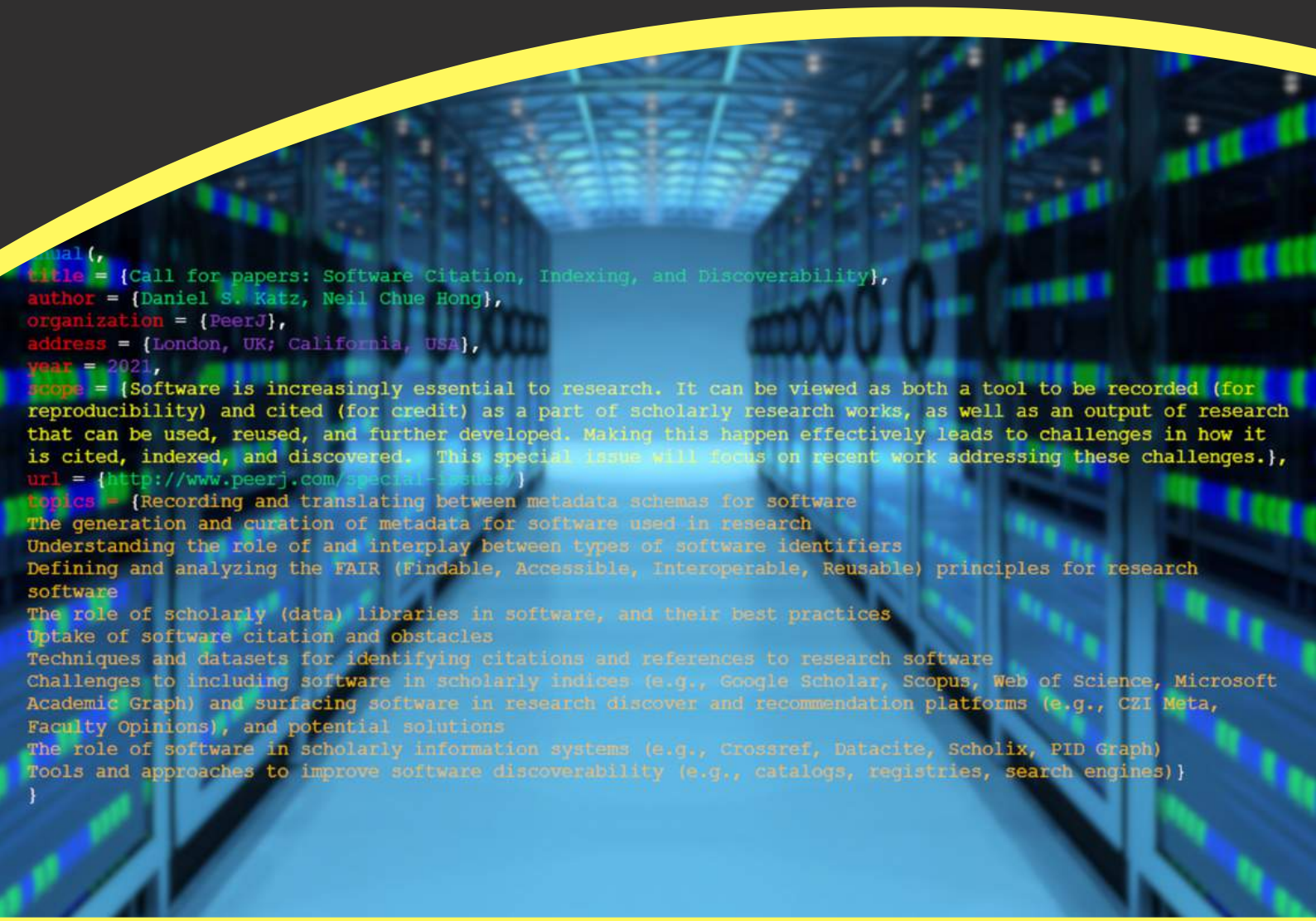
SPECIAL ISSUE

Software Citation, Indexing, and Discoverability

A *PeerJ Computer Science* Special Issue investigating the importance of - and best practice for - citing, indexing, and discovering software used as a scholarly research tool.



PeerJ Computer Science

A *PeerJ Computer Science* Special Issue

Software Citation, Indexing, and Discoverability

Editor Feature

PeerJ Computer Science

Special Issue Editor

Daniel S. Katz
Chief Scientist, National Center for Supercomputing Applications,
University of Illinois at Urbana-Champaign, USA

Dan Katz's interest is in the development and use of advanced cyberinfrastructure to solve challenging problems at multiple scales. His technical research interests are in applications, algorithms, fault tolerance, and programming in parallel and distributed computing, including HPC, Grid, Cloud, etc. He is also interested in policy issues, including citation and credit mechanisms and practices associated with software and data, organization and community practices for collaboration, and career paths for computing researchers.

Special Issue Editor

Neil P Chue Hong
Director, Software Sustainability Institute,
University of Edinburgh, UK.

Neil Chue Hong is the founding Director and PI of the Software Sustainability Institute, a collaboration between the universities of Edinburgh, Manchester, Oxford and Southampton. He enables research software users and developers to drive the continued improvement and impact of research software. From 2007-2010, he was Director of OMII-UK at the University of Southampton, which provided and supported free, open-source software for the UK e- Research community. In addition to sitting on several project advisory committees, he is the Editor-in-Chief of the Journal of Open Research Software, chair of the Met Office / UKRI ExCALIBUR Steering Committee, past chair of the EPSRC Strategic Advisory Team on e-Infrastructure, co-author of "Best Practices for Scientific Computing" and "An Open Science Peer Review Oath", and co-organiser of the Software Engineering for Science workshop series.

Special issue on software citation, indexing, and discoverability. Daniel S. Katz, Neil P. Chue Hong. https://doi.org/10.7717/peerj-cs.1951

The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central. David Schindler, Felix Bensmann, Stefan Dietze, Frank Krüger. https://doi.org/10.7717/peerj-cs.835

Research artefacts and citations in computer systems papers. Eitan Frachtenberg. https://doi.org/10.7717/peerj-cs.887

Understanding progress in software citation: a study of software citation in the CORD-19 corpus. Caifan Du, Johanna Cohoon, Patrice Lopez, James Howison. https://doi.org/10.7717/peerj-cs.1022

Nine best practices for research software registries and repositories. Daniel Garijo, Hervé Ménager, Lorraine Hwang, Ana Trisovic, Michael Hucka, Thomas Morrell, Alice Allen, Task Force on Best Practices for Software Registries, SciCodes Consortium. https://doi.org/10.7717/peerj-cs.1023

A survey of researchers' code sharing and code reuse practices, and assessment of interactive notebook prototypes. Lauren Cadwallader, Iain Hrynaszkiewicz. https://doi.org/10.7717/peerj.13933

PeerJ Computer Science

Special issue on software citation, indexing, and discoverability

Daniel S. Katz[1] and Neil P. Chue Hong[2,3]

[1] National Center for Supercomputing Applications, Department of Computer Science, Department of Electrical and Computer Engineering, School of Information Sciences, University of Illinois at Urbana-Champaign, Urbana, Illinois, United States of America
[2] Edinburgh Parallel Computing Centre, University of Edinburgh, Edinburgh, United Kingdom
[3] Software Sustainability Institute, University of Edinburgh, Edinburgh, United Kingdom

This Editorial article has not been externally peer reviewed

ABSTRACT

Software plays a fundamental role in research as a tool, an output, or even as an object of study. This special issue on software citation, indexing, and discoverability brings together five papers examining different aspects of how the use of software is recorded and made available to others. It describes new work on datasets that enable large-scale analysis of the evolution of software usage and citation, that presents evidence of increased citation rates when software artifacts are released, that provides guidance for registries and repositories to support software citation and findability, and that shows there are still barriers to improving and formalising software citation and publication practice. As the use of software increases further, driven by modern research methods, addressing the barriers to software citation and discoverability will encourage greater sharing and reuse of software, in turn enabling research progress.

Subjects Data Science, Digital Libraries, Emerging Technologies, Network Science and Online Social Networks, Software Engineering

Keywords Software, Citation, Indexing, Discoverability, FAIR principles, Research software

Submitted 29 February 2024
Accepted 29 February 2024
Published 26 March 2024

Corresponding author
Daniel S. Katz, d.katz@ieee.org

Article type
Editorial

Additional Information and Declarations can be found on page 5

DOI **10.7717/peerj-cs.1951**

INTRODUCTION

Software is increasingly essential to research. It can be viewed as both a tool to be recorded (for reproducibility) and cited (for credit) as a part of scholarly research works, as well as an output of research that can be used, reused, and further developed.

In 2021, PeerJ Computer Science staff invited us to propose a special issue, and because of our interest in the role of software in research, we chose to focus on Software Citation, Indexing, and Discoverability. This choice was also partially based on our co-leadership of the FORCE11 *Software Citation Implementation Working Group (2023)*, which followed on a set of software citation principles published in 2016 (*Smith, Katz & Niemeyer, 2016*), working to move the community from having an idea of what could and should be done in this area to actually having a culture of citing software. We were (and are) also pursuing other activities both individually and collectively, generally aimed at increasing the sustainability of research software, such as the Software Sustainability Institute (SSI)

(*Crouch et al., 2013*) and the US Research Software Sustainability Institute (URSSI) (*Carver et al., 2018*).

Ensuring that the use of software in research, particularly in publications, is effectively understood and recorded leads to challenges in how it is cited, indexed, and discovered. These include challenges relating to: software metadata; identifiers for software and their relationship to those of other research objects, and other software; the role of other stakeholders such as indexes, libraries and registries; fostering adoption; development of related tools; and the role of the FAIR principles in this space. The special issue was intended to focus on recent work addressing these challenges, particularly in the context of the FORCE11 working group.

Between 2017 and 2023, the FORCE11 group published a set of software citation implementation challenges (*Katz et al., 2019*), published checklists for (paper) authors (*Chue Hong et al., 2019b*) and (software) developers (*Chue Hong et al., 2019a*), published best practices for software repositories and registries (*Task Force on Best Practices for Software Registries et al., 2020*), published guidance for journals (*JATS4R, 2021*; *Katz et al., 2021*; *Stall et al., 2023*), and worked on metadata systems such as CodeMeta (*Jones et al., 2017*) and CITATION.cff (*Druskat et al., 2021*).

Additional relevant work in this area includes: FAIR for Research Software (FAIR4RS), which has created a new set of FAIR (findable, accessible, interoperable, and reusable) principles specifically for research software (*Chue Hong et al., 2022*); the work of the RDA/FORCE11 Software Source Code Identification working group to produce use cases for persistent identifiers for software source code (*Research Data Alliance/FORCE11 Software Source Code Identification WG et al., 2020*); the European Open Science Cloud's Task Force report on Scholarly Infrastructures of Research Software (*European Commission and Directorate–General for Research and Innovation, 2020*); NISO's efforts to standardize reproducibility badging, including metadata relevant for citation (*NISO, 2021*); and the work of the Digital Preservation Coalition and Software Preservation Network on motivations for preserving software (*Morrissey, 2020*).

The call for the special issue was issued in 2021, and in 2022, five papers successfully passed through the peer review and publication process. The remainder of this editorial discusses the papers and their potential impact, and where we think things are going next.

PAPERS AND IMPACT

The final special issue contains the following published articles:

- *Schindler et al. (2022)*: "The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central." This work built a 300-million-triple knowledge graph of 11.8 million software mentions and affiliated metadata generated through supervised information extraction models that distinguish different types of software and mentions, trained on a gold standard *corpus*, and applied to more than three million scientific articles. It then used this graph to perform a large-scale analysis of software usage and citation practices. The analysis provides insights into the evolution of software usage and citation patterns across various fields, ranks of journals, and

impact of publications. The authors publicly share all their data and models to facilitate further research into scientific use and citation of software.

- *Frachtenberg (2022)*: "Research artifacts and citations in computer systems papers." This paper studies the field of computer systems, which involves the engineering, implementation, and measurement of complex systems software and data. In this field, the reproducibility and replicability of research results depends on the availability of artifact, because system software often embodies numerous implicit assumptions and parameters that are not fully documented in articles. The work built a cross-sectional dataset of papers from 56 contemporaneous systems conferences, including data on conferences, papers, authors, citation counts, and the release, ongoing availability, badges, and locations of associated research artifacts. This data showed that artifacts were shared in 30% of all conference papers and 43% of papers in conferences that actively evaluated artifact sharing, and that the papers with shared artifacts had 75% more citations. Even after controlling for numerous confounding covariates, the release of an artifact increased a paper's citations by 34%.
- *Du et al. (2022)*: "Understanding progress in software citation: a study of software citation in the CORD-19 *corpus*." This work investigated progress toward improved software citation by examining current software citation practices. It used a machine-learning-based data pipeline to extract software mentions from a collection of more than 280,000 scholarly articles on COVID-19 and related historical coronaviruses. The authors then closely examined a sample of the extracted software mentions and searched online for the mentioned software projects and their citation requests, in order to understand the status of software citation. Positively, they found increasing mentions of software versions, increasing open source practices, and improving software accessibility. However, they also found high numbers of informal mentions that didn't credit software authors, as well as problems where software developers requested citations that did not match software citation advocacy recommendations and that were not followed by paper authors. Finally, they discussed implications for software citation advocacy and standard making efforts seeking to improve the situation.
- *Garijo et al. (2022)*: "Nine best practices for research software registries and repositories." Differing from the previous papers, this work was about the role of registries and repositories that aim to include software in their contents. These systems have a key role in supporting and improving software findability and research transparency, providing information for software citations, and fostering preservation of computational methods in a wide range of disciplines. However, developing them takes effort and there are few guidelines available to help their creators and operators. To address this need, the previously mentioned FORCE11 *Software Citation Implementation Working Group (2023)* convened a task force to distill the experiences of the managers of existing resources in setting expectations for all stakeholders. This paper described the resultant best practices, which include defining the scope, policies, and rules that govern individual registries and repositories, along with the background, examples, and collaborative work that went into their development. The paper's authors

believe that establishing specific policies such as those presented here will help other scientific software registries and repositories better serve their users and their disciplines.

- *Cadwallader & Hrynaszkiewicz (2022)*: "A survey of researchers' code sharing and code reuse practices, and assessment of interactive notebook prototypes." While the first three papers studied research works, and the fourth studied research repositories and registries, this paper studied researchers themselves. It asked researchers in computational biology about how often and why they look at code (most often, to gain a better understanding of the article), which methods of accessing code they find useful (most often, links to a code repository containing an archived version of the software) and why, what aspects of code sharing are important to them (ensuring that the code was running in the correct environment and sharing code with good documentation), and how satisfied they are with their ability to complete these tasks (generally, they were satisfied). The paper also asked researchers to examine a specific code-sharing tool that would enable readers to easily run the code, and if they would be willing to spend more time to use this tool. The average researcher was found to be unwilling to incur the additional costs (in time, effort or expenditure) that are currently needed to use code sharing tools alongside a publication. Based on this, the authors determined that different models are needed for funding and producing interactive or executable research outputs if they are to reach a large number of researchers.

These papers both create a set of work that can be further developed to better understand software citation, indexing, and discovery, and also demonstrate factors that should lead to increased understanding of the role of software in research and the importance of the work of its developers and maintainers.

Schindler et al. (2022) helps us understand how software is used and cited through their analysis of the published record as captured in PubMed Central, which can be considered a baseline of data and tools that can be used to collect future data and understand future changes as well.

Frachtenberg (2022) looks at artifacts, including software, and finds that their presence, at least in computer systems conference papers, leads to increased citations of these papers, which hopefully will lead to increased sharing of such artifacts in the future. Reuse of these artifacts should lead to increased software sharing in this field, and better recognition of software efforts.

Du et al. (2022) overlaps with *Schindler et al. (2022)* and *Frachtenberg (2022)* to some extent, though using a somewhat different set of research papers. It also focused on formal recommendations for citation, and how both software developers follow them as well as how paper authors use the requests from software developers.

Garijo et al. (2022) discusses important practices for repositories and registries that store or refer to software. Having a set of such practices leads to more better citations, indexing, and discoverability of software as used in papers and other research works.

Cadwallader & Hrynaszkiewicz (2022) helps us understand how researchers use code associated with publications, how they share it statically today, and the difficulties in sharing it more dynamically in the future.

LOOKING FORWARD

The papers in this special issue clearly provide a snapshot of citation practices today, and they define needs in the field and the research ecosystem, including for:

- Better and ongoing collection of data about software citation in publications. In particular, we observe that two to three of the five papers took advantage of the fact that a high fraction of the biomedical literature has been openly available *via* PubMed. This leads to collected data that represents this field, but may not represent other fields as well, if the fields and their practices differ. As open access publications become more common across fields, we hope that other disciplines will become as well studied as biomedicine, and that any disciplinary differences will become apparent so that they can be addressed.
- Better communication about recommended software citation to software developers, leading them to make citation requests that are more likely to be followed by paper authors.
- Better citation practices by paper authors, perhaps following community/discipline-specific guidelines. Note that very recent work by *Ram & Howison (2023)* has found that practitioners often understand *how* to cite software but have widespread uncertainty about community norms on *which* software to cite, given the limited available space for references in papers, so it seems likely that this need involves both the how discovered by papers in this special issue as well as the which.
- More consistent use of best practices for registries and repositories that store or refer to software.
- Policies and tools that are very low cost (in time and money) for researchers to use to include their software in publications, if we want to move beyond links from publications to static software in repositories, towards more interactive or reusable forms.

The increasing use and ubiquity of software in research—now also driven by data science, machine learning, and open research/open access—emphasises the importance of software citation on transparency, reproducibility and reusability of research.

Addressing these needs both individually and collectively will lead to software that is more frequently cited, indexed, and discovered, encouraging more software sharing and reuse, and in turn leading to better and faster research progress.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

Neil Chue Hong was supported by the UK Research Councils through grant EP/S021779/1.

Grant Disclosures

The following grant information was disclosed by the authors:
UK Research Councils through grant EP/S021779/1.

Competing Interests

Daniel S. Katz and Neil Chue Hong are Academic Editors for PeerJ Computer Science. Daniel S. Katz is also a Section Editor.

Author Contributions

- Daniel S. Katz conceived and designed the experiments, authored or reviewed drafts of the article, and approved the final draft.
- Neil P. Chue Hong conceived and designed the experiments, authored or reviewed drafts of the article, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

This is an Editorial.

REFERENCES

Cadwallader L, Hrynaszkiewicz I. 2022. A survey of researchers' code sharing and code reuse practices, and assessment of interactive notebook prototypes. *PeerJ* **10(103)**:e13933 DOI 10.7717/peerj.13933.

Carver JC, Gesing S, Katz DS, Ram K, Weber N. 2018. Conceptualization of a US research software sustainability institute (URSSI). *Computing in Science & Engineering* **20(3)**:4–9 DOI 10.1109/MCSE.2018.03221924.

Chue Hong NP, Allen A, Gonzalez-Beltran, de Waard A, Smith AM, Robinson C, Jones C, Bouquin D, Katz DS, Kennedy D, Ryder G, Hausman J, Hwang L, Jones MB, Harrison M, Crosas M, Wu M, Lowe P, Haines R, Edmunds S, Stall S, Swaminathan S, Druskat S, Crick T, Morrell T, Pollard T. 2019a. Software citation checklist for developers. *Zenodo*. *Available at https://doi.org/10.5281/zenodo.3482769*.

Chue Hong NP, Allen A, Gonzalez-Beltran A, de Waard A, Smith AM, Robinson C, Jones C, Bouquin D, Katz DS, Kennedy D, Ryder G, Hausman J, Hwang L, Jones MB, Harrison M, Crosas M, Wu M, Lowe P, Haines R, Edmunds S, Stall S, Swaminathan S, Druskat S, Crick T, Morrell T, Pollard T. 2019b. Software citation checklist for authors. *Available at https://doi.org/10.5281/zenodo.3479199*.

Chue Hong NP, Katz DS, Barker M, Lamprecht A-L, Martinez C, Psomopoulos FE, Harrow J, Castro LJ, Gruenpeter M, Martinez PA, Honeyman T, Struck A, Lee A, Loewe A, van Werkhoven B, Jones C, Garijo D, Plomp E, Genova F, Shanahan H, Leng J, Hellstrom M, Sandstrom M, Sinha M, Kuzak M, Herterich P, Zhang Q, Islam S, Sansone S-A, Pollard T, Atmojo UD, Williams A, Czerniak A, Niehues A, Fouilloux AC, Desinghu B, Goble C, Richard C, Gray C, Erdmann C, Nust D, Tartarini D, Ranguelova E, Anzt H, Todorov I, McNally J, Moldon J, Burnett J, Garrido-Sanchez J, Belhajjame K, Sesink L, Hwang L, Tovani-Palone MR, Wilkinson MD, Servillat M, Liffers M, Fox M, Miljkovic N, Lynch N, Martinez Lavanchy P, Gesing S, Stevens S, Martinez Cuesta S, Peroni S, Soiland-Reyes S, Bakker T, Rabemanantsoa T, Sochat V, Yehudi Y, RDA FAIR4RS WG. 2022. FAIR principles for research software (FAIR4RS Principles). *Zenodo*. *Available at https://doi.org/10.15497/RDA00068*.

