Nullius in Verba: Reproducibility for Database Systems Research, Revisited

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens AG, Corporate Research
Regensburg/Munich, Germany
wolfgang.mauerer@othr.de

Stefanie Scherzinger
University of Passau
Passau, Germany
stefanie.scherzinger@uni-passau.de

***Abstract*—Over the last decade, reproducibility of experimental results has been a prime focus in database systems research, and many high-profile conferences award results that can be independently verified. Since database systems research involves complex software stacks that non-trivially interact with hardware, sharing experimental setups is anything but trivial: Building a working reproduction package goes far beyond providing a DOI to some repository hosting data, code, and setup instructions.**

This tutorial revisits reproducible engineering in the face of state-of-the-art technology, and best practices gained in other computer science research communities. In particular, in the hands-on part, we demonstrate how to package entire system software stacks for dissemination. We show how version control systems that allow for non-linearly rewriting recorded history can document the structured genesis behind experimental setups in a way that is substantially easier to understand, without involvement of original authors, compared to detour-ridden, strictly historic evolution. To ascertain *long-term* reproducibility over decades (or ideally, forever), we discuss why relying on open source technologies massively employed in industry has essential advantages over approaches crafted specifically for research.

***Index Terms*—reproducible science, reproduction, reproduction package, docker, git, scientific attribution, scientific method**

I. Introduction & Title

The tutorial *Nullius in Verba: Reproducibility for Database Systems Research, Revisited* addresses one of the most crucial, yet unfortunately often underappreciated aspects of science: The provision of sufficient information on technical aspects, methodology, and computational steps to reproduce published results. *Nullis in verba*, the motto of the Royal Society as embodied in their coat of arms for centuries, is an epitome to the importance of this desideratum. Contemporary means of information technology make it easier than ever to realise the goal of pervasive reproducibility in science. Yet perhaps paradoxically, these means are far from receiving universal consideration and support: A survey (published in Nature [1]) among scientists shows that 70% have failed to replicate other work, and more than half have struggled to reproduce their own(!) experiments.

This discrepancy has become an academic topic of debate, and dedicated research evaluates the (oftentimes wanting) state of affairs in computer science research in general (see, e.g., Refs. [2]–[5]), and also in data management research in particular, such as in the VLDB ("pVLDB Reproducibility") and SIGMOD communities [6] ("ACM SIGMOD 2019 Reproducibility") (PDF provides clickable hyperlinks).

Unfortunately, the terminology is not globally standardised, but in the following, we adopt the corresponding conventions of the ACM (see "ACM review and badging"): experiments are expected to be *repeatable*; essentially, the same team with the same experimental setup can reliably achieve identical results in subsequent trials. Moreover, experiments should be *reproducible*, so that using the same experimental setup operated by a different team achieves the same results. Note that the ACM changed the definition of these terms on Aug 24, 2020.

Over the years, high-quality tutorials on reproducible engineering have been presented at database conferences [7]–[9] (and have recently been awarded[1]). After a decade of intense debate, we revisit this discussion by contrasting the state-of-the-art in reproducible engineering today in retrospective to earlier tutorials, and show lessons learned. Besides considering experience from multiple fields of computer science, we also include industrial recommendations: Reproducibility is a core concern of commercial systems engineering, where maintenance *and* simultaneous continuous development activities often span decades.[2] Methods and approaches developed in such scenarios also apply to scientific problems.

The approach we advocate in our tutorial rests on three pillars:

1.) Presenting the genesis of results in a structured, logically consistent way, instead of showing only the final state of research, or a complete temporal revision control system log. Following Knuth's seminal *literate programming* approach [10], we argue why programs (*and* their temporal evolution) should be seen as communication with fellow humans instead of instructions for machines to ease future reproduction. We use pragmatic approaches developed in large, international and multi-disciplinary infrastructure projects like the Linux kernel to achieve this goal.

[1]SIGMOD 2020 contribution award: https://sigmod.org/sigmod-awards/citations/2020-sigmod-contributions-award/.

[2]Consider the Boeing 747 aircraft as an extreme example: development started in 1966, and the last machines produced in 2022 will be in service until about 2050. This results in industrial maintenance and continuous development that spans almost a *century*.

2.) Provisioning *reproduction packages* in a multi-stage process that distinguishes between (a) *deterministically* building executable artefacts from source (ideally bit-wise constant) in a likewise deterministically constructed build environment, (b) bundling artefacts, input data and instrumentation scripts into a self-contained collection (inspired by the concept of apps), and (c) the execution of any measurement-based evaluations and experiments on diverse hardware, including output validation.
3.) Ascertaining *long-term availability* (over decades), based on open source community-supported, mature approaches that are themselves based on reproducible and long-term archived data formats, tools and conventions.
We argue, using examples, why the plain availability of portable source code, input data, build mechanisms and dispatcher scripts that were considered as gold standard a decade ago almost always fail to provide such guarantees.

