 2014.

[38] R. LeVeque. Python tools for reproducible research on hyperbolic problems. *Computing in Science & Engineering*, 11(1):19–27, Jan.-Feb. 2009.

[39] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, Aug. 2006.

[40] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, R. K. Bocinsky, Y. Cao, J. Cheney, F. Chirigati, S. Dey, J. Freire, C. Jones, J. Hanken, K. W. Kintigh, T. A. Kohler, D. Koop, J. A. Macklin, P. Missier, M. Schildhauer, C. Schwalm, Y. Wei, M. Bieda, and B. Ludäscher. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. *International Journal of Digital Curation*, 10:298–313, 2015.

[41] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, Reloaded. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management*, SSDBM'10, pages 471–481. Springer-Verlag, 2010.

[42] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware Storage Systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06. 2006.

[43] L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire. noWorkflow: Capturing and Analyzing Provenance of Scripts. In B. Ludäscher and B. Plale, editors, *Proceedings of the 5th International Provenance and Annotation Workshop, IPAW 2014*, pages 71–83. Springer International Publishing, 2015.

[44] G. Naik. Mistakes in Scientific Studies Surge, August 2011. `http://online.wsj.com/article/SB10001424052702303627104576411850666582080.html`.

[45] D. A. Norman. *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. Addison Wesley, 1994.

[46] R. Nuzzo. How Scientists Fool Themselves, and How They Can Stop. *Nature*, 526(7572):182–185, 2015.

[47] Q. Pham, T. Malik, and I. Foster. Using Provenance for Repeatability. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, TaPP '13, pages 2:1–2:4. USENIX Association, 2013.

[48] J. F. Pimentel, S. Dey, T. McPhillips, K. Belhajjame, D. Koop, L. Murta, V. Braganholo, and B. Ludäscher. Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow. In M. Mattoso and B. Glavic, editors, *Proceedings of the 6th International Provenance and Annotation Workshop, IPAW 2016*, pages 161–165. Springer International Publishing, 2016.

[49] J. F. Pimentel, J. Freire, V. Braganholo, and L. Murta. Tracking and Analyzing the Evolution of Provenance from Scripts. In M. Mattoso and B. Glavic, editors, *Proceedings of the 6th International Provenance and Annotation Workshop, IPAW 2016*, pages 16–28. Springer International Publishing, 2016.

[50] H. A. Piwowar, R. S. Day, and D. B. Fridsma. Sharing Detailed Research Data Is Associated with Increased Citation Rate. *PLoS ONE*, 2(3):e308, 03 2007.

[51] R. Rampin, F. Chirigati, D. Shasha, J. Freire, and V. Steeves. ReproZip: The Reproducibility Packer. *The Journal of Open Source Software (JOSS)*, 2016.

[52] A. Rokem and F. Chirigati. Glossary. In J. Kitzes, D. Turek, and F. Deniz, editors, *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences*, pages 71–91. UC Press, 2017.

[53] C. M. Saracco, M. Nicola, and L. Gandhi. A Matter of Time: Temporal Data Management in DB2 for z/OS. White paper, IBM, 2010.

[54] C. Scheidegger, H. Vo, D. Koop, J. Freire, and C. Silva. Querying and Creating Visualizations by Analogy. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1560–1567, 2007.

[55] C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and re-using workflows with vistrails. In *ACM SIGMOD*, pages 1251–1254, 2008.

[56] D. Tariq, M. Ali, and A. Gehani. Towards automated collection of application-level data provenance. In *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2012)*. USENIX, 2012.

[57] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff, and M. Frame. Data Sharing by Scientists: Practices and Perceptions. *PLOS ONE*, 6(6):1–21, 2011.

[58] R. Van Noorden. Science Publishing: The Trouble with Retractions. *Nature*, 478:26–28, 2011.

[59] P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible Research in Signal Processing - What, Why, and How. *IEEE Signal Processing Magazine*, 26(3):37–47, May 2009.