Crouch S, Chue Hong N, Hettrick S, Jackson M, Pawlik A, Sufi S, Carr L, De Roure D, Goble C, Parsons M. 2013. The software sustainability institute: changing research software attitudes and practices. *Computing in Science & Engineering* **15(6)**:74–80 DOI 10.1109/MCSE.2013.133.

Druskat S, Spaaks JH, Chue Hong N, Haines R, Baker J, Bliven S, Willighagen E, Pérez-Suárez, David, Konovalov O. 2021. Citation file format. *Zenodo*. *Available at https://doi.org/10.5281/zenodo.1003149*.

Du C, Cohoon J, Lopez P, Howison J. 2022. Understanding progress in software citation: a study of software citation in the cord-19 corpus. *PeerJ Computer Science* **8(1)**:e1022 DOI 10.7717/peerj-cs.1022.

European Commission and Directorate–General for Research and Innovation. 2020. *Scholarly infrastructures for research software—report from the EOSC executive board working group (WG) architecture task force (TF) SIRS*. Brussels: Publications Office of the European Union. *Available at https://data.europa.eu/doi/10.2777/28598*.

Frachtenberg E. 2022. Research artifacts and citations in computer systems papers. *PeerJ Computer Science* **8(7604)**:e887 DOI 10.7717/peerj-cs.887.

Garijo D, Ménager Hé, Hwang L, Trisovic A, Hucka M, Morrell T, Allen A, Best Practices for Software Registries TF, Consortium S. 2022. Nine best practices for research software registries and repositories. *PeerJ Computer Science* **8(1)**:e1023 DOI 10.7717/peerj-cs.1023.

JATS4R. 2021. NISO JATS4R software citations v1.0. *Available at https://doi.org/10.3789/niso-rp-40-2021*.

Jones MB, Boettiger C, Cabunoc Mayes A, Smith A, Slaughter P, Niemeyer K, Gil Y, Fenner M, Nowak K, Hahnel M, Coy L, Allen A, Crosas M, Sands A, Chue Hong N, Cruse P, Katz DS, Goble C. 2017. CodeMeta: an exchange schema for software metadata. version 2.0. KNB data repository. *Available at https://doi.org/10.5063/schema/codemeta-2.0*.

Katz DS, Bouquin D, Hong NPC, Hausman J, Jones C, Chivvis D, Clark T, Crosas M, Druskat S, Fenner M, Gillespie T, Gonzalez-Beltran A, Gruenpeter M, Habermann T, Haines R, Harrison M, Henneken E, Hwang L, Jones MB, Kelly AA, Kennedy DN, Leinweber K, Rios F, Robinson CB, Todorov I, Wu M, Zhang Q. 2019. Software citation implementation challenges. *ArXiv* DOI 1048550/arXiv.1905.08674.

Katz DS, Chue Hong NP, Clark T, Muench A, Stall S, Bouquin D, Cannon M, Edmunds S, Faez T, Feeney P, Fenner M, Friedman M, Grenier G, Harrison M, Heber J, Leary A, MacCallum C, Murray H, Pastrana E, Perry K, Schuster D, Stockhause M, Yeston J. 2021. Recognizing the value of software: a software citation guide [version 2; peer review: 2 approved]. *F1000Research* **9**:1257 DOI 10.12688/f1000research.26932.2.

Morrissey SM. 2020. *Preserving software: motivations, challenges and approaches*. Glasgow: Digital Preservation Coalition.

NISO. 2021. Reproducibility badging and definitions. *Available at https://doi.org/10.3789/niso-rp-31-2021*.

Ram DK, Howison DJ. 2023. Research software visibility infrastructure priorities report. *Available at https://doi.org/10.5281/zenodo.10060255*.

Research Data Alliance/FORCE11 Software Source Code Identification WG, Allen A, Bandrowski A, Chan P, Di Cosmo R, Fenner M, Garcia L, Gruenpeter M, Jones CM, Katz DS, Kunze J, Schubotz M, Todorov IT. 2020. Use cases and identifier schemes for persistent software source code identification (v1.0). *Research Data Alliance* 1–42 DOI 10.15497/RDA00053.

Schindler D, Bensmann F, Dietze S, Kruger F. 2022. The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central. *PeerJ Computer Science* **8(1)**: e835 DOI 10.7717/peerj-cs.835.

Smith AM, Katz DS, Niemeyer KE, FORCE11 Software Citation Working Group. 2016. Software citation principles. *PeerJ Computer Science* **2(2)**:e86 DOI 10.7717/peerj-cs.86.

Software Citation Implementation Working Group. 2023. FORCE11 software citation implementation working group home page. GitHub. *Available at https://github.com/force11/force11-sciwg*.

Stall S, Bilder G, Cannon M, Chue Hong N, Edmunds S, Erdmann CC, Evans M, Farmer R, Feeney P, Friedman M, Giampoala M, Hanson RB, Harrison M, Karaiskos D, Katz DS, Letizia V, Lizzi V, MacCallum C, Muench A, Perry K, Ratner H, Schindler U, Sedora B, Stockhause M, Townsend R, Yeston J, Clark T. 2023. Journal production guidance for software and data citations. *Scientific Data* **10(1)**:656 DOI 10.1038/s41597-023-02491-7.

Task Force on Best Practices for Software Registries, Monteil A, Gonzalez-Beltran A, Ioannidis A, Allen A, Lee A, Bandrowski A, Wilson BE, Mecum B, Du CF, Robinson C, Garijo D, Katz DS, Long D, Milliken G, Menager H, Hausman J, Spaaks JH, Fenlon K, Vanderbilt K, Hwang L, Davis L, Fenner M, Crusoe MR, Hucka M, Wu M, Hong NC, Teuben P, Stall S, Druskat S, Carnevale T, Morrell T. 2020. Nine best practices for research software registries and repositories: a concise guide. *ArXiv* DOI 1048550/arXiv.2012.13117.

PeerJ Computer Science

The role of software in science: a knowledge graph-based analysis of software mentions in PubMed Central

David Schindler[1], Felix Bensmann[2], Stefan Dietze[2,3] and Frank Krüger[1,4]

[1] Institute of Communications Engineering, University of Rostock, Rostock, Germany
[2] GESIS - Leibniz Institute for the Social Sciences, Cologne, Germany
[3] Heinrich-Heine-University, Düsseldorf, Germany
[4] Department Knowledge, Culture & Transformation, University of Rostock, Rostock, Germany

ABSTRACT

Science across all disciplines has become increasingly data-driven, leading to additional needs with respect to software for collecting, processing and analysing data. Thus, transparency about software used as part of the scientific process is crucial to understand provenance of individual research data and insights, is a prerequisite for reproducibility and can enable macro-analysis of the evolution of scientific methods over time. However, missing rigor in software citation practices renders the automated detection and disambiguation of software mentions a challenging problem. In this work, we provide a large-scale analysis of software usage and citation practices facilitated through an unprecedented knowledge graph of software mentions and affiliated metadata generated through supervised information extraction models trained on a unique gold standard *corpus* and applied to more than 3 million scientific articles. Our information extraction approach distinguishes different types of software and mentions, disambiguates mentions and outperforms the state-of-the-art significantly, leading to the most comprehensive *corpus* of 11.8 M software mentions that are described through a knowledge graph consisting of more than 300 M triples. Our analysis provides insights into the evolution of software usage and citation patterns across various fields, ranks of journals, and impact of publications. Whereas, to the best of our knowledge, this is the most comprehensive analysis of software use and citation at the time, all data and models are shared publicly to facilitate further research into scientific use and citation of software.

Subjects Data Mining and Machine Learning, Data Science, Digital Libraries, Natural Language and Speech, World Wide Web and Web Science
Keywords Knowledge graph, Software mention, Named entity recognition, Software citation

Submitted 8 October 2021
Accepted 7 December 2021
Published 14 January 2022

Corresponding authors
David Schindler,
david.schindler@uni-rostock.de
Frank Krüger,
frank.krueger@uni-rostock.de

Academic editor
Sedat Akleylek

Additional Information and Declarations can be found on page 42

DOI **10.7717/peerj-cs.835**

INTRODUCTION

Science across all disciplines has become increasingly data-driven, leading to additional needs with respect to software for collecting, processing and analyzing data. Hence, transparency about software used as part of the scientific process is crucial to ensure reproducibility and to understand provenance of individual research data and insights. Knowledge about the particular version or software development state is a prerequisite for reproducibility of scientific results as even minor changes to the software might impact them significantly.

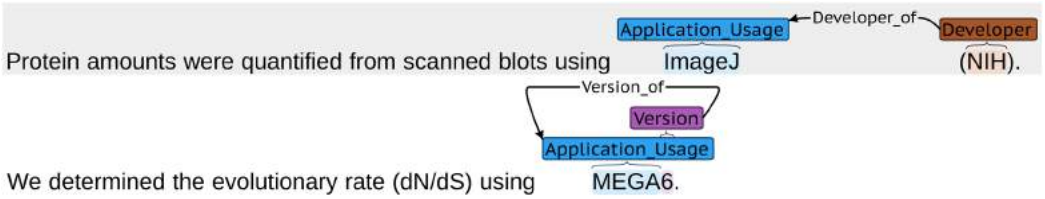


Figure 1 Annotated sentences from SoMeSci missing information required by software citation standards. Full-size DOI: 10.7717/peerj-cs.835/fig-1

Furthermore, from a macro-perspective, understanding software usage, varying citation habits and their evolution over time within and across distinct disciplines can shape the understanding of the evolution of scientific disciplines, the varying influence of software on scientific impact and the emerging needs for computational support within particular disciplines and fields. Initial efforts are made to provide publicly accessible datasets that link open access articles to respective software that is used and cited, for instance, the OpenAIRE Knowledge Graph (*Manghi et al., 2019*) or SoftwareKG (*Schindler, Zapilko & Krüger, 2020*). Given the scale and heterogeneity of software citations, robust automated methods are required, able to detect and disambiguate mentions of software and related metadata.

Despite the existence of software citation principles (*Smith, Katz & Niemeyer, 2016*; *Katz et al., 2021*), software mentions in scientific articles are usually informal and often incomplete—information about the developer or the version are often missing entirely, see Fig. 1. Spelling variations and mistakes for software names, even common ones (*Schindler, Zapilko & Krüger, 2020*), increase the complexity of automatic detection and disambiguation. Training and evaluation of information extraction approaches requires reliable ground truth data of sufficient size, raising the need for manually annotated gold standard corpora of software mentions.

Most works concerned with recognition of software mentions in scientific articles apply manual analysis on small corpora in order to answer specific questions (*Howison & Bullard, 2016*; *Nangia & Katz, 2017*) or are limited to specific software (*Li, Lin & Greenberg, 2016*; *Li, Yan & Feng, 2017*). Automatic methods, enabling large scale analysis, have been implemented by iterative bootstrapping (*Pan et al., 2015*) as well as machine learning on manually engineered rules (*Duck et al., 2016*). However, both achieve only moderate performance. Extraction through deep learning with a Bi-LSTM-CRF (*Schindler, Zapilko & Krüger, 2020*) shows promise, but requires sufficient and reliable ground truth data which only recently became available.

Available corpora (*Duck et al., 2016*; *Schindler, Zapilko & Krüger, 2020*; *Du et al., 2021*) do not cover all available metadata features, cater for disambiguation of different spelling variations of the same software or distinguish between the purpose of the mention such as creation or usage. In SoMeSci (*Schindler et al., 2021b*), we have introduced a gold standard knowledge graph of software mentions in scientific articles. To the best of our

knowledge, SoMeSci is the most comprehensive gold standard *corpus* of software mentions in scientific articles, created by manually annotating 3,756 software mentions with additional information about types of software, mentions and related features, resulting in 7,237 labeled entities in 47,524 sentences from 1,367 PMC articles.

In this work, we provide a large-scale analysis of software usage and citation practices facilitated through an unprecedented knowledge graph of software mentions and affiliated metadata generated through a supervised information extraction model trained on SoMeSci and applied to more than 3 million scientific articles. In summary, our contributions include:

- **A large-scale analysis of software usage** across 3,215,386 scholarly publications covering a range of diverse fields and providing unprecedented insights into the evolution of software usage and citation patterns across various domains, distinguishing between different types of software, mentions as well as rank of journals and impact of publications. Results indicate strongly discipline-specific usage of software and an overall increase in software adoption. To the best of our knowledge, this is the most comprehensive analysis of software use and citation at the time.
- **A comprehensive knowledge graph of software citations** in scholarly publications comprising of 301,825,757 triples describing 11.8 M software mentions together with types and additional metadata. The knowledge graph is represented using established vocabularies capturing the relations between citation contexts, disambiguated software mentions and related metadata and provides a unique resource for further research into software use and citation pattern.
- **Robust supervised information extraction models for disambiguating software mentions and related knowledge** in scholarly publications. As part of our experimental evaluation, our model based on SciBERT and trained on SoMeSci *Schindler et al. (2021b)* for NER and classification outperforms state-of-the-art methods for software extraction by 5 pp on average. Software mentions are disambiguated and different variations interlinked, *e.g.*, abbreviations and name- and spelling-alternatives, of the same software.

Through these contributions, we advance the understanding of software use and citation practices across various fields and provide a significant foundation for further large-scale analysis through an unprecedented dataset as well as robust information extraction models.

The remaining paper is organized as follows. Related work is discussed in the following section, whereas the *Methods and Materials* introduces developed information extraction methods together with datasets used for training and testing. *Results: Information Extraction Performance* describes the performance results obtained on the various information extraction tasks, while the *Results: Analysis of Software Mentions* introduces an in-depth analysis of the extracted data. Key findings are discussed in the *Discussion*, followed by a brief conclusion and introduction of future work.

Table 1 Summary of investigations concerning software in science together with source of the articles, number of articles and software, and a quality indicator. Level of extracted details varies between listed approaches. Note that PLoS is a subset of PMC. M, manual; A, automatic; k, Cohen's; F, FScore; O, Percentage Overlap.

	Approach	Quality	Source	Articles	Software
M	*Howison & Bullard (2016)*	O = 0.68–0.83	Biology	90	286
	Nangia & Katz (2017)	–	Nature (Journal)	40	211
	Du et al. (2021)	O = 0.76	PMC, Economics	4,971	4,093
	Schindler et al. (2021b)	κ = 0.82, F = 0.93	PMC	1,367	3,756
A	*Pan et al. (2015)*	F = 0.58	PLoS ONE	10 K	26 K
	Duck et al. (2016)	F = 0.67	PMC	714 K	3.9 M
	Schindler, Zapilko & Krüger (2020)	F = 0.82	PLoS (Social Science)	51 K	133 K

RELATED WORK

Requirements for large scale software citation analyses

Software mentions in scientific articles have been analyzed for several reasons including mapping the landscape of available scientific software, analyses of software citation practices and measuring the impact of software in science (*Krüger & Schindler, 2020*). This includes manual analyses based on high quality data, such as *Howison & Bullard (2016)*, *Du et al. (2021)*, *Nangia & Katz (2017)* and *Schindler et al. (2021b)* but also automatic analyses such as *Pan et al. (2015)*, *Duck et al. (2016)* and *Schindler, Zapilko & Krüger (2020)*. While manual analyses provide highly reliable data, results often only provide a small excerpt and do not generalize due to small sample size. Analyses based on automatic data processing, in contrast, allow to make more general statements, for instance, regarding trends over time or across disciplines, but require high quality information extraction methods which themselves rely on reliable ground truth labels for supervised training. Table 1 compares manual and automatic approaches with respect to sample size and quality indicators such as IRR or FScore. Manual approaches provide substantial to almost perfect IRR, but are restricted to less than 5,000 articles at most. *Howison & Bullard (2016)*, for instance, analyzed software mentions in science by content analysis in 90 articles. The main objective of *Du et al. (2021)* and *Schindler et al. (2021b)* was to create annotated corpora of high quality for supervised learning of software mentions in scientific articles. *Du et al. (2021)* provide labels for software, version, developer, and URL for articles from PMC, which is multidisciplinary but strongly skewed towards Medicine (see Table A11) and Economics. *Schindler et al. (2021b)* exclusively used articles from PMC, and provide labels for software, a broad range of associated information, software type, mention type, and for disambiguation of software names.

Early automatic approaches, such as *Pan et al. (2015)* and *Duck et al. (2016)* achieve only moderate recognition performance of 0.58 and 0.67 FScore, but perform analyses on up to 714 K articles raising doubts about the reliability and generalizability of the described results. *Pan et al. (2015)* used iterative bootstrapping—a rule-based method that learns context rules—as well as a dictionary of software names based on an initial set of seed

names. *Duck et al. (2016)* employ machine learning classifiers on top of manually engineered rules. With the availability of large language models and deep learning methods for sequence labeling, *Schindler, Zapilko & Krüger (2020)* employed a Bi-LSTM-CRF and achieved an FScore of 0.82 for the recognition of software mentions in scientific articles. Most recently, *Lopez et al. (2021)* compare Bi-LSTM-CRF and SciBERT-CRF models on Softcite (*Du et al., 2021*) software entity recognition at paragraph level. They achieve 0.66 and 0.71 FScore, respectively, and further improved performance to 0.74 FScore by linking entities to Wikidata during postprocessing.

Beside high recognition rates, and thus the basis for reliable statements, *Schindler, Zapilko & Krüger (2020)* demonstrate the capabilities of semantic web technologies for information structuring and data integration with respect to analyzing software usage. They provide a KG—SoftwareKG—representing a source for structured data access for analyses. Moreover, the performed disambiguation of software mentions allows to draw conclusion on the level of software rather than software mentions, even with spelling variations. Finally, the linked nature of KG allows the integration of external data sources enabling further analyses. Following the direction of *Schindler, Zapilko & Krüger (2020)*, large scale analyses of software mentions in scholarly articles requires (1) robust information extraction and disambiguation techniques that achieve results on the level of manual approaches, and (2) the provision of all data in a standardized way that allows the reuse and the integration of external knowledge.

Previous analyses of software in scholarly publication

As described above, previous studies on software mentions in scholarly publication were based on high quality manual analyses with small sample sizes or automatic analyses with large sample size but moderate quality. Most studies report basic descriptive statistics such as the number of overall mentions given in Table 1 or the distribution of software mentions over different software. *Howison & Bullard (2016)* report an average of 3.2 software mentions per article in Biology while *Duck et al. (2016)* report 12.9. In PMC, *Duck et al. (2016)* report an average of 5.5 mentions while *Du et al. (2021)* report 1.4 and *Schindler et al. (2021b)* 2.6. Similarly, *Pan et al. (2015)* and *Schindler, Zapilko & Krüger (2020)* report values of 2.7 and 2.6 for sub-selections of PLoS. Interestingly, *Du et al. (2021)* report a low value of 0.2 for Economics and *Duck et al. (2016)* a high value of 30.8 for Bioinformatics. Some of those results clearly show disciplinary differences, while others such as the PMC discrepancies might be attributed to methodical differences, for instance, publication time of articles in the investigated sets. Articles within *Du et al. (2021)* are significantly older than articles in *Schindler et al. (2021b)* which could result in a lower average software usage. This is also supported by the finding of *Duck et al. (2016)* who analyze software mentions up to 2013 and report a rapid increase in software usage between 2000 and 2006.

Other findings regard the distribution with respect to unique software names. *Pan et al. (2015)* report that 20% of software names account for 80% of mentions. *Duck et al. (2016)* report that 5% of software names account for 47% of mentions, and, similarly, 6.6% of entities are responsible for 50% of mentions in *Schindler et al. (2021b)*. Therefore, all prior

studies agree that the distribution of software within articles is highly skewed, pointing towards the fact that there are few pieces of general purpose software such as SPSS or R that support the scientific infrastructure. On the other hand, there is a high number of rarely mentioned software that is likely to be highly specialized towards problems and domains. *Duck et al. (2016)* perform an analysis of domain specific software to investigate disciplinary differences in software usage. They were able to confirm the existence of domain specific software and showed, for instance, that 65% of software used in medicine was not used in other analyzed domains. They also analyze journal specific software and applied a clustering analysis with respect to journal and software names.