Our tutorial will, using live and hands-on examples, discuss which state-of-the-art software and approaches allow for creating reproduction packages that enable third parties to understand existing research in detail, perform the exactly same experiments and evaluations as the original authors, and fully reproduce published results. We particularly consider how to ascertain that a reproduction (really and realistically) works without assistance by the original authors, and after substantial periods of time have passed since publication.

II. Tutorial presenters

Wolfgang Mauerer is a professor of theoretical computer science at the Technical University of Applied Sciences Regensburg, and a senior research scientist at Siemens AG, Corporate Research, Munich. His interests focus on quantitative and empirical software engineering, low-level systems engineering, and quantum computing. He received his PhD in physics (where performing reproducible measurements and experiments is one of the key educational aspects) from the Max Planck Institute for the Science of Light. He has worked extensively with open source communities for commercial and academic purposes, and many of his papers have received recognitions for reproducible science. He is one of the founders and a member of the technical steering committee of the Civil Infrastructure Platform, an international consortium dedicated to providing ultra-long-term maintainable software base platforms with multi-decade lifetimes – a challenge that has the reproducible provision of curated, rapidly changing software components at its very core.

Stefanie Scherzinger is a professor at the University of Passau, where, as of recently, she chairs the *Scalable Database Systems* group. Traditionally, her research has had a focus on schema evolution, and is motivated by her previous work experience as a software engineer at first IBM and then Google. At Google, she was involved with achieving reproducible software builds by managing dependencies between software libraries, a challenge that is quite related to achieving reproducible results in academic research. In particular, her recent work on conducting and reproducing schema evolution case studies has sparked her interest in reproducible engineering.

The presenters' diverse domain knowledge, and their expertise in both academic research and commercial software development, enables them to provide a multi-facetted discussion of challenges, and a balanced evaluation of the benefits and drawbacks of alternative approaches.

III. Outline

The proposed tutorial is structured as follows.

a) Producing consistent, readable histories: The first part of the tutorial addresses the creation of author-supplied components, and their documentation. Reproduction packages typically consist of several artefacts: Generated or pre-existing input data, processing and measurement code, output or derived data, and visualisation scripts. Additionally, they may contain *extensions* to existing software.

Such complex packages are the result of an iterative process that usually produces an incremental, step-wise understanding of features or hypotheses. Eventually, this leads to either accepting or refuting a hypothesis, or to performing final, definitive runs of a measurement after a series of pre-trials, followed by the evaluation of the results. This raises the question of what (and what not!) to document in a reproduction package, and how.

While any *structural decisions* are worth preserving, the temporal order of the *thought process* that led to intermediate results or to said decisions is usually not. Standard log books, or version control system logs are a modern digital alternative. Yet typically, these contain many unproductive detours that do not provide significant illumination, or they document transient technical issues and glitches. In either case, such minutiae make it harder than necessary for third-party researchers to follow the line of work that resulted in a particular finding. Using examples from published research, we demonstrate how a chronological line of thought that invariably arises in research can be turned into a consistent, logical sequence of steps that represents the *outcomes* of the thought process, not unlike propagated in literate programming [10]. Especially nonlinear history rewriting as offered by modern revision control systems is a powerful, yet not often utilised means to achieve these ends. The efforts result in *patch stacks* comprising orthogonal commits (i.e., the smallest reasonable increment in research code or analysis scripts that is worth preserving) that are also well suited to augment existing software with new functionality, and to build components that comprise external and custom-developed parts.

Additionally, we discuss established conventions for documenting commits that have been devised to understand the historical evolution of large software systems, but can likewise be applied to documenting research progress. Similarly, we introduce techniques to provide *trails of responsibility* (who jointly authored changes, who provided reviews, who participated in design decisions etc.) that are routinely created outside academia, but not established in many areas of computer

science, contrariwise to the care taken in giving credit and attribution in published papers.

As an alternative solution to turning chronological records into structured logs, we discuss the use of scientific notebooks like Jupyter, where similar goals can be achieved with systematic document reorganisation.

b) Packaging research artefacts: The second part of the tutorial focuses on the reproducible composition of long-term stable execution environments that run software code and analysis scripts produced as research artefacts. Almost all research relies on standard IT base systems like Linux and Windows, and it is well known that such systems exhibit drastic change rates in terms of adding new features, deprecating old ones, or changing the semantics of existing ones.

Stable environments for both, building and executing experiment code (including compilers, middleware, libraries, and possibly also an OS kernel) are required. Additionally, issues specific to data-centric research like the provision of voluminous amounts of data must be considered.

We show how to combine (and identify!) any required base components that must be included in from-scratch system installations, how to automatically compose these into standalone collections (i.e., a virtual machine that produces identical result regardless of a the host system), and how to run scientific analyses on them. We also address the question how to best integrate the creation of such collections into the research process from the start, with little overhead, and how geographically distributed research benefits from the effort.

We give guidelines when to rely on binary system sources for distribution-level software, and when to re-build components from source. We also investigate the question of long-term availability of external sources, and provide guidelines on how to *not* rely on the long-term availability of these.

Since a substantial fraction of published research relies on and interacts with open source software that is also included in packages, we briefly discuss typical license options and the arising obligations. This includes integrating closed-source systems and handling non-disclosure agreements.

c) Describing execution environments: Finally, we illustrate means of *properly* specifying combinations of hardware and software. Underspecifying the execution environment has been identified an issue in previous tutorials on reproducible engineering (e.g., see [7]), and still seems to be an ongoing education process within the community.

Typical specifications provide experimental conditions like "Linux version 5.1.92 on a Dull Powervortex 4711 with 24 GiB of RAM was used". This is insufficient for reliable reproductions—non-standard kernel extensions that may vary widely depending on the distribution, specific settings for tuning parameters that exist in a wide variety on every system, and many other factors that may easily be dismissed as irrelevant technical details can impact the results of measurements by orders of magnitude, as our tutorial will demonstrate.

The overall process and artefact collection that we advocate in the tutorial is illustrated in Figure 1: A Docker build recipe produces a whole-system container in an open, publicly documented format that can be generated without modifying the host system. The build recipe performs the composition, and integratesm binary sources (pre-compiled executables for standard tasks, but possibly also input data), custom research software, and changes to existing source codes in form of orthogonal patch stacks. Any measurements and experiments performed in a paper can then either be directly performed *within* the container, or on external hardware, both local machines and cloud deployments. To guarantee a consistent execution environment regardless of the underlying target platform, we show how a collection with executables for all measurements, dispatchers and evaluation scripts, and data generators or pre-generated data sets should be created. By transferring this collection to a target, experiments can be automatically executed, and charts, tables, and other forms of visualisation be generated.

d) Further challenges: Finally, the tutorial addresses how to handle a number of details that may seem trivial, but often lead to substantial problems when trying to perform an independent reproduction years after publication (e.g., as we discuss in Ref. [11]):

- documenting and automating the mechanics of an analysis process,
- creating long-term available, DOI-safe archives of all artefacts that really work (which is not always guaranteed by following the DOI requirements),
- making measurements on hardware reproducible, and limitations of the endeavour,
- ascertaining (ideally bit-wise) reproducible builds, and the limits thereof,
- dealing with proprietary and closed-source components.

We round up the discussion by identifying some bad practices and anti-patterns of reproducible research.

IV. Mode of Delivery

For the hands-on part of the tutorial, we assume that our attendees have a working installation of docker on their machine (which can run Linux, Windows, or MacOS). Additionally, we assume a working git installation, some familiarity with the command line (which we will prefer over GUI based tools, because it allows for a very efficient workflow that does not add much overhead to the usual research routine, once mastered), a text-based editor, and either Python or R to follow the data analysis examples.

We will provide a pre-fabricated docker image that is systematically extended during the tutorial session, and that can serve as blueprint for reproduction packages.

We have designed the tutorial so that it can be taught in a virtual format that does not require physical presence, as is usual in these pandemic days.

V. Length

The intended length of the tutorial is 3 hours, roughly split between the discussion of the two aspects of (a) consistent and understandable histories and (b) packaging research artefacts.