Completeness of software mentions and citations is of high importance since employed software can only be clearly identified with sufficient information. Providing information such as the specific version or developer of software is, therefore, essential for provenance of study results or to provide credit for the creation of scientific software. For this purpose, guidelines for proper software citation have been established (*Smith, Katz & Niemeyer, 2016*; *Katz et al., 2021*) that recommend the following information to be included: name, author, version/release/date, location, venue, and unique ID, *e.g.*, DOI. *Howison & Bullard (2016)* analyze the completeness of software mentions with respect to formal citation 44%, version 28%, developer 18% and URL 5%. Based on the given information they were able to locate 86% of software online, but only 5% with the specific version. Completeness analyses by *Du et al. (2021)* showed that a total of only 44% of software mentions include further information with version being included in 27%, publisher in 31%, and URL in 17%. An analysis by *Schindler et al. (2021b)* showed that 39% mentions included a version, 23% a developer, 4% a URL and 16% a formal citation. Overall, the studies show that software mentions are still often informal and incomplete, but exhibit some notable differences between reported values. The problem of formal and informal software citation was also included in the automatic analysis of *Pan et al. (2015)* who identified formal citations for recognized software by automatic string pattern matching. They report a correlation between the number of mentions of a software and its formal citation frequency.

Availability of used software is crucial as studies conducted with commercial software might not be reproducible by other research teams. Furthermore, implementation details for non open source software cannot be reviewed by the scientific community and can potentially bias study results. Therefore, different studies included analyses regarding commercial, free and open source software usage. *Pan et al. (2015)* found that of the most frequent software mentions, which were labeled for availability manually, 64% are free for academic use. Moreover, they found that free software received more formal citations than commercial software. *Howison & Bullard (2016)* include an analysis for accessibility, license and source code availability and report that commercial software is more likely to be mentioned similar to scientific instruments (including details on developer and its location) while open access software is more often attributed with formal citations. However, they note that there is no overall preferred style for any group of software. *Schindler, Zapilko & Krüger (2020)* show a comparison of software mention

numbers for free, open source and commercial software over time that showed no clear trend towards a specific group.

Beside analyses about software in scholarly publications in general, several studies focus on particular aspects such as specific software or the relation of software usage to bibliometric measures. *Li, Lin & Greenberg (2016)*, analyze mentions of the specific engineering software (LAMMPS) and found that the given information is often not complete enough to determine how it was applied with respect to version, but also regarding software specific settings. *Li, Yan & Feng (2017)* analyze software citation for R and R packages. They report inconsistency resulting from a variety in citation standards, which are also not followed well by authors. Overall, they show a trend towards more package mentions, and find a comparably high number of formal citations for R packages (72%). *Mayernik et al. (2017)* discuss data and software citation and conclude that there is no impact measure for software available. *Allen, Teuben & Ryan (2018)* analyse the availability of source code in astrophysics and report that it could only be located for 58% of all mentions. *Pan et al. (2018)* analyze the completeness for usage statements of three specific bibliometric mapping tools and find provided versions in 30% of cases, URLs in 24%, and formal citations in 76%. They argue that the high formal citation might be due to good author citation instruction given by the tools. *Howison & Bullard (2016)* report that articles published in high impact journals mention more software. The platform swMATH (*Greuel & Sperber, 2014*) aims to establish a mapping of software used in mathematical literature by manually labeling software present in zbMATH articles pre-filtered through an automatic, heuristic search.

Most studies agree that software citation is important but often incomplete and report similar trends about the frequency of software mentions. They deviate, however, when it comes to particular numbers such as software mentions per article. This could be the result of (1) discipline specific citation habits, (2) small sample sizes in analysis studies, and (3) insufficient quality of automatic information extraction. A large scale study based on reliable automatic information extraction is required to draw conclusions across different disciplines.

METHODS AND MATERIALS

Information extraction

Training dataset

We apply automatic information extraction based on supervised machine learning for recognizing software in science and use SoMeSci—Software Mentions in Science—a *corpus* of annotated software mentions in scientific articles (*Schindler et al., 2021b*). It contains 3,756 software annotations in 1,367 PubMed Central (PMC) articles as well as annotations for different software types such as *Programming Environment* or *Plug-In*, mention types such as *Usage* or *Creation*, and additional information such as *Version* or *Developer*. Moreover, it provides unique entity identities for all software annotations, which allows to not only develop a system for software name recognition but also for disambiguating

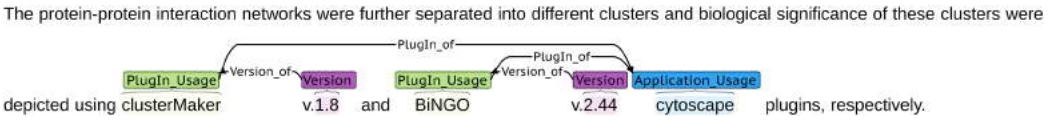


Figure 2 Sentence from SoMeSci annotated with respect to software, additional information, mention type, and software type as well as corresponding relations.
Full-size DOI: 10.7717/peerj-cs.835/fig-2

Table 2 Overview of the SoMeSci *corpus*. Further details can be found in *Schindler et al. (2021b)*.

SoMeSci statistics	
# Articles	1,367
# Sentences w/ Software	2,728
# Sentences w/o Software	44,796
# Annotations	7,237
# Software	3,756
# unique Software	883
# Relations	3,776
Software Type	*Application, PlugIn, Operating System (OS), Programming Environment (PE)*
Mention Type	*Allusion, Usage, Creation, Deposition*
Additional Information	*Developer, Version, URL, Citation, Extension, Release, License, Abbreviation, Alternative Name*

names, an essential inference step in building a software Knowledge Graph. This level of detail is not represented in other available software datasets such as BioNerDs (*Duck et al., 2016*) or Softcite (*Du et al., 2021*). SoMeSci does also contain recent articles and is, therefore, suited to represent the recent shift in awareness and recommendations for software citation. Quality of SoMeSci annotations was assessed through IRR and is reported to be high with a value of $\kappa = 0.82$. SoMeSci is available from Zenodo (https://doi.org/10.5281/zenodo.4968738) and an annotated example with markup from the web-based annotation tool BRAT (*Stenetorp et al., 2012*) is given in Fig. 2. For all reported information extraction problems described below we use the same 60:20:20 division in train, development, and test set as the SoMeSci baseline.

An overview of the different annotations along with the overall statistics of the SoMeSci dataset is given in Table 2. SoMeSci distinguishes each mention of a software by two types: mention and software. Mention type can take the values of *Usage* if the software was actively used and is contributing to the articles results, *Creation* if it was created in the scope of the article, *Deposition* if it was created and additionally published, and *Allusion* if its name was merely stated, *e.g.*, in an comparison with another software. Similarly, software type is distinguished between, *Application* if the software can be run as a stand-alone software, *PlugIn* if it is an extension to an existing host software, *Operating System* and *Programming Environment* if it is a framework for writing and executing program code. More details on the different types and relations are provided in the *Taxonomy for Software and Related Information*.

Inference dataset

The inference dataset includes 3,215,386 articles indexed in PMC acquired *via* bulk download (https://www.ncbi.nlm.nih.gov/pmc/tools/ftp/). on January 22, 2021. Construction of SoftwareKG requires metadata and plain text of each article. To acquire the information, JATS was used instead of the also available Portable Document Format (PDF). PDF is the standard form in which humans consume scientific articles, however, there are drawbacks for machines due to formatting artifacts caused by elements such as headers, footers, page numbering, or multi-column formats. While some tools, such as *GROBID (2021)*, perform well on pdf to text conversion, using JATS prevents errors resulting from text formatting. JATS on the other hand is an XML-based format, and while specific tagging conventions vary between different journals indexed in PMC, they all follow a common scheme, making it a suited source for both metadata and plain text. Both were extracted using a custom implementation available in the associated source code (https://github.com/dave-s477/SoftwareKG).

Entity recognition and classification

The objective of this information extraction step is to recognize software mentions and associated additional information, and to classify software according to its *Software Type* and *Mention Type*. The target labels are summarized in Table 2. The task is modelled as an NER sequence tagging problem where each sentence is considered as a sequence of tokens each of which has to be assigned a correct output tag.

Different suited state-of-the-art machine learning models are considered for the task. We compare the given baseline results on SoMeSci *Schindler et al. (2021b)*, which were established by an un-optimized Bi-LSTM-CRF model, to other machine learning models suited for scientific literature, for instance, SciBERT (*Beltagy, Lo & Cohan, 2019*). To establish a consistent naming scheme we label all implemented and tested models by *type*, classification *target* and *optimization* state: $M_{type,target,optimization}$. Results for NER are reported by mean and standard deviation for repeated training runs because performance can vary between runs due to randomization in initialization and training. Results of at least 4 different training runs are provided for hyper-parameter optimization and 16 for final performance estimation. The best model is selected on the problem of identifying software mentions ($M_{-,sw,-}$) as we consider it the most important quality measure and the main problem all other tasks relate to.

Bi-LSTM-CRFs ($M_{L,sw,-}$) were selected as they are well established for NER and have been reported to achieve state-of-the-art results (*Ma & Hovy, 2016*). Further, they have previously been applied to the problem of recognizing software in scientific literature (*Schindler, Zapilko & Krüger, 2020*; *Schindler et al., 2021b*; *Lopez et al., 2021*). More details on the model can be found in *Ma & Hovy (2016)*, *Schindler, Zapilko & Krüger (2020)* as well as in the implementation details in our published code.

BERT (*Devlin et al., 2019*) is a transformer-based model that is pre-trained on a masked language prediction task and has proven to achieve state-of-the-art performance across a wide range of NLP problems after fine-tuning. Different adaptions of the BERT pre-training procedure exist for scientific literature resulting in the two well established

Table 3 Hyper-parameters considered for BERT models including their default setting.

Parameter	Default
Learning Rate (LR)	1e−5
Sampling	all data
Dropout	0.1
Gradient Clipping	1.0

models BioBERT (*Lee et al., 2019*) ($M_{BB,sw,-}$) and SciBERT (*Beltagy, Lo & Cohan, 2019*) ($M_{SB,sw,-}$). While BioBERT is pre-trained on PubMed abstracts as well as PMC full-texts SciBERT is pre-trained on full-text articles from semantic scholar with 18% of articles coming from the domain of Computer Science and the remaining 82% from Biomedicine. To reduce run-time requirements, hyper-parameter optimization was only performed for the best performing BERT model that was chosen by comparing both models after fine-tuning with the default configuration summarized in Table 3. The parameter *Sampling* reduces the size of the training set by randomly suppressing sentences from the training *corpus* that do not contain software.

The overall, best model based on the development set is selected and extended to solve the 3 main objectives ($M_{-,sw+info,-}$) of the initial information extraction step: (1) recognize software mentions and corresponding additional information, (2) classify software type, (3) classify mention type of extracted software mentions. The combined problem is modeled as hierarchical multi-task sequence labeling and illustrated in Fig. 3. Multi-task learning can improve recognition performance and help to learn better representations if the given tasks are related as it implicitly increases the sample size (*Ruder, 2017*). Therefore, the main layers of the model share their weights across all sub-tasks and are updated with loss signals from all individual tasks. The output of each sub-task is calculated by a separate fully connected layer with softmax activation. For backpropagation we chose the simple approach of summing over the three cross-entropy losses, however, this could be further explored in the future, for instance, as described by *Kendall, Gal & Cipolla (2018)*.

The hierarchical component is added by passing the classification result of lower hierarchy sub-tasks as input to higher sub-tasks. The classification layer for mention type receives the output of software recognition and the software type layer the output of both software recognition and mention type. There is no gradient passed backward through the hierarchy so the weight updates in each classification layer are only based on the individual task loss. Teacher forcing—passing the correct prediction regardless of the actual prediction—is performed during training with respect to the output of lower layers in the hierarchy. As a result, we expect better update steps and faster learning convergence by providing more gold label information to higher classification layers. Additionally, teacher forcing should motivate the constraint that a software type or mention type should only be classified if a software was classified before. Note, that hyper-parameters for the $M_{-,sw+info,opt}$ are based on the best set of parameters identified for software recognition $M_{-,sw,opt}$.

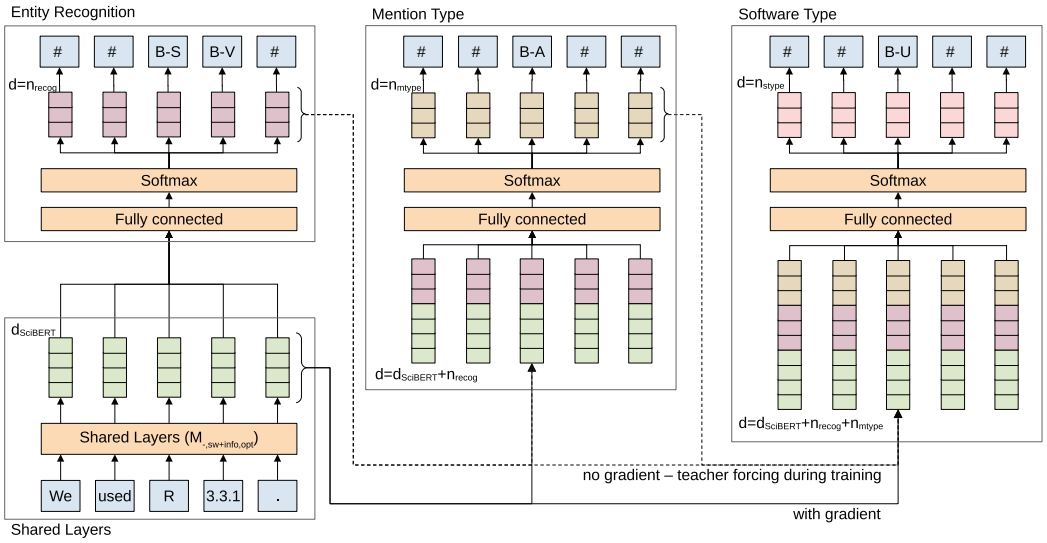


Figure 3 Illustration of the employed multi-task, hierarchical, sequence labeling model. Features are generated based on shared layers. The features are passed to 3 separate tasks and loss signals are summed to update shared weights. Outputs of classification layers are passed back to the network as input features to other classification layers, depicted from left to right in the image. Teacher forcing—replacing lower level classification outputs with gold label data—is used during training to stop potentially wrong classification outputs from being passed to other classification layers. Colors represent similar types of information.
Full-size DOI: 10.7717/peerj-cs.835/fig-3

Table 4 Example for enforcing tagging consistency. Inconsistencies are underlined.

Sentence	We	Used	SPSS	Statistics	16	.
Entities	O	O	B-App	I-App	I-Ver	O
Types	O	O	B-Use	I-Mention	O	O
Fixed	O	O	B-App-Use	B-App-Use	B-Ver	O

As labels for multiple tasks have to be combined with potential tagging inconsistencies for each task we experimented with adding a CRF layer on top of BERT to improve performance by learning inter-dependencies and constraints between labels. We found no improvement in performance but additional time complexity and did not further pursue the model. Instead, we enforce tagging consistencies by applying a simple set of rules: (1) all I-tags without leading B-tags are transformed to B-tags—including I-tags that do not match their leading B-tags; (2) entity boundaries for higher hierarchy tasks are adjusted to the base task entity boundaries; (3) when there are multiple conflicting labels in higher hierarchy steps for one identified software entity, the label for the first token is chosen. An example is given in Table 4.

The performance of $M_{\_,sw+info,opt}$ is evaluated against the SoMeSci baseline (*Schindler et al., 2021b*) described above. In contrast to our implementation, information is not shared between tasks in the baseline model. Instead, all classifications are performed hierarchically and individually. Therefore, the reported results for the baseline are

subject to error propagation as recognition of additional information, software type classification and mention type classification all assume an underlying perfect software recognition. As our implementation does take error propagation into account the SoMeSci baseline overestimates performance in a direct comparison.

Relation extraction

For Relation Extraction (RE), the task of classifying if and which relationships exist between entities, we considered all relations available from the training dataset. All additional information can be related to software, versions and developers to licenses, and URLs to licenses or developers. Software mentions can be related to each other by the *plugin-of* relation, representing one mention as the host software and the other as the PlugIn, or by the *specification-of* relation if both mentions refer to the same real world entity. Some possible relations are also depicted in Fig. 2. Its important to note that RE is the second information extraction step and, therefore, directly dependent on entity extraction. For developing and testing RE we rely on gold level entities, but in practice RE performance is expected to be lower due to false negatives and false positives resulting from entity extraction errors.

SoMeSci (*Schindler et al., 2021b*) provides a baseline model for classifying relations between software associated entities based on manually engineered features and an optimized Random Forest classifier. All features are implemented to yield Integer or Boolean results and take into account (1) entity order, (2) entity types, (3) entity length, (4) entity distance, (5) number of software entities, (6) sub-string relations, and (7) automatically generated acronyms.

We chose to adapt and enhance the SoMeSci baseline model instead of using more complex deep learning models because the baseline achieved good results. Moreover, RE for software associated entities is less challenging as general RE problems as we impose a large number of constraints on how entities can be related. To improve the given rule set we individually fine-tuned the implementation of each rule. Moreover, we experimented with multi-layer perceptrons and SVMs as alternative to the Random Forest classifier. In initial tests, they did not achieve better performance and we chose to retain the Random Forest classifier as it has the benefit of offering better explainability. The Random Forest was trained with 100 trees, unlimited maximum depth, and no restrictions to splitting samples.

Software disambiguation

Software is referred to by different names due to abbreviations, geographical differences, or time. *Schindler, Zapilko & Krüger (2020)*, for instance, report up to 179 different spelling variations for the commonly used software SPSS. This raises the need for software name disambiguation as a core requirement for constructing SoftwareKG. SoMeSci provides a gold standard for this problem through manually assigned unique identifiers in form of links to external knowledge bases. However, existing knowledge bases, such as Wikidata (*Vrandečić, 2012*) or DBpedia (*Auer et al., 2007*), are sparse when it comes to

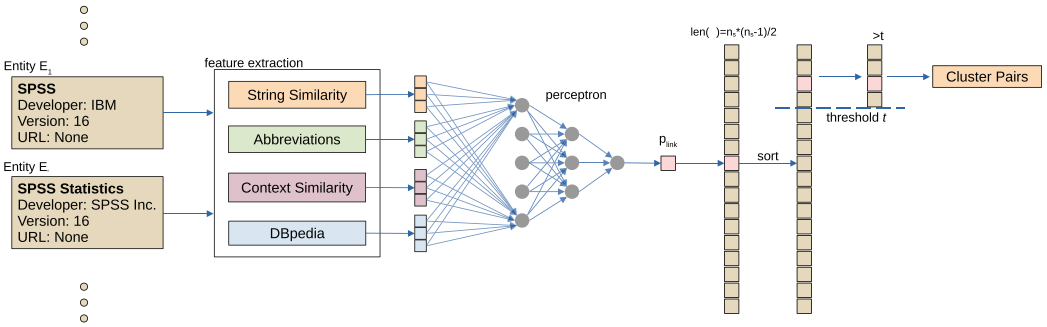


Figure 4 Overview of the software name disambiguation. For all pairs of extracted software entities (E_1, E_2), features are extracted (feature extraction) and used to determine a probability of linking (perceptron). Finally, agglomerative clustering is performed to cluster similar software names.
Full-size DOI: 10.7717/peerj-cs.835/fig-4

scientific software which is illustrated by an analysis of the SoMeSci disambiguation ground truth: only 205 of 883 (23%) unique and 2,228 of 3,717 (60%) software mentions are represented in Wikidata. Therefore, we adapt and develop an entity disambiguation method able to handle previously unknown software names such as those from creation statements without the need to link to external knowledge bases. In consequence, we contribute to establish a more complete KG of software.

The objective of software entity disambiguation is as follows: Given a pair (E_1, E_2) of software entities the goal is to determine whether they refer to the same real world entity. For that purpose we employ agglomerative clustering following the procedure illustrated in Fig. 4. First, manually engineered features are calculated for each pair, resulting in a feature vector $\mathbf{v}_{E1,E2}$. Features take into account: (1) string similarity, (2) similarity of extracted context information, (3) automatically generated abbreviations, and (4) software related information queried from DBpedia.