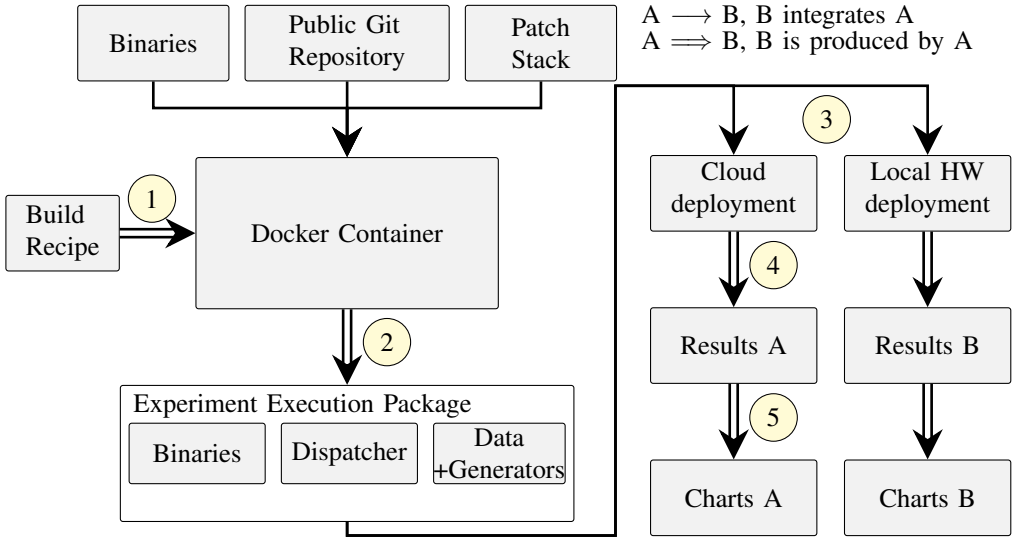

Figure 1. Structure of reproduction packages as advocated in the tutorial: Based on system binaries, external and internal code in git repositories, and patch stacks with changes to existing components (both organised as orthogonal changes with proper credit tracking), a build recipe induces generation of a host-system independent docker container as (static and immutable) build environment for measurement binaries (1). The result of the build process is an experiment execution package (2) that can be deployed (3) on cloud systems, or on local hardware, each time without any dependence on target-system provided artefacts. The experimental runs (4) generate data, which are post-processed, evaluated and visualised (5) by scripts and code contained in the measurement package.

It would be possible to limit the tutorial to 1.5 hours, and only cover the second aspect. Ascertaining the goals of the latter is more pressing than the former, but the presenters feel that both topics are best addressed together.

VI. Target Audience

Our target audience includes anyone who actively performs and publishes scientific research, from beginning graduate students coming up to speed with contemporary research methods, to seasoned researchers and faculty who want to improve their reproducibility skills, or want to learn about technological developments and new tools that might not have been in common use when they started their careers.

The tutorial introduces the audience to established methods of creating reproduction packages [12]–[14], inspired by working conventions from multiple fields of science, and tailored for typical scenarios as they arise in database systems research. We hope to further the use of such methods and techniques in research from the ground up, which should lead to more robust and trustworthy results, and a community that has more time to build upon existing results (and therefore, stand on the shoulders of giants) instead of re-building previous research artefacts from PhD generation to PhD generation.

VII. Prior Offerings

This tutorial has not been offered before in this form.

We are aware of earlier tutorials from within the database research community, e.g., [7], [8], also at ICDE [9], from about ten years ago. Our proposal revisits this important topic, discusses new methodologies and considerations, and the effectiveness of previous guidance from hindsight.

References

[1] M. Baker, "Is there a reproducibility crisis?" *Nature*, vol. 533, pp. 452–454, 05 2016.

[2] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis *et al.*, "The Seattle Report on Database Research," *SIGMOD Rec.*, vol. 48, no. 4, Feb. 2020.

[3] M. Pawlik, T. Hütter, D. Kocher, W. Mann, and N. Augsten, "A Link is not Enough – Reproducibility of Data," *Datenbank-Spektrum*, vol. 19, no. 2, pp. 107–115, Jul 2019.

[4] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich *et al.*, "The repeatability experiment of SIGMOD 2008," *SIGMOD Rec.*, vol. 37, no. 1, pp. 39–45, 2008.

[5] C. Collberg and T. A. Proebsting, "Repeatability in Computer Systems Research," *Commun. ACM*, vol. 59, no. 3, p. 62–69, Feb. 2016.

[6] S. Manegold, I. Manolescu, L. Afanasiev, J. Feng *et al.*, "Repeatability & workability evaluation of SIGMOD 2009," *SIGMOD Rec.*, vol. 38, no. 3, pp. 40–43, 2009.

[7] S. Manegold and I. Manolescu, "Performance Evaluation in Database Research: Principles and Experience," in *Proc. EDBT '09*, 2009, slide deck at https://homepages.cwi.nl/~manegold/DBDM/DBexperimentsTutorial-1x1.pdf.

[8] P. Bonnet, D. Shasha, and J. Freire, "Computational reproducibility: state-of-the-art, challenges, and database research opportunities," in *Proc. SIGMOD '12*, 2012, pp. 593–596.

[9] I. Manolescu and S. Manegold, "Performance Evaluation in Database Research: Principles and Experience," in *Proc. ICDE '08*, 2008.

[10] D. E. Knuth, "Literate programming," *Comput. J.*, vol. 27, no. 2, p. 97–111, May 1984.

[11] D. Braininger, W. Mauerer, and S. Scherzinger, "Replicability and reproducibility of a schema evolution study in embedded databases," *Proc. EmpER 2020*, 2020.

[12] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[13] R. Chamberlain and J. Schommer, "Using docker to support reproducible research," *DOI: https://doi. org/10.6084/m9. figshare*, vol. 1101910, p. 44, 2014.

[14] W. Elmenreich, P. Moll, S. Theuermann, and M. Lux, "Making computer science results reproducible – A case study using Gradle and Docker," *PeerJ Preprints*, vol. 6, 2018.

Acknowledgement. We thank Edson Lucas for drawing Figure 1.