For each pair (E_1,E_2), vector $\mathbf{v}_{E1,E2}$ is mapped to a probability estimate p_{link} for if they should be linked by a 4-layer perceptron ($15 \times 10 \times 5 \times 1$) with low complexity $p_{link} = f_{perceptron}(\mathbf{v}_{E1,E2})$. The model is trained supervised to predict if a link exist $l = \{0,1\}$ based on splitting all possible combinations from the ground truth set in train, development and test set in a 60:20:20 ratio. Since the class was trained as a binary classification the output of the perceptron is the result of a sigmoid layer $d \in [0,1]$ and is used in combination with a threshold in the following steps. We considered applying dropouts but found a decreasing performance in initial tests. We also did not find any increase in performance for increasing model complexity.

For disambiguation we have to consider the influence of the sample size on the density of samples in the resulting features space. For n extracted mentions of software the number of entity pairs that need to be disambiguated accumulates to $n^2 - n$. In the small training set data points are less dense than in the large inference set. Moreover, the inference set does contain false positive mentions with strong resemblance to software resulting from prediction errors in the entity extraction step. This makes it difficult to find reliable decision boundaries on the training set alone. During testing it became apparent that due to the described effect the perceptron trained only on gold standard labels

could not learn suited decision boundaries to disambiguate entities pairs in the inference set. To counteract this problem, data augmentation was applied to add further entities resembling false positive extracted software names, which should not be linked to any other mentions. To simulate closeness to existing software names the new samples were generated by recombining sub-strings of existing samples, for instance, *ImageJ* and *SPSS Statistics* could be combined to form *Image Statistics*. During creation we made sure to not re-create given software names as well as duplicates. In total, $2n$ augmented samples were created once for the n original software mentions and included at each training epoch. They were also included in the test set with the same factor in order to estimate performance under the chance of false positive samples. As we only add negative samples to the test set there is no risk to overestimate the performance with the employed metrics of Precision and Recall.

Based on the predicted probabilities p_{link} for entity pairs a agglomerative clustering is performed. In each step, the two clusters with the largest probability are combined. As stopping criterion the threshold t is introduced and defined as the minimal probability for which pairs are linked. It is optimized based on the available gold standard labels. Here, the creation of reliable decision boundaries within the densely populated feature space is also an issue. To counteract it the threshold is optimized taking into account all available data points from gold standard and inference set by combining both sets. This approach allows to evaluate how well the gold labeled mentions are clustered within the densely populated feature space. The performance is estimated in terms of Precision, Recall and FScore at t.

We considered single and average linkage for clustering and found almost identical performance for varying thresholds based on gold standard mentions only. Given the similar performance during the initial tests, single linkage was preferred as it offers benefits in run-time and space complexity because it allows to re-use the initially calculated similarities. Average linking, in contrast, would require additional computation for the per-cluster-pair average similarity. Due to the run-time issues described below an evaluation of average linkage would not have been feasible with the evaluation method described above. Single linkage was then applied and evaluated as described.

A major issue we faced for disambiguation was run-time complexity as the number of pairs accumulates to $n^2 - n$ with $n > 11M$ software mentions. Therefore, we had to optimize for run-time complexity. Our initial optimization step was to assume symmetric feature vectors between entities E_1 and E_2 $\mathbf{v}_{E1,E2} = \mathbf{v}_{E2,E1}$ reducing the number of required compares to $\frac{n(n-1)}{2}$, even so they are not strictly symmetric because string length of entities are included as normalization factors. Further, we made the assumption that all software with the same exact string refers to the same real-world software entity and only included a limited number of $n_{unique} = 6$ samples of each name. The work of *Schindler et al. (2021b)* showed that this can in rare cases lead to false positive clustering, but in this case the benefit outweighs this risk because otherwise the computation would not have been feasible. Disambiguation on the remaining set of ~1.4 M mentions took approximately 6 days, with feature calculations parallelized over 6 Intel® Xeon® Gold 6248 CPUs (2.50 GHz, 40 Threads).

Schindler et al. (2021b) provide a baseline implementation for entity disambiguation on SoMeSci which uses manually engineered rules and external knowledge from DBpedia to disambiguate software names. For completeness we provide baseline results, however, as explained above, the density of the features space increases strongly by including additional data samples and our evaluation specifically includes augmented negative samples. Thus, the baseline cannot directly be compared to the implemented method in terms of disambiguation quality, but serves as an indicator.

SoftwareKG: knowledge graph of software mentions

Taxonomy for software and related information

We define software and its related information following the taxonomy presented by *Schindler et al. (2021b)* that describes the intricacies of in-text software mentions in scientific publications. The taxonomy distinguishes *Type of Software* describing which artifacts are considered as software, *Type of Mention* describing the context in which software was applied, and *Additional Information* that is provided to closer describe a software entity.

Type of software

Based on the distinction between end-user *application* (software) and *package* introduced by *Li, Yan & Feng (2017)*, *Schindler et al. (2021b)* distinguish the following categories of software:

Applications are standalone programs, designed for end-users, that usually result in associated data or project files, *e.g.*, Excel sheets. This definition includes software applications that are only hosted and available through web-based services, but excludes web-based collections of data. The definition also excludes databases that are used to store collections of scientific data. To be considered as an *application* a web-service has to provide functionality beyond filtered access to data.

Programming Environments (PE) are environments for implementing and executing computer programs or scripts. They are built around programming languages such as Python but also integrate compilers or interpreters in order to create executables from developed code. PEs play an important role in many scientific investigations and are particularly important for computationally heavy scientific disciplines such as computer science.

PlugIns are extensions specifically developed to be used with existing applications or PEs and cannot be used individually. As such, in the context of PEs, the category *PlugIn* could also be called package or library. Often, the original application can be concluded from the PlugIn, *e.g.*, scikit-learn is a frequently used Python package for machine learning. The usage of Plugins is well established in the scientific community as it allows to extend the function range of well established software libraries. This allows to implement custom software without the need to establish more complex stand-alone application.

Operating Systems (OS) build the basis for running software on a computer by managing its hardware components and the execution of all other software. OS are necessary when running a software application and they are, overall, less mentioned than other software. In many cases authors still choose to attribute common operating systems such as Windows, OS X, or Android as well as lesser used ones such as Ubuntu or Raspbian.

Type of mention

The definition of *Schindler et al. (2021b)* introduces a hierarchy of reasons why software is mentioned within scholarly articles based on the basic distinction between *mention* and *usage* introduced by *Howison & Bullard (2016)*:

Allusion of software describes each mention of a software name within a scholarly article. Aside appearance of the software name there are no further requirement for an allusion. It should especially be noted that no indication of actual usage is required, for instance, a fact about the software can be stated or multiple software can be compared. In the context of software mentions, allusions are comparable with scholarly citations used to refer to related work.

Usage (sub-type of Allusion) defines that a software made a contribution to a study and was actively used during the investigation, which makes the software a part of the research's provenance. Therefore, usage statements are required to allow conclusions regarding provenance. This is in line with the definition of software usage by *Lopez et al. (2021)*.

Creation (sub-type of Allusion) indicates that software was developed and implemented as part of a scientific investigation and is itself a research contribution. Knowledge of creation statements allows to track research software to its developers in order to provide credit to them as well as to discover and map newly published scientific software.

Deposition (sub-type of Creation) indicates that a software was published in the scope of a scientific investigation on top of being developed. In difference to creation statements, depositions require that authors provide either a URL to access the software or the corresponding publication license. Deposition statements, therefore, allow to provide additional information about discovered scientific software.

Both *Type of Software* and *Type of Mention* are required to fully describe a software mention in a scientific publication.

Additional Information and Declarations

Software is constantly updated and changing. Moreover, software names are ambiguous (*Schindler, Zapilko & Krüger, 2020*). Therefore, software citation principles (*Smith, Katz & Niemeyer, 2016*; *Katz et al., 2021*) have been established to precisely identify software in publications. They require that software mentions in scholarly articles are accompanied by additional information allowing the unique identification of the actually used software, information that is often missing in practice (*Howison & Bullard, 2016*; *Du et al., 2021*; *Schindler et al., 2021b*). Here we employ the following definitions for additional information about software as defined by *Schindler et al. (2021b)*. *Developer* describes the

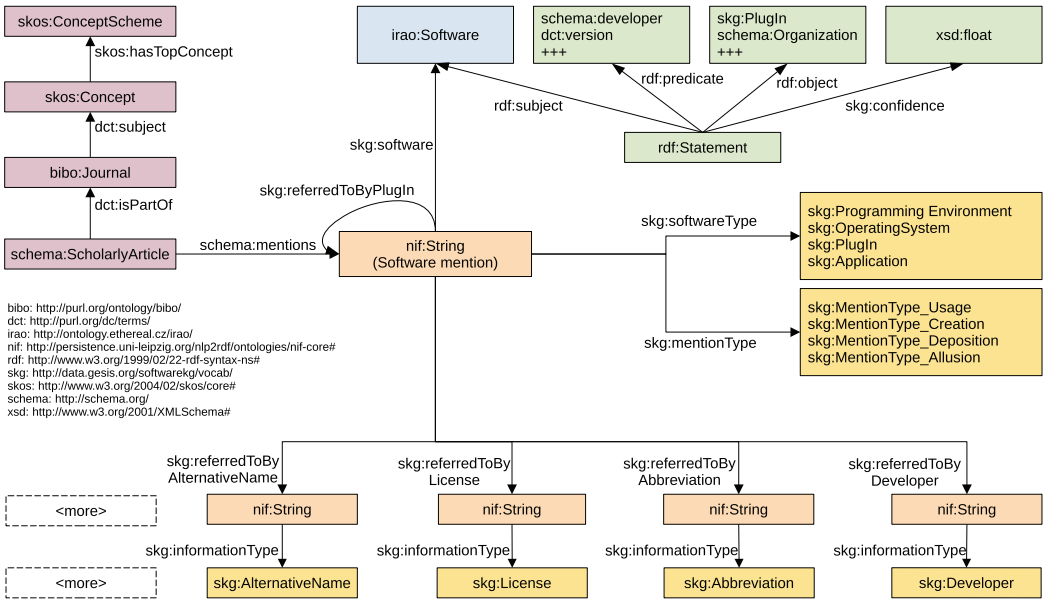


Figure 5 Data model of the Knowledge Graph representing extracted software mentions and their related information. For reasons of conciseness some details are left out.
Full-size DOI: 10.7717/peerj-cs.835/fig-5

person or organization that developed a software while *Version* indicates a defined state in the software life-cycle, typically identified by a version number, *Release* indicates a defined state in the software life-cycle by using a date based identifier, and *Extension* indicates different function ranges for the same base software such as professional and basic versions. *URL* gives a location for further information and download, *Citation* provides a formal, bibliographic citation, and *License* covers the permission and terms of usage. Lastly, *Abbreviation* gives a shortened name for a software while *Alternative Name* provides a longer name. All additional information is related to the specific entity it describes. In most cases this is a software, but licenses can also be specified by versions, URLs and abbreviations, while developers can be closer described by URLs and abbreviations.

Data Model and RDF/S lifting

In order to ensure interpretability and reusability, extracted data is lifted into a structured KG based on established vocabularies. KGs represent a meaningful way to semantically structure information in an unambiguous way and provide a reasonable approach to make data accessible for later reuse. In particular, KGs enable the FAIR publication of research data.

The data model of the KG is depicted in Fig. 5. It can be subdivided into different areas that represent different types of information. Bibliographic information about articles, journal and authors (depicted in violet color) is represented by employing terms from the Bibliographic Ontology (BIBO) (*D'Arcus & Giasson, 2009*), Dublin Core Metadata Initiative Terms (DCT) (*DCMI Usage Board, 2020*), Simple Knowledge Organization System (SKOS) (*Miles et al., 2005*), and schema.org (*Guha, Brickley & Macbeth, 2016*).

The representation of entity mentions that were automatically extracted from the texts (orange color), is mainly built upon the NLP Interchange Format (NIF) (*Hellmann et al., 2013*) and Datacite (*Peroni et al., 2016*). Disambiguated software are represented by *Software* from Informatics Research Artifacts Ontology (IRAO) (*Bach, 2021*). For the metadata of software we examined dedicated vocabularies and ontologies including DOAP (Description of a Project) (*Wilder-James, 2018*), SDO (Software Description Ontology) (*Garijo et al., 2019*), SWO (Software Ontology) (*Malone et al., 2014*), OS (OntoSoft) (*Gil, Ratnakar & Garijo, 2015*), and Codemeta (*Jones et al., 2017*) (including their crosswalk), but did not use those terms as they do not represent the textual information but the real entities. For clear separation of fact and prediction we opted to not create entities from our mentions but model the mentions as they are and provide information inferred on top of them in the form of reification statements (green color). Whenever we were not able to identify existing vocabularies that allow the representation, we introduced new terms under the prefix skg (http://data.gesis.org/softwarekg/vocab/). This was necessary for modelling the information, mention and software types.

Articles and mentions are central entities of the KG. Mentions of all pieces of information extracted from an article (*schema:ScholarlyArticle*) such as software, version or developer are represented by *nif:String*. Software mentions are assigned a software type (*skg:softwareType*) and a mention type (*skg:mentionType*, yellow). For all other mentions the type is noted using the *skg:informationType* property (yellow). To represent relations at the textual level, we introduced predicates for each possible relation. The mention of a software, for instance, refers to the corresponding version *via skg: referredToByVersion*.

In order to indicate different degrees of probabilities for information aggregated over disambiguated software entities we use reification statements (*rdf:Statement*) instead of domain entities. Confidence values based on the frequency within and across articles are used to provide a measure of certainty. Formally, let $I_{r,x}$ be the set of all forms of a piece of information for a given relation r and software x. Further, let D be the set of all articles and $m_{r,a,x}$ the mapping of a piece of information $a \in I_{r,x}$ to x under the relation r, we then define the confidence score $c_{m_{a,x}}$ as given in:

$$c_{m_{r,a,x}} = \frac{1}{|\{d \in D | m_{r,b,x} \in d, b \in I_{r,x}\}|} \cdot \sum_{d \in D} \frac{|\{m_{r,a,x} \in d\}|}{\sum_{b \in I_{r,x}} |\{m_{r,b,x} \in d\}|}, \; a \equiv b.$$

$a \equiv b$ signals that both, a and b represent the same type of information, *e.g.*, name or developer. This way we achieve a ratio based fair weighting on mention level and on document level. All values range from 0 to 1 and also add up to 1.

Additional information sources

SoftwareKG was build upon data from PMC making use of the PMC OA JATS XML data set as structured information source for article metadata. Data from PubMedKG (*Xu et al., 2020*) was integrated to allow bibliometric and domain specific analyses. In particular, we used PKG2020S4 (1781-Dec. 2020), Version 4 available from http://er.tacc.utexas.edu/datasets/ped. It includes Scimago data on journal H-index, journal rank, best quartiles as

Table 5 Development set results on software mention recognition. Models marked with *opt* were optimized with respect to hyper-parameters, models marked with *plain* were not. Bold results highlight best performance for both plain and optimized models.

	Precision	Recall	FScore
	Model compare (n = 499)		
SoMeSci Baseline	0.82	0.77	0.79
$M_{L,sw,opt}$	0.829 (±0.016)	0.762 (±0.011)	0.794 (±0.004)
$M_{BB,sw,plain}$	0.862 (±0.005)	0.808 (±0.011)	0.834 (±0.006)
$M_{SB,sw,plain}$	**0.863** (±0.016)	**0.844** (±0.009)	**0.853** (±0.003)
$M_{SB,sw,opt}$	**0.868** (±0.006)	**0.865** (±0.012)	**0.866** (±0.008)

well as their domains and publishers. Moreover, it includes citation information for articles in PubMed from PubMed itself and Web of Science. For integration of PubMedKG we matched PMC identifiers to PubMed identifiers based on PMC's mapping service, available at https://www.ncbi.nlm.nih.gov/pmc/pmctopmid/. Specifically, we used their CSV table to match the PMC-ID in PubMed Central with PM-ID from PubMedKG.

Journal specific information vary over time so we modelled them in an *skg:JournalInfo*-entity that encapsulates information per year. Citation data are integrated in two ways: (1) all citations between PMC Open Access articles are inserted as *schema:citation* in the KG and (2) the overall number of citations an article received is included as a citation count. This allows analyses based on citation counts, but also provides a basis to identify particular citations paths.

RESULTS: INFORMATION EXTRACTION PERFORMANCE

Entity recognition and classification

Performance for software recognition on development set, used to select the best model, is summarized in Table 5. All values are provided by mean and standard deviation for repeated training to assess the effect of randomization in the training process of deep learning models. We found that both BERT based models perform better than $M_{L,sw,opt}$ in both Precision and Recall. As described above, $M_{SB,sw,plain}$ and $M_{BB,sw,plain}$ were initially compared with the same set of default hyper-parameters and only the best of the two models was optimized. In the initial comparison, $M_{SB,sw,plain}$ showed better performance than $M_{BB,sw,plain}$ with respect to Recall and was therefore selected. We found that hyper-parameter optimization for $M_{SB,sw,plain}$ improved performance further, especially in terms of Recall. A detailed overview of all performed hyper-parameter tests for the Bi-LSTM-CRF ($M_{L,sw,\_}$) is given in supplementary Tables A1–A6 and for SciBERT ($M_{SB,sw,\_}$) in supplementary Tables A7–A10. The chosen hyper-parameter configuration for $M_{SB,sw,opt}$ is summarized in Table 6. It outperforms baseline by 7 pp on the development set and is selected as the best model for the task.

The test set performance of $M_{SB,sw+info,opt}$ on all classification tasks is summarized and compared to the baseline in Table 7. Software extraction and overall entity recognition perform well with respective FScores of 0.883 (±0.005) and 0.885 (±0.005). The entity types Extension, Release, and AlternativeName, for which the fewest data samples are

Table 6 Selected hyper-parameters for $M_{SB,sw,opt}$ fine-tuning.

Parameter	Value
Learning Rate (LR)	1e−6
Sampling	all data
Dropout	0.2
Gradient clipping	1.0

Table 7 Software mention extraction results for $M_{SB,sw+info,opt}$ in comparison with SoMeSci baseline as reported by *Schindler et al. (2021b)*, where *n* denotes the number of samples available for each classification target. Please note that the baseline model applies hierarchical classifiers on the task and does not adjust the performance for error propagation between the initial classification of software and all other down-stream tasks. Therefore, all baseline results except for *software* are prone to overestimate performance when compared to the given results. Bold results highlight best performance in terms of FScore.

	$M_{SB,sw+info,opt}$			SoMeSci baseline	
	Precision	**Recall**	**FScore**	**FScore**	***n***
Software	0.876 (±0.011)	0.891 (±0.009)	**0.883** (±0.005)	0.83	590
Abbreviation	0.884 (±0.046)	0.879 (±0.025)	**0.881** (±0.029)	0.71	17
AlternativeName	0.719 (±0.09)	0.734 (±0.061)	**0.726** (±0.075)	0.25	4
Citation	0.868 (±0.018)	0.855 (±0.027)	0.861 (±0.015)	**0.87**	120
Developer	0.867 (±0.025)	0.901 (±0.029)	**0.883** (±0.023)	0.88	110
Extension	0.331 (±0.045)	0.688 (±0.099)	0.444 (±0.053)	**0.60**	5
License	0.799 (±0.056)	0.83 (±0.061)	**0.814** (±0.057)	0.80	14
Release	0.499 (±0.049)	0.771 (±0.027)	0.605 (±0.042)	**0.82**	9
URL	0.858 (±0.028)	0.979 (±0.006)	0.914 (±0.016)	**0.95**	53
Version	0.927 (±0.014)	0.94 (±0.006)	**0.934** (±0.008)	0.92	190
Entities	0.875 (±0.009)	0.897 (±0.009)	**0.885** (±0.005)	0.85	1,112
Application	0.788 (±0.012)	0.865 (±0.014)	**0.824** (±0.007)	0.81	415
OS	0.933 (±0.036)	0.852 (±0.023)	**0.89** (±0.023)	0.82	30
PlugIn	0.652 (±0.05)	0.408 (±0.029)	**0.5** (±0.023)	0.43	78
PE	0.924 (±0.014)	0.998 (±0.005)	0.96 (±0.009)	**0.99**	63
Software Type	0.792 (±0.010)	0.818 (±0.01)	**0.800** (±0.008)	0.78	590
Creation	0.784 (±0.043)	0.805 (±0.024)	**0.794** (±0.029)	0.64	53
Deposition	0.71 (±0.058)	0.821 (±0.018)	**0.761** (±0.036)	0.65	28
Allusion	0.603 (±0.058)	0.464 (±0.046)	**0.522** (±0.038)	0.29	71
Usage	0.832 (±0.013)	0.883 (±0.011)	**0.857** (±0.007)	0.80	438
Mention Type	0.794 (±0.016)	0.823 (±0.01)	**0.806** (±0.01)	0.74	590

available, show a lower performance compared to the other entities. Software types are recognized with a good overall performance of 0.800 (±0.008). Especially the types Programming Environment and Operating System are recognized with high performance. The software type Application is also recognized well, but PlugIn shows a lower performance of 0.5 (±0.023). Mention type classification also performs well with 0.806

Table 8 Summary of RE results for both development and test set. SoMeSci represents baseline FScores for comparison. P, Precision; R, Recall; F1, FScore; n, Number of samples per relation. Bold results highlight best performance in terms of FScore.

	Development set					Test set				
	Random forest		SoMeSci			Random forest		SoMeSci		
Label	P	R	F1	F1	*n*	P	R	F1	F1	*n*
Abbreviation	1.00	1.00	1.00	1.00	17	1.00	0.94	0.97	0.97	17
Developer	0.94	0.97	0.95	0.95	87	0.95	0.95	**0.95**	0.94	111
AltName	1.00	1.00	**1.00**	0.83	6	1.00	1.00	1.00	1.00	4
License	0.88	0.70	**0.78**	0.57	10	1.00	0.93	**0.96**	0.64	14
Citation	0.94	0.97	**0.95**	0.83	90	0.94	0.92	**0.93**	0.86	121
Release	0.78	1.00	**0.88**	0.80	7	0.88	0.78	**0.82**	0.53	9
URL	0.93	0.94	**0.94**	0.80	70	0.98	0.92	**0.95**	0.89	53
Version	0.97	0.99	**0.98**	0.96	139	0.98	0.96	**0.97**	0.95	190
Extension	1.00	1.00	1.00	1.00	5	1.00	1.00	**1.00**	0.89	5
PlugIn	0.77	0.66	**0.71**	0.60	35	0.85	0.72	**0.78**	0.65	39
Specification	0.67	0.67	**0.67**	0.60	6	0.83	0.62	**0.71**	0.22	8
Overall	0.93	0.94	**0.93**	0.87	472	0.95	0.92	**0.94**	0.88	571

(±0.01). Here, mention type Allusion is the most challenging target with 0.522 (±0.038) FScore, while all other targets are extracted with a satisfactory performance.

Relation extraction

The results for RE by the Random Forest as well as the original SoMeSci baseline on both, development set and test set are given and compared in Table 8. In summary, high recognition rates with 0.94 FScore are observed, showing improvements resulting from our extension and optimization to the baseline. At the level of the individual relation types, high FScores (>0.9) are observed for all types except for Release $F1 = 0.82$, PlugIn $F1 = 0.78$, and Specification $F1 = 0.71$. This indicates that relations between two software entities, *i.e.*, PlugIn and Specification, are particularly challenging classification targets.

Software disambiguation

As described above, the disambiguation first uses a perceptron model to estimate linking probabilities between entity pairs and afterwards uses the probabilities for agglomerative clustering. The optimized perceptron predicted links between software entities with a Precision 0.96, Recall 0.90, and FScore 0.93. These values were estimated on the test dataset with a threshold of $t = 0.5$ based on the sigmoid output. The perceptron performance does influence the final performance, but during clustering entity pairs $p_{link}(E_A,E_C) < t$ can still be linked even if they have not been predicted by the perceptron through a chain of closer entities: $p_{link}(E_A,E_B) > t$, $p_{link}(E_B,E_C) > t$. Therefore, evaluation of the perceptron alone does not allow to make statements about disambiguation quality.

For the actual agglomerative clustering based on single linkage performance was estimated with a Precision of 0.99, Recall of 0.96, and FScore of 0.97 at a optimal threshold of $t = 0.00347$ for clustering all gold label data in a common features space with all

extracted data. The SoMeSci baseline results are reported with Precision of 0.99, Recall of 0.96 and FScore of 0.97, but as noted above they cannot be compared to the results reported here. The small threshold is a clear indicator of how densely populated the feature space is considering all extracted software mentions. In total 605.364 clusters of software were generated.

RESULTS: ANALYSIS OF SOFTWARE MENTIONS

KG statistics

The KG was constructed from 301,825,757 subject-predicate-object triples, representing 11.8 M software mentions in more than 3.2 M articles in 15,338 journals from 2,136 publishers. On average, each journal contains 210 articles, ranging from 1 article to 239,962 articles in the journal PLoS One.

For ~8.7 K journals (containing 2.8 M articles, 86.7%) additional information, including citations, research domain and journal ranks was identified from integrating data of PubMedKG (*Xu et al., 2020*). For almost 2.5 M articles a citation count different from 0 could be found. In summary, 303 categories from 27 top level domains were found, see Table 9.

A detailed overview of article and journal frequencies per research domain is provided in supplementary Table A11. As expected from a repository of Open Access articles from Biomedicine and Life Sciences, the distribution of journals and articles is skewed towards Medicine, as ~1.9 M articles from 4,455 journals are related to Medicine, while only 2,178 articles from 181 journals are related to Economics. However, there is a high relative amount of articles not directly related to medicine (more than 30%). This includes disciplines such as Computer Science (~77 K articles from 396 journals) and Mathematics (~39 K articles 364 journals), but also Business (~3 K articles from 173 journals) and Arts and Humanities (~9 K articles from 469 journals).

Software mentions

Different spellings of the same software were grouped during disambiguation, resulting in 605,362 unique software instances with 1.08 different spellings and 19.48 mentions on average. A highly skewed distribution of mentions per software can be observed, where about 10% of the software account for about 90% of the software mentions across all articles. Figure 6 illustrates this distribution graphically. Table 10 provides an overview of the 10 most frequent software, including their absolute and relative number of mentions across all articles. Furthermore, the number of articles containing at least one mention of the respective software is given in the column # Articles. With 539,250 respectively 469,751 mentions, SPSS and R are mentioned most frequently across all articles, where 440 different spellings were observed for SPSS and only 1 for R. The different spellings for SPSS include common names such as “SPSS” (78.4%), “SPSS Statistics” (10.8%), and “Statistical Package for the Social Sciences” (3.8%), but also those with spelling mistakes such as “Statistical Package for the Spcial [*sic*] Sciences”.

Figure 7 illustrates the top 10 software per research domain. Domain-specific differences can be observed from the plot. No domain is consistent with the

Table 9 Overview of the 27 main research domains and 3 of their sub categories that were used to group journals. Bold font highlights the abbreviation of the respective research domain used here.

Main research domain	Research subcategories (excerpt)
Physics and Astronomy	Acoustics and Ultrasonics, Astronomy and Astrophysics, Atomic and Molecular Physics, and Optics
Chemistry	Analytical Chemistry, Chemistry (miscellaneous), Electrochemistry
Social Sciences	Anthropology, Archeology, Communication
Materials Science	Biomaterials, Ceramics and Composites, Electronic
Engineering	Aerospace Engineering, Architecture, Automotive Engineering
Economics, Econometrics and Finance	Economics and Econometrics, Economics, Econometrics and Finance (miscellaneous)
Multidisciplinary	Multidisciplinary
Energy	Energy (miscellaneous), Energy Engineering and Power Technology, Fuel Technology
Agricultural and Biological Sciences	Agricultural and Biological Sciences (miscellaneous), Agronomy and Crop Science, Animal Science and Zoology
Environmental Science	Ecological Modeling, Ecology, Environmental Chemistry
Veterinary	Equine, Food Animals, Small Animals
Nursing	Advanced and Specialized Nursing, Assessment and Diagnosis, Care Planning
Decision Sciences	Statistics, Probability and Uncertainty, Information Systems and Management
Earth and Planetary Sciences	Atmospheric Science, Computers in Earth Sciences, Earth and Planetary Sciences (miscellaneous)
Pharmacology, Toxicology and Pharmaceutics	Drug Discovery, Pharmaceutical Science, Pharmacology
Mathematics	Algebra and Number Theory, Analysis, Applied Mathematics
Computer Science	Artificial Intelligence, Computational Theory and Mathematics, Computer Graphics and Computer-Aided Design
Biochemistry, Genetics and Molecular Biology	Aging, Biochemistry, Biochemistry
Dentistry	Dentistry (miscellaneous), Oral Surgery, Orthodontics
Neuroscience	Behavioral Neuroscience, Biological Psychiatry, Cellular and Molecular Neuroscience
Arts and Humanities	Archeology (arts and humanities), Arts and Humanities (miscellaneous), Conservation
Psychology	Applied Psychology, Clinical Psychology, Developmental and Educational Psychology
Business, Management and Accounting	Accounting, Business and International Management, Business
Medicine	Anatomy, Anesthesiology and Pain Medicine, Biochemistry (medical)
Immunology and Microbiology	Applied Microbiology and Biotechnology, Immunology, Immunology and Microbiology (miscellaneous)
Health Professions	Chiropractics, Complementary and Manual Therapy, Health Information Management
Chemical Engineering	Bioengineering, Catalysis, Chemical Engineering (miscellaneous)

domain-independent view (see Table 10), and each domain is characterized by a different distribution of the top 10 software. While SPSS (top 1 for 13/27) and R (top 1 for 6/27) together represent the top mentioned software in more the 70% of the domains, Excel, BLAST, Prism, and ArcGIS (each 1/27) are the top software in Economics, Energy, Immunology, and Business, respectively. The software SHELXL, SHELXS, SAINT, and SHELXTL play a mayor role in Chemistry, Materials, and Physics, taking ranks among 1–5 in each of these domains, but are not among the top 10 in any other field. Several programming environments are listed among the top 10 software, including R, Python, Java, C, and C++, most prominent in Mathematics, Engineering, and Computer Science. Material Science plays a special role, when it comes to domain specific software because

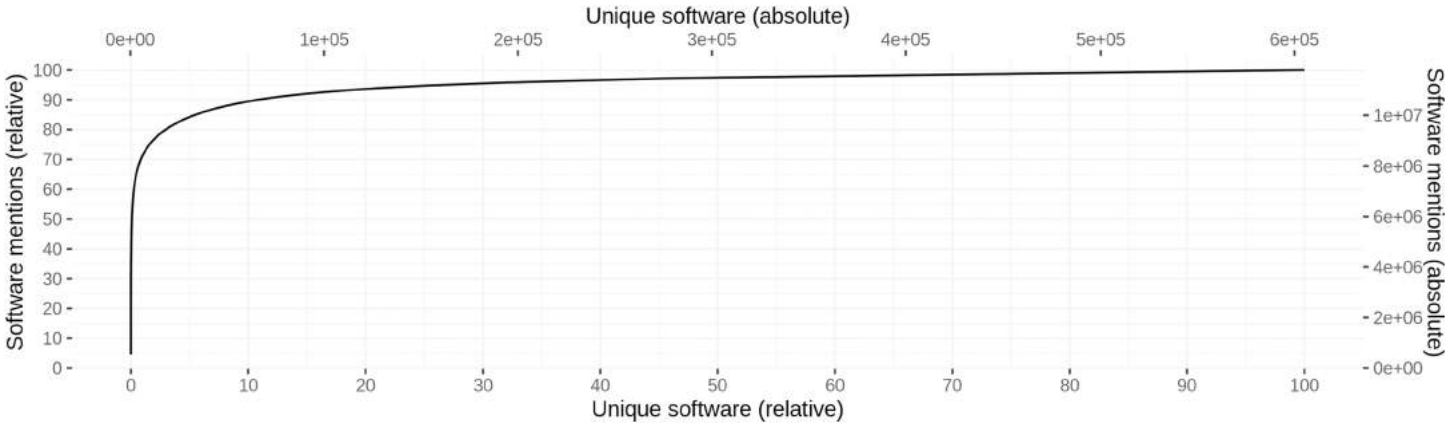


Figure 6 Cumulative distribution of software mentions per unique software. Left (bottom) scale gives the relative values, whereas right (top) scale provides the absolute numbers. Full-size DOI: 10.7717/peerj-cs.835/fig-6

Table 10 Information about the 10 most frequent software mentions across all disciplines together with their absolute and relative number of mentions, the number of articles that contain at least one mention and the number of spelling variation that could be disambiguated.

Software	Absolute #	Relative #	# Articles	# Spellings
SPSS	539,250	4.57	466,505	440
R	469,751	3.98	235,180	1
Prism	220,175	1.87	189,578	1
ImageJ	228,140	1.93	144,737	83
Windows	140,941	1.19	127,691	6
Stata	147,586	1.25	118,413	141
Excel	151,613	1.29	118,082	54
SAS	140,214	1.19	112,679	215
BLAST	271,343	2.30	104,734	383
MATLAB	160,164	1.36	89,346	6

the Operating System Windows is the only software that is shared with the top 10 overall software, while the remaining software allows a unique characterization. Regarding operating systems, Windows is frequently mentioned in all research domains, except for Earth and Planetary Sciences and Energy. The Linux operating system, in contrast, is ranked 5th in Mathematics, 7th in Decision Sciences, and 9th in Computer Science, respectively. The source code and data repositories GitHub and FigShare are listed among the top 10 software in Decision Science with rank 5 and 9.

Article level statistics

On the article level, we observe that each article contains 3.67 software mentions on average, ranging from 0 software mentions for 1,301,192 articles to a maximum of 673 mentions for one article. Looking at the number of articles per year, it can be observed that the relative number of articles mentioning at least one software increases over all articles. Figure 8 (blue line) illustrates this graphically. Considering those articles only, a similar

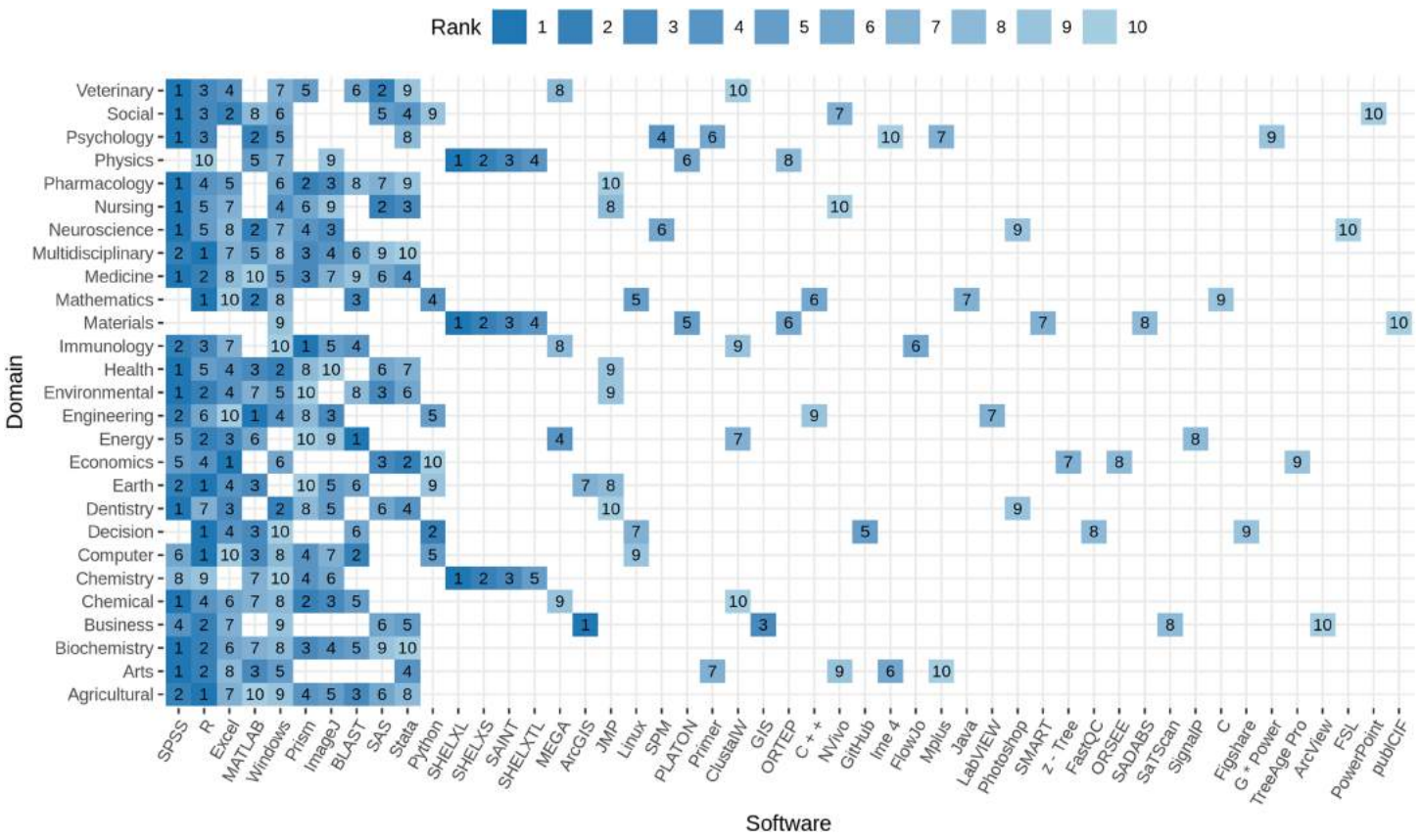


Figure 7 Top 10 software per domain. Higher rank within the domain is represented by darker color. The number on the tile gives the rank within the domain. Software with rank higher than 10, are excluded from the plot to improve readability. Software are ordered by rank over all domains left to right.
Full-size DOI: 10.7717/peerj-cs.835/fig-7

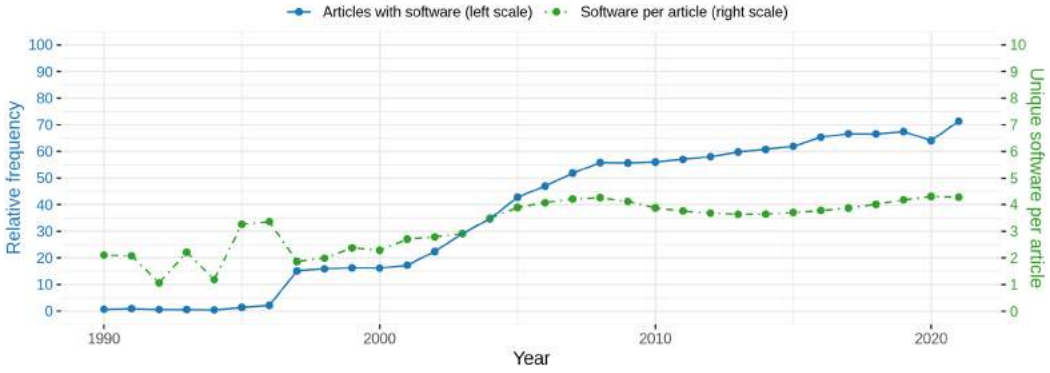


Figure 8 Blue: Relative frequency of articles with at least one software mention per year. Green: Absolute mean frequency of unique software mentioned per article with at least one software mention per year. Please note that standard deviations are at the same level as the actual average values but are omitted here for reasons of readability.
Full-size DOI: 10.7717/peerj-cs.835/fig-8

trend can be observed from the mean frequency of software mentioned within one article (Fig. 8, green line). Due to the low number of software mentions before 1997 (blue line), the estimation of mean software frequencies per article is less reliable before 1997.

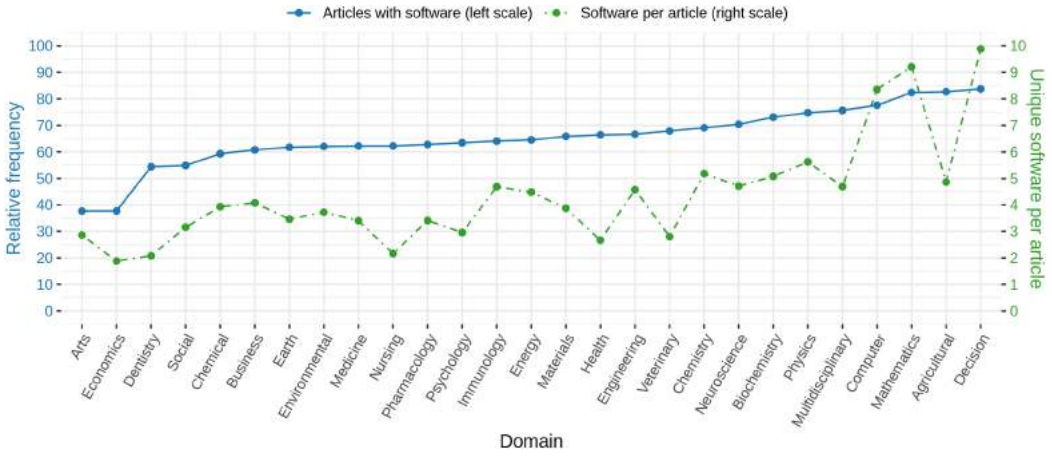


Figure 9 **Blue: Relative frequency of articles with at least one software mention per research domain. Green: Average number of different software mentioned per article with at least one software mention given by research domain.** Note that standard deviations are large (similar to average values) and are omitted here. Full-size DOI: 10.7717/peerj-cs.835/fig-9

In 1997, a steep increase in the number of articles with software can be observed which remains constant until 2000. From 2001 another increase until 2008 can be observed, which is followed by a phase where the relative frequency of articles with software increases more slowly until 2021. From 2007 more than 50% of the articles contain at least one software, increasing to almost 75% in 2021. A notable decrease was observed for 2020. With respect to the number of software per article (green line), the frequency remains at a constant level of ~4 from 2005. Standard deviations are omitted but are on a high level between 2 for low mean values and 4 for higher means. To determine the effect of the year on the number of software per article, a linear model was fitted to explain the binary logarithm of the number of software per article by the interaction of year and domain. We found a significant ($p < 0.001$) but small influence of the year (slope = 0.017, $S_e = 0.0006$, $R^2 = 0.06525$), when considering the interaction with the domain.

When looking at the relative amount of articles with software per research domain, we found notable differences between the individual domains. Figure 9 (blue line) illustrates those differences graphically. While in Arts and Economics only 40% of the articles contain software mentions, in Mathematics, Agriculture and Biological Sciences, and Decision Sciences, more than 80% of the articles mention at least one software. The number of different software per article draws a similar picture (Fig. 9, green line), ranging from values of 2 or 3 in Arts, Economics, and Dentistry to values above 8 for Computer Science, Mathematics, and Decision Science. A one-way ANOVA revealed these differences to be significant ($p < 0.001$, $F_{26,3468692} = 7950.1$).

Comparing the amount of articles with software mentions with journal rank and citation count per year, similar observations were found. Both graphs are illustrated in Fig. 10. The graph illustrates the ventiles (20-quantiles) of the journal rank, grouped by domain to prevent domain specific biases due to higher journal ranks. In detail, in the first step, for each domain, the journals were distributed according to their rank ventiles and the resulting ventiles were then merged across all domains. A similar approach was

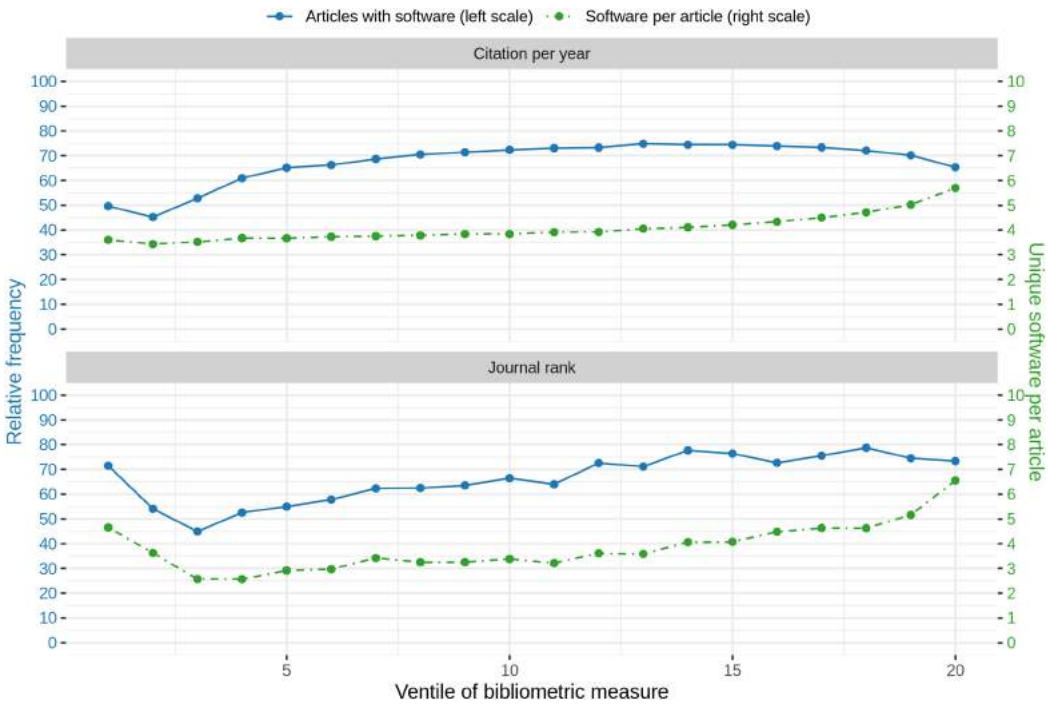


Figure 10 **Blue: Relative frequency of articles that contain at least one software mention per rank of bibliometric measure. Green: Average number of different software per article per bibliometric measure.** Note that the high standard deviation (at the same level as average values) are left out to increase readability. Full-size DOI: 10.7717/peerj-cs.835/fig-10

chosen for summarizing the articles *via* citation count ventiles. Please note that while for journal rank based analysis all ranked journals could be considered, for citation count analysis we restricted the analysis to all articles published before 2020 to reduce a bias towards 0 citations. When considering the journal rank, we found that almost 75% of the articles on the lowest rank contain software mentions, followed by a strong decrease for the next two ventiles (blue line). For the remaining ventiles an increasing trend up to almost 80% for higher journal ranks could be observed. When considering the amount of software per article, an initial high-point and decrease for the four lower journal ranks could be observed followed by an increasing trend with increasing journal rank (Fig. 10, green line). A linear model to explain the relation between binary logarithm of the number of software per article and the journal rank grouped by domain revealed a small but significant ($p < 0.001$) effect (slope = 0.18, $S_e = 0.0017$, $R^2 = 0.087$).

Similarly, a high value followed by a slight decrease could be observed for the relative number of articles mentioning at least one software per citation count per year (blue line), even though the pattern is not that distinctive. After reaching the minimum frequency of articles containing software mentions in the 2nd ventile, the graph shows an increasing trend for the remaining ventiles, with a slow decrease for the last seven ventiles. The maximum frequency of articles with software mentions is reached at 13th citation count ventile. A linear model to explain the relation between binary logarithm of the

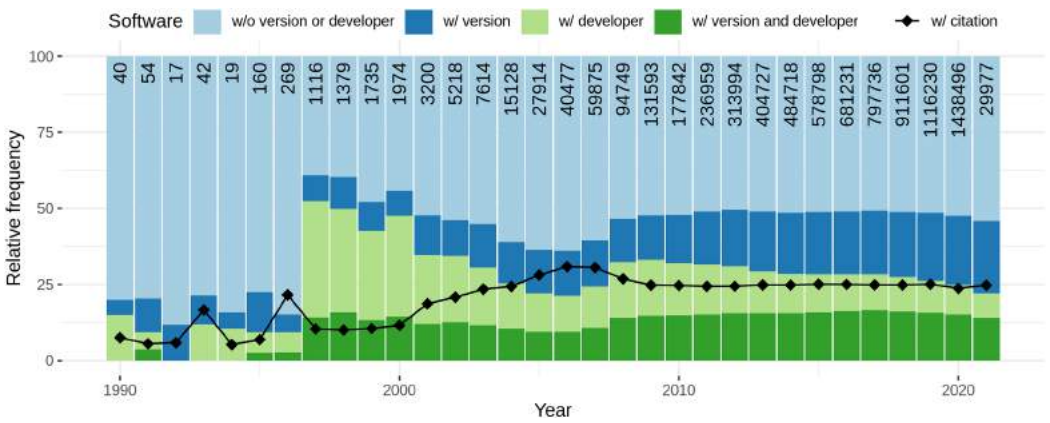


Figure 11 Distribution of software completeness per year with the percentage of unique software per article that is cited with provided additional information. The colored bars represent the different levels of completeness while the line chart separately indicates how many software mentions were accompanied by a formal citation. The numbers at the top of the bars represent the absolute number of software considered per year. Full-size DOI: 10.7717/peerj-cs.835/fig-11

number of software per article and the citation count per year grouped by domain revealed a small but significant ($p < 0.001$) effect (slope = 0.01, $S_e = 0.0003$, $R^2 = 0.065$).

Software citation completeness

Considering articles containing software mentions, we analysed the amount of information necessary to identify particular software provided for each software mentioned within one article. For each unique software (which might be referred to multiple times within the same article) we examined whether the version and/or the developer was mentioned within the article. Moreover, the frequency of formal citation (a citation referring to the bibliography of the article) was investigated.

Figure 11 depicts the completeness of software mentions per year. From the numbers at the top of the bars, it can be seen that the number of unique software per article increased over the years, ending with 1.44 M software mentions (unique per article) in 2020. The low number in 2021 reflects the time of data retrieval. Regarding citation completeness, from 1990 to 1996 the amount of both, information accompanying mentions as well as formal citations, is low for an overall low number of software mentions. From 1997 to 2000 there is a peak in additional information provided for software mentions with a low number of corresponding formal citations. Afterwards, up to year 2007, there is a decay in additional information for informal mentions and a contrary increase in formal citations. From 2008 to 2010 there is another increase of the amount of provided information and a decrease in formal citation. The numbers then stagnate up to 2021. Overall, it can be seen that the frequency of software with developer decreases and with version increases. Both, the relative amount of formal citations and the amount of software accompanied with version and developer remained constant since 2009.

Considering domain specific software citation habits, Fig. 12 illustrates the amount of information provided per software over different research domains. Mathematics, Decision Science, and Computer Science provide the least additional information with

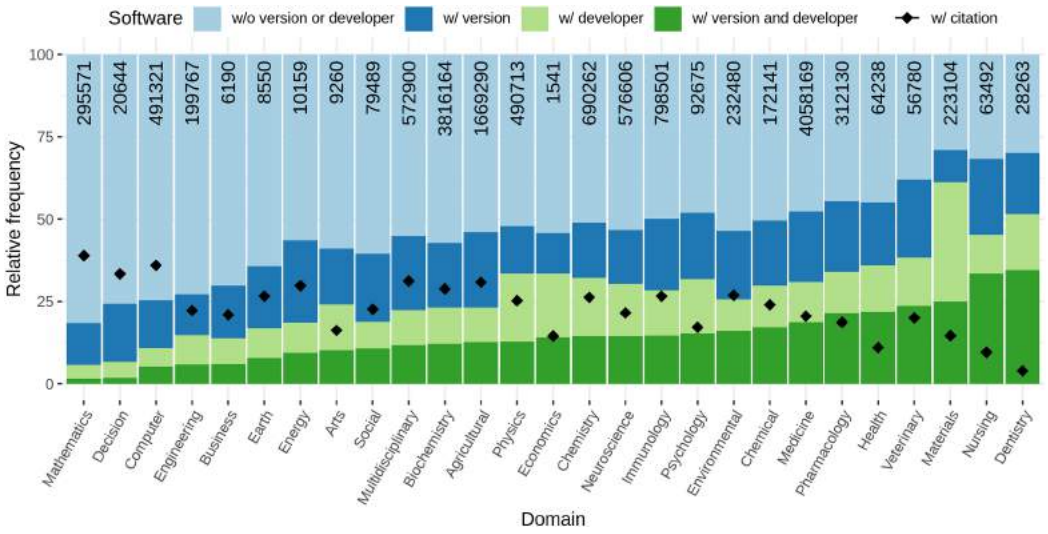


Figure 12 Distribution of software completeness per research domain. The numbers at the top of the bars represent the absolute numbers of software considered per domain. Please note that articles may belong to multiple categories. Full-size DOI: 10.7717/peerj-cs.835/fig-12

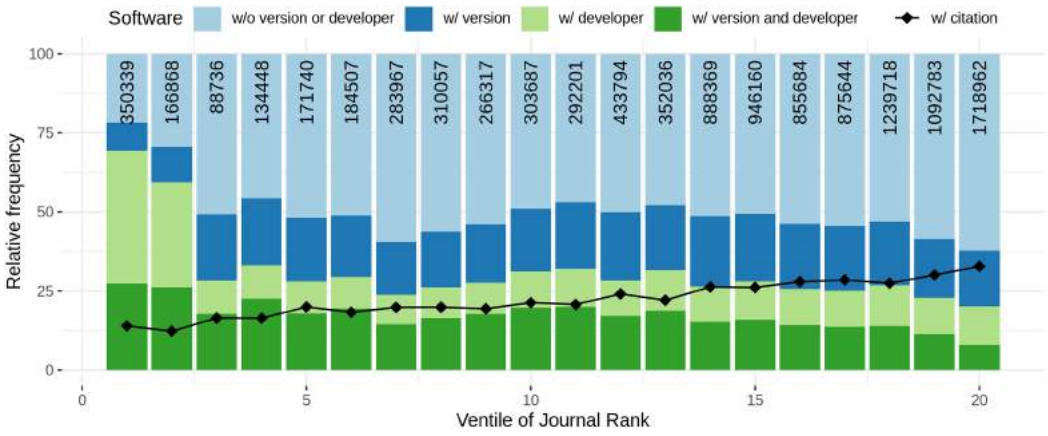


Figure 13 Distribution of software completeness per ventile of journal rank per research domains. The numbers at the top of the bars represent the absolute numbers considered per ventiles. Full-size DOI: 10.7717/peerj-cs.835/fig-13

down to 20% of mentions, but all three have comparably high numbers of formal citation with around 40%. Dentistry, Nursing, and Materials, in contrast, provide most additional information with up to 70% of mentions but low numbers of formal citations with down to 5%. Nursing and Dentistry also have the highest share of software together with version and developer. In general, citation completeness is not better in domains that use most software.

With respect to journal rank, a slight negative correlation between the rank ventile and the amount of additional information can be observed. Figure 13 illustrates this relation graphically. In contrast, a positive correlation between amount of formal citation and journal rank can be seen. While most software are accompanied with developer for low ranked journals (almost 70%), the percentage decreases with rising journal rank,

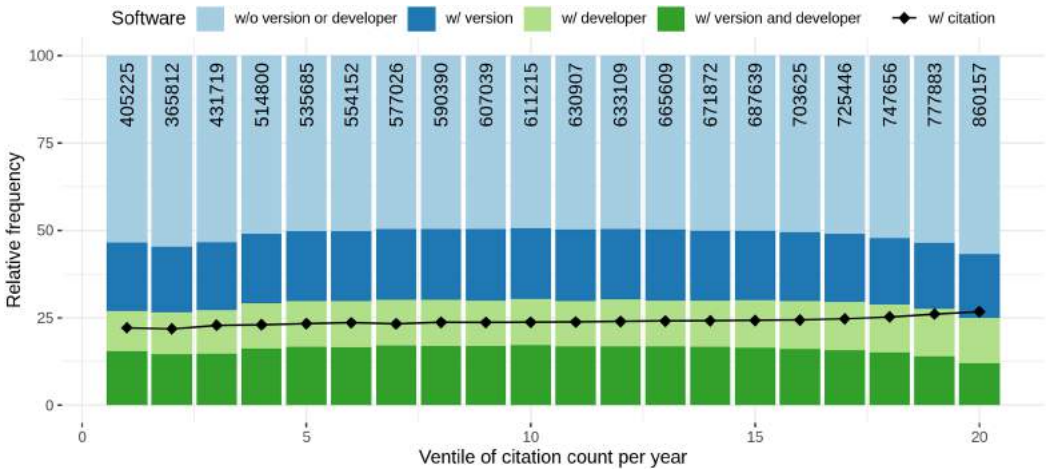


Figure 14 Distribution of software completeness per ventile of citation count per research domain. Note that only articles published before 2020 were included to prevent a bias towards lower citation ventiles. Full-size DOI: 10.7717/peerj-cs.835/fig-14

reaching a first local minimum at rank 8 and the absolute minimum at the highest rank. The use of formal citations, in contrast, shows an increase from ~10% at the lowest ranked journals to about 30% at the highest ranked journals.

Considering the citation rank (see Fig. 14), a slight increase in provided additional information could be observed in the first four ventiles and a slight decrease in the last four. No notable difference could be observed for the remaining citation ventiles. For formal citation, a slight increase over the ventiles can be observed, starting with 20% and reaching up to over 25%.

Types of software mention

For each software mention, the KG contains information about the type of mention and the type of software. The most frequent type of software is Application with 84.49%, followed by Programming Environment with 7.29%, PlugIn with 6.27%, and Operating Systems with 1.95% of the mentions. When looking at the disambiguated software, 88.52% of the software are Applications, 10.74% PlugIns, 0.41% Programming Environments, and 0.33% Operating Systems. With respect to the type of mention, we observed that most mentions of software reflect Usage with 82.31%, whereas 15.09% represent Allusion. Only 2.01% (0.6%) of the software mentions represent Creation (Deposition) statements. Table 11 gives a fine-grained overview of the relation between type of mention and type of software. The mention type Usage is prevalent over all software types and the software type Application over all mention types. Furthermore, we found that the relative frequency for both Creation and Deposition is 0 for Operating Systems and Programming Environments. When looking into domain specific differences, it can be observed that Mathematics (51.4%), Engineering (55.0%), and Computer Science (55.0%) have the lowest share of usage statements, while disciplines such as Energy (90.2%), Materials Science (93.1%), Nursing (93.3%), Dentistry (93.9%), and Veterinary (95.7%) have the highest share of usage statements. The opposite trend can be observed for software

Table 11 Overview of the relative frequency of software and mention types as well as their combinations over all software mentions. Note that overall numbers do not necessarily sum to 100 due to rounding issues.

	Allusion	Creation	Deposition	Usage	Overall
Application	13.95	1.80	0.56	68.14	84.49
OperatingSystem	0.26	0.00	0.00	1.69	1.95
PlugIn	0.47	0.17	0.04	5.58	6.27
ProgrammingEnvironment	0.40	0.00	0.00	6.88	7.29
Overall	15.09	2.01	0.60	82.31	100.00

Table 12 Most frequent host software, i.e., mentioned together with a PlugIn, in combination with the most frequently used PlugIns for each of them. # PlugIn, distinct, disambiguated PlugIns; # Mention, overall PlugIn mentions.

Software	# PlugIn	# Mention	Top 5 PlugIn incl. % of mentions
R	19,442	220,750	Bioconductor (4.79%), ggplot 2 (3.63%), lme 4 (3.21%), vegan (3.09%), DESeq 2 (2.52%)
MATLAB	4,442	18,616	Psychophysics Toolbox (6.75%), Psychtoolbox (6.03%), Statistics Toolbox (3.92%), Image Processing Toolbox (2.87%), Neural Network Toolbox (1.43%)
Python	3,157	11,688	scikit - learn (12.55%), SciPy (4.10%), TensorFlow (3.75%), Network (2.37%), scipy (2.30%)
python	1,449	3,533	scikit - learn (10.44%), scipy (3.85%), sklearn (2.83%), matplotlib (2.52%), HTSeq (2.12%)
ImageJ	1,286	10,761	Fiji (44.10%), NeuronJ (3.00%), Cell Counter (2.77%), MTrackJ (2.11%), Analyze Particles (1.64%)
Stata	809	2,190	metan (5.94%), runmlwin (3.06%), mvmeta (2.33%), Image Composite Editor (1.87%), metareg (1.78%)
Perl	774	1,176	MISA (3.74%), speaks - NONMEM (2.64%), Bioconductor (2.21%), NONMEM (1.36%), Shell (1.11%)
Excel	644	1,946	XLSTAT (17.83%), nSolver (3.96%), Microsatellite Toolkit (2.77%), Analysis ToolPak (2.00%), @ Risk (1.95%)
Cytoscape	553	5,902	ClueGO (13.62%), MCODE (13.00%), BiNGO (7.66%), NetworkAnalyzer (7.56%), Enrichment Map (5.71%)
SPM	521	2,671	DARTEL (20.10%), MarsBaR (9.55%), Marsbar (2.92%), CONN (2.62%), DPARSF (2.55%)

Allusion statements, starting with Veterinary (3.9%) and ending with Mathematics (40.7%). Software Creation statements have the highest proportion in Mathematics with 6% of all mentions, followed by Engineering (5.4%) and Computer Science (5.4%). In Dentistry and Veterinary only 0.3% and 0.4% are Creation statements. Decision Sciences (1.8%), Mathematics (1.7%) and Computer Science (1.5%) have the highest share of deposition statements, in contrast, less than 0.1% of software mentions are Deposition statements in Materials Science and Veterinary.

The software type PlugIn plays a special role, as it can only be used together with another software, requiring the mention of both, the host software and the PlugIn. Table 12 lists the top 10 host software together with the number of PlugIns identified for this software, the overall amout of PlugIn mentions for the software and its top 5 PlugIns. The Programming Environment R was found to be by far the software with most PlugIns (19,442 distinct PlugIns), followed by Matlab and Python. As most frequently mentioned PlugIns for R we found Bioconductor, ggplot2, lme4, vegan, and DESeq2 which together account for 17.2% of all R PlugIn mentions. Note that the two different spellings *Python* and *python* were not linked together but reflect a similar distribution of PlugIns.

Table 13 Top 10 most frequent URLs accompanying software deposition and usage statements together with their absolute and relative frequencies.

Deposition			Usage		
URL	Absolute	Relative	URL	Absolute	Relative
github.com	8,602	13.93	github.com	18,918	3.90
journals.plos.org	5,926	9.60	ncbi.nlm.nih.gov	16,832	3.47
sourceforge.net	918	1.49	r-project.org	13,176	2.71
cran.r-project.org	673	1.09	pacev2.apexcovantage.com	10,504	2.16
bioconductor.org	651	1.05	ebi.ac.uk	9,850	2.03
ebi.ac.uk	478	0.77	blast.ncbi.nlm.nih.gov	8,797	1.81
ncbi.nlm.nih.gov	454	0.74	cbs.dtu.dk	6,539	1.35
bitbucket.org	423	0.69	fil.ion.ucl.ac.uk	6,439	1.33
code.google.com	353	0.57	cran.r-project.org	6,015	1.24
string-db.org	204	0.33	targetscan.org	5,738	1.18

Software creation and deposition

Software Usage and Deposition statements are often accompanied by URLs to provide a location to access a software and make it findable for the scientific community. Table 13 shows the most common domains of URLs mentioned in combination with software Usage and software Depositions. Usage domains (right) correspond to specific software, for instance, blast.ncbi.nlm.nih.gov or r-project.org for BLAST and R, but also to software repositories such as GitHub and specific software package repositories such as CRAN cran.r-project.org. For Depositions (left) we found that most of the URLs point to source code and software repositories. GitHub (github.com) is the most frequent domain with 14% of overall URLs, but other public repositories such as SourceForge, BitBucket or Google Code are present as well as repositories focused on packages such as CRAN and BioConductor.

Another aspect of recognizing software Creation and Deposition statements in articles is that it allows to identify journals that are most frequently used for the description of new software. By analyzing the relative number of articles per journal that contain either a software Creation or Deposition statement, we were able to find the most active journals when it comes to software description. Considering only journals with at least 10 articles, we found that with 90% of the articles containing software Creation statements, the *Proceedings of the VLDB Endowment* is ranked highest among journals introducing software. It is followed by the journal *Source Code for Biology and Medicine* with 84.4%, *Database: the journal of biological databases and curation* with 74.4%, and the *Journal of Open Research Software* and *Bioinformatics* with 72.7% and 70.8%, respectively. The journal *Source Code for Biology and Medicine* contains the most articles with software Deposition statements (64.7%), followed by the *Journal of Open Research Software* and *Neuroinformatics* with 54.5% and 47.1%, respectively. From 15,388 journals, only 3,622 journals contain at least one article with either Creation or Deposition statements.

DISCUSSION

Reliable method for software mention extraction

Information extraction is based on reliable ground truth data from SoMeSci (IRR $F = 0.93$, $\kappa = 0.82$). In combination with state-of-the-art language models for scientific articles such as SciBERT, we achieve state-of-the-art performance for software mention extraction in scientific articles. Regarding software recognition, the SoMeSci baseline was outperformed by a notable margin raising performance to $F = 0.88$ from $F = 0.83$ by 5 pp. This also represents an increase over the previous automatic approaches by *Pan et al. (2015)* with $F = 0.58$, *Duck et al. (2016)* with $F = 0.67$, *Lopez et al. (2021)* with $F = 0.74$ and *Schindler, Zapilko & Krüger (2020)* with $F = 0.82$, however, as prior work was based on different data bases, the results are not directly comparable. With respect to the chosen NER architecture, SciBERT achieved superior recognition rates when compared with Bi-LSTM based models, illustrating the effectiveness of SciBERT for mining scholarly documents. Interestingly, *Lopez et al. (2021)* report notable lower performances for both architectures, which we believe results from the less reliable input including PDF conversion artifacts and ground truth annotation.

Regarding the identification of additional information ($F = 0.89$, baseline $F = 0.85$) as well as software type ($F = 0.80$, SoMeSci baseline $F = 0.78$) and mention type ($F = 0.81$, SoMeSci baseline $F = 0.74$), we achieve better performance than baseline results, especially considering that the reported results already take error propagation into account, which is not the case for baseline results. We also achieve better performance for Version, Developer, and URL as reported for the Softcite corpus (*Du et al., 2021*; *Lopez et al., 2021*), however, these results cannot be directly compared due to different training and test data. Moreover, the study presented here is the first that classifies software mentions according to both, software and mention type. However, we found that software type PlugIn and mention type Allusion were extracted with lower performance as other types. In both cases the lower performance is mainly due to confusion with another class (Application and Usage) with corresponding higher prior probability but a difficult to distinguish context. This is consistent with the results of the manual annotation performed for SoMeSci (*Schindler et al., 2021b*), where annotation IRR was also found to be lowest for these classes.

RE and disambiguation for software mentions has, to the best of our knowledge, not been evaluated as part of any scientific investigation of software mentions in scholarly publications besides the SoMeSci baseline. We improve RE performance by 6 pp from $F = 0.88$ to $F = 0.94$. RE performs well for additional information related to software, but it is challenging ($F = 0.71$–0.78) to predict relations between software entities such as the *plugin-of* relation. This was expected since, by definition, additional information is always related to another entity while two software entities are not necessarily related to each other. Consequently, relations between software are rare compared to the overall number of software mentions. It should also be noted that both, baseline and our reported results, do not take error propagation from entity recognition to RE into account. Overall, we achieved superior recognition rates compared to previous, automatic, large scale analyses

of software mentions in scholarly publications and conclude, thus, that software mentions and additional information extracted by our pipeline are more reliable.

With respect to disambiguation, it has to be noted that previous large scale analyses did not consider spelling variations but only summarized software mentions with equal (or similar) spellings. As this is the first study to use disambiguation methods for software mentions, comparison to state-of-the-art results is not possible. However, the disambiguation baseline performance for SoMeSci of $F = 0.97$ was matched, while considering a much denser feature space. In small data sets, only few spelling variations (and other features used for disambiguation) of software exist; this number increases with the size of the data set. This means that finding reliable boundaries between different software gets harder with increasing size of the data set as rare spelling variations (and other features) of software with similar names tend to overlap stronger with increasing amounts of data. In our case, the training data set contains 3,756 software mentions from 637 different software while information extraction resulted in almost 12 M software mentions. To recreate this effect for the training data, we included a large set of augmented, fictional software names. With respect to evaluation, the negative effect of increasing sample size on the ability of finding reliable boundaries between different software prevents the transfer of quality statement from training to inference dataset. To counteract this effect, we included the manually disambiguated training data in the inference dataset, determined the clustering threshold, and evaluated the quality based on those samples. Same as for RE, it should be noted that error propagation from the previous information extraction steps influence disambiguation performance. The additional augmented samples simulate the effect of false positives, but we cannot estimate to what extent they are successful at suppressing resulting errors. False negative samples do in practice directly influence linking quality.

Due to computational and space complexity we chose a single linkage-based clustering, which is known for semantic drift away from cluster means, but enables an efficient implementation when distance between all pairs is pre-computed and sorted up to a given threshold. Average linkage would have required to re-compute the average distance of all clusters in each step. An initial evaluation showed only marginal differences between single and average linking based clustering for disambiguation, which seemed sufficient for the task at hand. Overall, disambiguation provides reasonable results; 440 different spellings for SPSS[1], have, for instance, been discovered. For the different spellings python and Python (see Table 12), however, no common cluster could be determined. While this clearly represents an error when considering the string only, our machine learning based distance additionally considers the context and accompanying entities such as developer and URL. We believe that the reason here is the low number of spelling variations that prevent the semantic drift to counteract misleading linking probabilities. In consequence, this would mean that more frequent software (with many different contexts and spelling variations) are more likely to be disambiguated than less frequent software (with fewer contexts and spelling variations), which may have had a reinforcing effect on the power law distribution of mentioned software. For software with more frequent spelling and context variations, in contrast, this might result in more false positives and

[1] All 440 different spellings of SPSS have manually been validated.

thus overestimate the software use. For Excel, 54 spelling variations were found that represent 152 K mentions. From those only about 150 K mentions (from 13 different spellings) can be considered as correctly classified. The remaining mentions contain software such as Firefox (1,162, 0.7%) or F (185, 0.1%). A similar phenomenon could be observed for Stata. While disambiguation performance is satisfactory the algorithm can be improved in future work, for instance, by including information on PlugIns provided with software names after an initial disambiguation of the PlugIn names. However, this would lead to higher run-time requirements because a higher number of mention contexts needs to be considered to cover rare features such as PlugIns. The number is currently limited to $n = 6$.

SoftwareKG: knowledge graph of software mentions

SoftwareKG represents the largest dataset of software mentions and related metadata in scholarly publication. It contains 11.8 M software mentions of over 605 K different software automatically extracted from more than 3 M Open Access articles from PMC. Moreover information from PubMedKG was integrated to allow bibliometric analyses. The KG was created by re-using established vocabularies for data representation, such as schema.org, BIBO, and DCT and is available as JSON-LD under Creative Commons Attribution at Zenodo (*Schindler et al., 2021a*). The published version of the KG only contains information available under open licenses. As this is not the case for most of the bibliometric data, those parts where excluded from publication.

SoftwareKG consists of over 300 M triples describing the properties and relations between more than 55 M resources. A summary of the properties of SoftwareKG is given in Table 14. In SoftwareKG, we employ frequency-based confidence values to provide a transparent way to analyse errors that originate from information extraction or author spelling variations. For names, developers as well as software type and other information we included those confidence values in the reification statements to allow further analyses.

SoftwareKG facilitates the large-scale analysis of software mentions in scholarly publications and allows to give insights into the role of software in science. A tutorial to recreate all tables and figures from the KG is included in the Supplemental Material (https://github.com/f-krueger/SoftwareKG-PMC-Analysis). This article contains first analyses and sketches the potential for more elaborate studies. This includes the creation of an impact measure for scientific software but also to provide a software mapping for science in general such as swMath (*Greuel & Sperber, 2014*) for Mathematics.

The role of software in science/PMC

Software mentions

With an average of 3.67 software mentions per article, our result confirms previous studies, ranging from 2.6 (*Schindler et al., 2021b*) to 3.2 (*Howison & Bullard, 2016*) to 5.5 (*Duck et al., 2016*) in different subsets of PMC. With over 605 K, the number of different software from over 11.8 M mentions is high, given that 3.2 M articles were investigated. This number probably overestimates the actual number of software used in science, due to errors from information extraction and disambiguation. The distribution of software

Table 14 Statistics of SoftwareKG. Left: general KG properties. Right: frequencies of resources per type.

Property	Frequency
Triples	301,825,757
Resources	55,953,270
Distinct Types	12
Distinct Properties	47
Reification Statements	2,042,076
Type	**Frequency**
nif:String	22,066,759
schema:Person	20,373,227
schema:Organization	7,063,708
schema:ScholarlyArticle	3,215,346
rdf:Statement	2,042,076
irao:Software	605,352
skg:SoftwareVersion	380,234
skg:JournalInformation	134,369
bibo:Journal	15,338
dct:LicenseDocument	4,748
skos:Concept	303
skos:ConceptScheme	27

mentions per software shows that only few software are used in a large number of articles and thus play a major role in science. This distribution partly confirms the general trend but shows even higher skewness as previously reported statements about the distribution of software mentions in scholarly articles (*Pan et al., 2015*; *Duck et al., 2016*). This amplified trend could be the result of software name disambiguation which was not applied in previous studies and highlights the importance of considering all spelling variations for software usage analysis. The most frequent software (7 from the top 10) are mainly used for statistical data analysis. A closer look at the domain specific distribution of the top 10 software revealed domain specific differences as it characterizes all of the analysed domains uniquely. The software SHELXL and SAINT, for instance, are most frequently but exclusively used in Chemistry, Materials and Physics, whereas Excel is frequently used in almost all other research domains except for them. Overall, an increased role of applications that can be controlled *via* scripts rather than point and click software can be observed. *Schindler, Zapilko & Krüger (2020)* reported that the Programming Environment R superseded SPSS in an excerpt of articles in PLoS One from 2017. While a similar trend can be seen here, the particular ranks did not change yet, see Fig. 15. While the usage of SPSS, Excel, and SAS remained constant over the last 5 years at the relative level, usages of R and Python increased. Considering articles from PLoS One only, R replaced SPSS at the top position, which confirms the result and suggests journal specific software preferences.

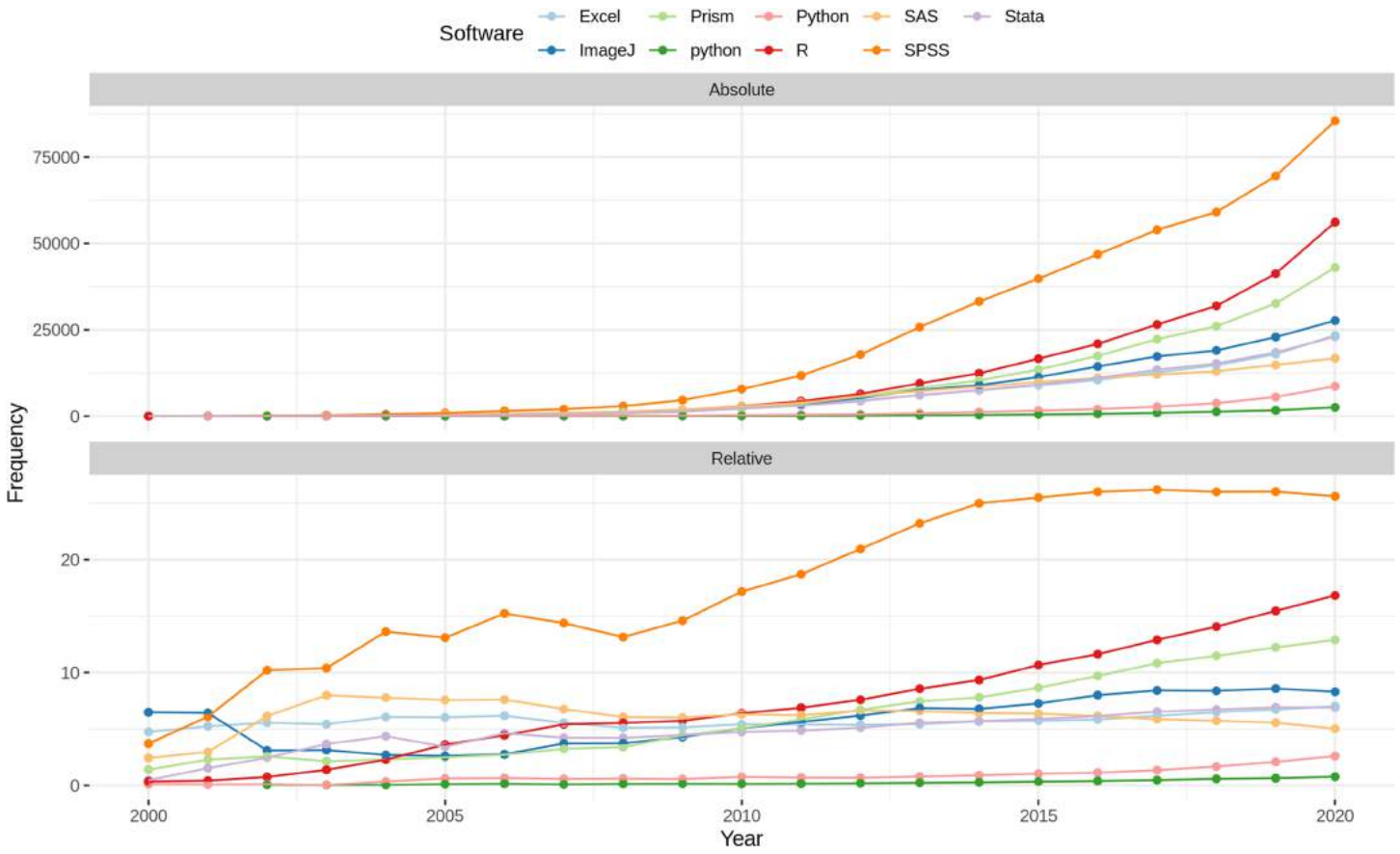


Figure 15 **Relative and absolute amount of articles per year mentioning the top statistical software.**
Full-size DOI: 10.7717/peerj-cs.835/fig-15

Article level statistics

The importance of software increased in recent years for both, the actual investigation as well as the reporting within the scholarly publication. This is suggested by the increasing trend of including software mentions into the textual description and the increasing number of different software per article. A reason could be the growing complexity of data driven analyses requiring more software to be employed, coupled with a high awareness about transparency and reproducibility in general. The positive correlation between journal rank and number of different software supports this by suggesting strong rigor in the description of the analysis[2]. The positive correlation of the number of software and the number of citations per article indicates a growing appreciation of the traceability of the described research processes. The observed domain specific difference in software usage could reflect the role of data in those domains. While in Arts and Humanities and Economics only few articles mention only few software suggesting an important role of manual data analysis, in Mathematics, Computer Science, Decision Science, and Agricultural and Biological Science many articles mention multiple different software indicating automatic and complex data collection and analyses.

[2] When interpreting the journal rank as an indicator of journal quality and thus for review quality.

Software citation completeness

Citation completeness has not improved over recent years, that is, the information provided to identify the particular software is not provided to full extent. This suggests a lack of awareness about the necessity to understand and reproduce research processes and its requirement for identifying particular software versions. When comparing formal citation, *i.e.*, providing a formal literature reference, with in-text mention, we found contrary trends both over time and across categories. Here, we consider in-text citations as complete when version and developer are included, while formal citations are always considered sufficient. Following this definition, only about 38% of the articles in 2020 allow the unique identification of the used software based on the provided information. Software usages in the technical domains use formal citation more frequently in contrast to research domains related to medicine. We believe the reason to be that the latter more frequently employ other materials and devices and adapt the same citation style for all research objects other than scholarly publications. With respect to journal rank, we see a growing trend in the usage of formal software citation with rising rank and a contrary trend in the completeness of in-text software mention. We believe that this supports the statement of increasing rigor in scholarly review and the request for more traceable descriptions in higher quality journals. This is also supported by the slight increase of formal citation frequency and contrary decrease of citation completeness for rising citation count. In summary, our results indicate that software citation standards, as suggested by *Katz et al. (2021)* or *Smith, Katz & Niemeyer (2016)* have not been adequately adapted in scholarly publications, yet.

Types of software mention

Each software mention is classified according to mention type indicating the reason why the software was mentioned within the scholarly article and software type providing information about the particular kind of software. Analyses of disciplinary differences showed that most software is created and published by scientists from the technical domains (Mathematics, Engineering and Computer Science). This reflects the domains with the highest interest in automating complex calculations combined with the programming knowledge to implement new suited software applications. Further, software allusions without actual usage are also more common within scholarly publications from those disciplines indicating more description, discussion and comparison between software entities. On the other hand there are disciplines which mostly reuse existing scientific software such as Material Science, Nursing, Dentistry, and Veterinary.

When looking at the different kinds of software, we found an increasing trend of using PlugIns over recent years, see supplementary Fig. A1. The relative frequency of Application mentions, in contrast, declined. We see this as an indicator toward the usage and extension of established software frameworks. With respect to the host software, we found a notable overlap in the most frequently used software (Table 10) and the most important host software (Table 12). This includes the Programming Environments R and Matlab, but also the Applications ImageJ, Stata, and Excel. More than 19 K PlugIns were found for the Programming Environment R making it the most important host

software for scientific investigations. While this number seems high at first glance, inspecting the two most important package repositories, CRAN (https://cran.r-project.org/) and Bioconductor (https://www.bioconductor.org/) with 18,312 and 2,042 unique packages[3] indicates these results to be plausible. However, the comparably low FScores for the identification of PlugIns might have resulted in an overestimation of less frequent PlugIns. This growing interest in the Programming Environment R and its package universe was previously investigated (*Li, Yan & Feng, 2017*; *Li & Yan, 2018*), some results of which are confirmed here. In particular, we see an overlap for the most frequent R packages.

[3] Package counts for CRAN and Bioconductor were retrieved on October 4th, 2021.

Software creation and deposition

By analyzing the mention types Creation and Deposition, we were able to identify the most important targets for the publication of software. On the one hand this includes web services such as GitHub for general purpose software and CRAN for R packages, on the other hand software journals. Specifically designed repositories to host and assign Digital Object Identifiers (DOI) to scientific research data such as Zenodo are not commonly used for publishing scientific software with a share of <1% of depositions. While this allows to provide researchers with recommendations on where to publish their software and/or the corresponding description, it also enables the search for software. Moreover, the identification of Creation and Deposition allows to track the scientific software landscape with low latency. It has to be noted that the second most frequent deposition URL is the result of a false software mention detection and its propagation.

Summary

The importance of software in science has been growing in recent years, in both relative and absolute numbers. The awareness for providing all necessary information to enable the identification of the particular software by others, in contrast, remains unchanged. Software citation principles have not been adapted yet in scholarly publications. However, articles in higher ranked journals tend to more formal software citations instead of in-text citations, which reflects recent software citation recommendations (*Katz et al., 2021*). Articles in lower ranked journals provide more complete in-text citations, *i.e.*, Version and Developer. We identified domain specific software citation habits: Medicine related domains prefer in-text citation, while technical domains tend to more formal citations. Domain independent as well as domain specific software is used across most research domains, the top 10 of which represent domain specific characteristics. Most software mentioned in scholarly articles are software for statistical analysis, such as SPSS, R, and Prism. Interestingly, we identified an increased interest in the usage of PlugIns, which allow the problem specific extension of general purpose software. The most important representatives of them are the Programming Environments R, MATLAB, and Python. Finally, we confirmed GitHub as a central repository for scientific software, for both publication and re-use, as previously assumed (*Russell et al., 2018*).

Limitations of the study

The study presented in this article involves complex data processing and information extraction steps, where each of these is subject to limitations that are discussed in the following.

The articles in SoftwareKG cover a broad range of scientific disciplines, however, the selection of PMC as primary data source implies a bias towards Medicine. While the training data set SoMeSci itself is also taken from PMC, the selection might introduce domain specific biases. Therefore, the trends reported here, might be different if we look at a domain such as computer science in general. For instance, BLAST is the second most used software in computer science in our set, which would likely not be true when looking at computer science in general, as this software is primarily used in Bioinformatics. Another bias in the selection of articles is towards open access, as all article are from the PMC Open Access subset. Researchers choosing to publish under open access might also be supporters of open data movement and, therefore, have a better awareness for attributing other open work such as software.

For information extraction and disambiguation, we found high performance for all employed machine learning methods. However, it is important to consider error propagation between them. The given evaluation for software and mention type classification does take error propagation into account, but the results for RE and entity disambiguation do not. Therefore, the $F = 0.94$ performance for RE might overestimate the true performance as it relies on results of $F = 0.885$ entity recognition. For disambiguation we model the effects of false positive entities by data augmentation, but it is hardly possible to tell if this completely suppresses their effect and false negatives do directly influence disambiguation performance. Moreover, evaluation for disambiguation has proven to be challenging and the gold standard dataset alone is no good predictor for performance on large scale entity disambiguation. We, therefore, adjusted our evaluation method to take the large scale data into account, but note that further systematic evaluation is required for entity disambiguation.

Our analyses regarding domains, journal rank and citation count rely on external data and are, thus, influenced by their quality. For instance, only 86.7% of our data was covered by Scimago data on domains and journal rank. Regarding the external citation data we assumed completeness but in case article citations were missing from the list they were not included in the computed citation count. In case one article would miss completely it would be counted with a citation count of 0.

An analysis of the article types (*skg:documentType*) contained in SoftwareKG showed that aside the largest group of research articles it also covers review articles and abstracts, but also case reports or letters and several other categories. For each of the groups we did find publications that cite software, but the prior probability for software mentions across article types differs. Therefore, it is important to note that the reported results are not specific to only research articles but to the distribution of scientific publications indexed in the PMC OA subset. The information about the article type, however, is included in the KG enabling others to analyse these effects.

CONCLUSION

In this article we presented the largest analysis of software usage in scholarly publications over the longest duration covering articles between 1990 and 2021[4]. Software mentions were identified by automatic information extraction covering NER for software and associated information, software and mention type classification, and RE between software and additional information. Moreover, in difference to previous studies, software names were automatically disambiguated to allow reasoning about software usage even when the same software is referred to by different names. The analysis covers 3.2 M articles, mentioning a total of 11.8 M software.

From the extracted information, we created SoftwareKG, the largest KG describing software mentions in scholarly publications. The KG was created by re-using existing vocabularies and published under an Open Access license to support further research on the role of software in science. SoftwareKG consists of over 300 M triples and contains information about software, accompanying information as well as information about articles, journals, authors, and publishers.

We performed a large-scale analysis on SoftwareKG with respect to publication date, article domain, journal rank and article citation count in order to identify differences and trends in software mention. Overall, the results show that software usage has increased over the course of the last 10 years, but we found no change in citation completeness during this time frame. This leads us to believe that there is still a lack in awareness for software citation in science, even so software citation standards have been available and promoted since 2016, *e.g.*, by *Smith, Katz & Niemeyer (2016)*.

We also identified a trend towards using extendable software architectures instead of stand-alone software, especially in combination with the Programming Environment R. Overall, their design allows an easy extension and offers high flexibility. Especially adding functionality and publishing new packages or PlugIns is facilitated. We could also show this trend by analyzing which infrastructure is used by scientists to publish their software, with GitHub playing a central role, but CRAN and Bioconductor being especially important in combination with R.

In general, we show that there are many domain specific peculiarities in software usage. We showed that the amount of software usage as well as the most used software per domain and their application purpose varies significantly. Domain specific citation habits are also reflected in preferences to formal and informal software citation, ranging from 5–40% in formal citation contrasting to 1–35% software citation completeness with opposing trends.

Overall, we believe that SoftwareKG provides a valuable data source for further investigations about the role of software in science. One finding that should be further investigated is, for instance, the influence of journal rank on formal citation of software usage. The trend could, for instance, be explained by higher review quality and journal policies enforcing better software citation. Further insights could allow to give better recommendations for journals to encourage software citations habits.

In future work SoftwareKG can build the basis to further explore software usage in science, for instance, as a mapping for available software and newly established software.

[4] SoftwareKG actually contains articles from before 1990, but we restricted most analyses to time between 1990 and 2021.

It can also be used to track software usage and establish software impact measures. Furthermore, the investigation of formal software citations should be extended to include the citation targets. Currently, formal citations are recognized, but not further analysed. In the future we need to include a distinction between software citation and software article citation and model citation completeness within formal citations.

Software and Data

We implemented all machine learning models for information extraction in Python 3.9.5 (*Van Rossum & Drake, 2009*), utilizing the following packages: PyTorch 1.9.0 (*Paszke et al., 2019*) for deep learning models, Huggingface transformers 4.9.1 (*Wolf et al., 2020*) to load and fine-tune pre-trained BERT models, Gensim 4.0.1 (*Řehůřek & Sojka, 2010*) for pre-training word embeddings, scikit-learn 0.24.2 (*Pedregosa et al., 2011*) for implementation of RE models, articlenizer R-14.06.2021 (*Schindler, Zapilko & Krüger, 2020*, https://github.com/dave-s477/articlenizer) for preprocessing of scientific articles, and NLTK 3.6.2 (*Loper & Bird, 2002*) for feature extraction. Moreover, to extract JATS XML meta data we used lxml 4.6.3 (*Behnel, Faasen & Bicking, 2005*) and for knowledge graph construction rdflib 6.0.0 (RDFLib Team https://github.com/RDFLib/rdflib). For statistical analysis and generation of figures we used R 4.1.1 (*R Core Team, 2021*), utilizing tidy verse 1.3.1 (*Wickham et al., 2019*) for data processing and plotting and SPARQL 1.16 (*van Hage et al., 2013*) to access the KG interface. To setup a SPARQL endpoint for SoftwareKG we used OpenLink Virtuoso Open Source Edition 07.20.3229 (*OpenLink, 2021*), available as docker from https://hub.docker.com/r/tenforce/virtuoso/.

Source code for construction and analysis is published on GitHub at https://github.com/f-krueger/SoftwareKG-PMC-Analysis and the data for SoftwareKG (*Schindler et al., 2021a*) itself is available on Zenodo *via* https://doi.org/10.5281/zenodo.5553737. To facilitate reproducibility of our data analysis, a docker file including a suited R environment to execute analyses on SoftwareKG is included.

ABBREVIATIONS

KG	Knowledge Graph
IRR	Inter-Rater Reliability
PMC	PubMed Central
pp	percentage points
JATS	Journal Article Tag Suite
NER	Named Entity Recognition
RE	Relation Extraction

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was financially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the projects SFB 1270/2 (grant: 299150580) and ScienceLinker (grant: 404417453). Parts of the computation were done by using a

computer cluster funded by DFG (grant: 440623123). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) SFB 1270/2: 299150580.
ScienceLinker: 404417453.
DFG: 440623123.

Competing Interests

The authors declare that they have no competing interests.

Author Contributions

- David Schindler conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the paper, and approved the final draft.
- Felix Bensmann performed the experiments, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the paper, and approved the final draft.
- Stefan Dietze conceived and designed the experiments, authored or reviewed drafts of the paper, and approved the final draft.
- Frank Krüger conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the paper, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

The source code is available at GitHub: https://github.com/f-krueger/SoftwareKG-PMC-Analysis.

The data is available at Zenodo: Schindler, David, Bensmann, Felix, Dietze, Stefan, & Krüger, Frank. (2021). SoftwareKG-PMC (0.2) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5713973.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.835#supplemental-information.

REFERENCES

Allen A, Teuben PJ, Ryan PW. 2018. Schroedinger's code: a preliminary study on research source code availability and link persistence in astrophysics. *The Astrophysical Journal Supplement Series* **236(1)**:10 DOI 10.3847/1538-4365/aab764.

Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z. 2007. DBpedia: a nucleus for a web of open data. In: *The Semantic Web*. Berlin: Springer, 722–735 DOI 10.1007/978-3-540-76298-0_52.

Bach NV. 2021. Informatics research artifacts ontology. Ontology Specification for IRAO version 1.1.1. *Available at http://ontology.ethereal.cz/irao*.

Behnel S, Faasen M, Bicking I. 2005. lxml: XML and HTML with Python. *GitHub*. Version 4.6.3. *Available at https://github.com/lxml/lxml*.

Beltagy I, Lo K, Cohan A. 2019. SciBERT: a pretrained language model for scientific text. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong: Association for Computational Linguistics, 3615–3620 DOI 10.18653/v1/D19-1371.

D'Arcus B, Giasson F. 2009. Bibliographic ontology specification revision: 1.3. Ontology Specification for BIBO. *Available at https://bibliontology.com/*.

DCMI Usage Board. 2020. Dcmi metadata terms. Ontology Specification for DCT. *Available at http://dublincore.org/specifications/dublin-core/dcmi-terms/2020-01-20/*.

Devlin J, Chang M-W, Lee K, Toutanova K. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Vol. 1. Minneapolis: Association for Computational Linguistics, 4171–4186 DOI 10.18653/v1/N19-1423.

Du C, Cohoon J, Lopez P, Howison J. 2021. Softcite dataset: a dataset of software mentions in biomedical and economic research publications. *Journal of the Association for Information Science and Technology* **72(7)**:870–884 DOI 10.1002/asi.24454.

Duck G, Nenadic G, Filannino M, Brass A, Robertson DL, Stevens R. 2016. A survey of bioinformatics database and software usage through mining the literature. *PLOS ONE* **11(6)**:1–25 DOI 10.1371/journal.pone.0157989.

Garijo D, Ratnakar V, Gil Y, Khider D, Osorio M. 2019. The software description ontology. Revision: 1.4.0. Ontology Specification for SDO. *Available at https://w3id.org/okn/o/sd/1.4.0*.

Gil Y, Ratnakar V, Garijo D. 2015. Ontosoft: capturing scientific software metadata. In: *Proceedings of the 8th International Conference on Knowledge Capture, K-CAP 2015*, New York: Association for Computing Machinery DOI 10.1145/2815833.2816955.

Greuel G-M, Sperber W. 2014. swmath—an information service for mathematical software. In: *International Congress on Mathematical Software*. Berlin: Springer, 691–701 DOI 10.1007/978-3-662-44199-2_103.

GROBID. 2021. Grobid. *GitHub*. *Available at https://github.com/kermitt2/grobid*.

Guha RV, Brickley D, Macbeth S. 2016. Schema.org: evolution of structured data on the web. *Communications of the ACM* **59(2)**:44–51 DOI 10.1145/2844544.

Hellmann S, Lehmann J, Auer S, Brümmer M. 2013. Integrating NLP using linked data. In: Alani H, Kagal L, Fokoue A, Groth P, Biemann C, Parreira JX, Aroyo L, Noy N, Welty C, Janowicz K, eds. *The Semantic Web – ISWC 2013*. Berlin: Springer, 98–113 DOI 10.1007/978-3-642-41338-4_7.

Howison J, Bullard J. 2016. Software in the scientific literature: problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology* **67(9)**:2137–2155 DOI 10.1002/asi.23538.

Jones MB, Boettiger C, Cabunoc Mayes A, Smith A, Slaughter P, Niemeyer K, Gil Y, Fenner M, Nowak K, Hahnel M, Coy L, Allen A, Crosas M, Sands A, Hong NC, Katz DS, Goble C. 2017.

Codemeta: an exchange schema for software metadata. version 2.0. *GitHub*. *Available at https://github.com/codemeta/codemeta*.

Katz DS, Chue Hong NP, Clark T, Muench A, Stall S, Bouquin D, Cannon M, Edmunds S, Faez T, Feeney P, Fenner M, Friedman M, Grenier G, Harrison M, Heber J, Leary A, MacCallum C, Murray H, Pastrana E, Perry K, Schuster D, Stockhause M, Yeston J. 2021. Recognizing the value of software: a software citation guide. *F1000Research* **9**:1257 DOI 10.12688/f1000research.26932.2.

Kendall A, Gal Y, Cipolla R. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Piscataway: IEEE, 7482–7491.

Krüger F, Schindler D. 2020. A literature review on methods for the extraction of usage statements of software and data. *Computing in Science & Engineering* **22(1)**:26–38 DOI 10.1109/MCSE.2019.2943847.

Lee J, Yoon W, Kim S, Kim D, Kim S, So CH, Kang J. 2019. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics* **36(4)**:1234–1240 DOI 10.1093/bioinformatics/btz682.

Li K, Lin X, Greenberg J. 2016. Software citation, reuse and metadata considerations: an exploratory study examining lammps. *Proceedings of the Association for Information Science and Technology* **53(1)**:1–10 DOI 10.1002/pra2.2016.14505301072.

Li K, Yan E. 2018. Co-mention network of R packages: scientific impact and clustering structure. *Journal of Informetrics* **12(1)**:87–100 DOI 10.1016/j.joi.2017.12.001.

Li K, Yan E, Feng Y. 2017. How is R cited in research outputs? Structure, impacts, and citation standard. *Journal of Informetrics* **11(4)**:989–1002 DOI 10.1016/j.joi.2017.08.003.

Loper E, Bird S. 2002. Nltk: the natural language toolkit. In: *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, ETMTNLP '02.* Vol. 1. Stroudsburg: Association for Computational Linguistics, 63–70 DOI 10.3115/1118108.1118117.

Lopez P, Du C, Cohoon J, Ram K, Howison J. 2021. Mining software entities in scientific literature: document-level ner for an extremely imbalance and large-scale task. In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21). Virtual Event, QLD*. New York: ACM.

Ma X, Hovy E. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNS-CRF. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1064–1074 DOI 10.18653/v1/P16-1101.

Malone J, Brown A, Lister AL, Ison J, Hull D, Parkinson H, Stevens R. 2014. The software ontology (swo): a resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of Biomedical Semantics* **5(25)**:149 DOI 10.1186/2041-1480-5-25.

Manghi P, Bardi A, Atzori C, Baglioni M, Manola N, Schirrwagen J, Principe P. 2019. The OpenAIRE research graph data model. *Available at https://graph.openaire.eu/*.

Mayernik MS, Hart DL, Maull KE, Weber NM. 2017. Assessing and tracing the outcomes and impact of research infrastructures. *Journal of the Association for Information Science and Technology* **68(6)**:1341–1359 DOI 10.1002/asi.23721.

Miles A, Matthews B, Wilson M, Brickley D. 2005. Skos core: simple knowledge organisation for the web. In: *Proceedings of the 2005 International Conference on Dublin Core and Metadata Applications: Vocabularies in Practice, Dublin Core Metadata Initiative (DCMI '05)*.

Nangia U, Katz DS. 2017. Understanding software in research: Initial results from examining nature and a call for collaboration. In: *2017 IEEE 13th International Conference on e-Science (e-Science)*. Piscataway: IEEE, 486–487 DOI 10.1109/eScience.2017.78.

OpenLink. 2021. Virtuoso open-source edition. Virtuoso version 07.20.3229 on Linux (x86_64-pc-linux-gnu), Single Server Edition. *Available at http://vos.openlinksw.com/dataspace/owiki/wiki/VOS/VOSIndex?rev=7*.

Pan X, Yan E, Cui M, Hua W. 2018. Examining the usage, citation, and diffusion patterns of bibliometric mapping software: a comparative study of three tools. *Journal of Informetrics* **12(2)**:481–493 DOI 10.1016/j.joi.2018.03.005.

Pan X, Yan E, Wang Q, Hua W. 2015. Assessing the impact of software on science: a bootstrapped learning of software entities in full-text papers. *Journal of Informetrics* **9(4)**:860–871 DOI 10.1016/j.joi.2015.07.012.

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. 2019. Pytorch: an imperative style, high-performance deep learning library. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Vancouver, Canada, **32**:8026–8037.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. 2011. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research* **12**:2825–2830 DOI 10.5555/1953048.2078195.

Peroni S, Shotton D, Ashton J, Barton A, Gramsbergen E, Jacquemot MC. 2016. Datacite2rdf: mapping datacite metadata schema 3.1 terms to rdf. DOI 10.6084/m9.figshare.2075356.v1.

R Core Team. 2021. *R: a language and environment for statistical computing*. Vienna: The R Foundation for Statistical Computing. *Available at http://www.R-project.org/*.

Řehůřek R, Sojka P. 2010. Software framework for topic modelling with large corpora. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, 45–50.

Ruder S. 2017. An overview of multi-task learning in deep neural networks. *ArXiv*. *Available at https://arxiv.org/abs/1706.05098*.

Russell PH, Johnson RL, Ananthan S, Harnke B, Carlson NE. 2018. A large-scale analysis of bioinformatics code on GitHub. *PLOS ONE* **13(10)**:e0205898 DOI 10.1371/journal.pone.0205898.

Schindler D, Bensmann F, Dietze S, Krüger F. 2021a. SoftwareKG-PMC. *Available at https://doi.org/10.5281/zenodo.5553737*.

Schindler D, Bensmann F, Dietze S, Krüger F. 2021b. Somesci—a 5 star open data gold standard knowledge graph of software mentions in scientific articles. In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21)*. Virtual Event, QLD: Association for Computing Machinery DOI 10.1145/3459637.3482017.

Schindler D, Zapilko B, Krüger F. 2020. Investigating software usage in the social sciences: a knowledge graph approach. In: *The Semantic Web, number 12123 in Lecture Notes in Computer Science*. Heraklion: Springer International Publishing, 271–286 DOI 10.1007/978-3-030-49461-2_16.

Smith AM, Katz DS, Niemeyer KE. 2016. Software citation principles. *PeerJ Computer Science* **2**:e86 DOI 10.7717/peerj-cs.86.

Stenetorp P, Pyysalo S, Topić G, Ohta T, Ananiadou S, Tsujii J. 2012. BRAT: a web-based tool for NLP-assisted text annotation. In: *Proceedings of the Demonstrations at the 13th Conference of*

the European Chapter of the Association for Computational Linguistics. Avignon: Association for Computational Linguistics, 102–107.

van Hage WR, with contributions from: Kauppinen T, Graeler B, Davis C, Hoeksema J, Ruttenberg A, Bahls D. 2013. SPARQL: SPARQL client. R package version 1.16. *Available at https://rdrr.io/cran/SPARQL/*.

Van Rossum G, Drake FL. 2009. *Python 3 reference manual*. Scotts Valley: CreateSpace.

Vrandečić D. 2012. Wikidata. In: *Proceedings of the 21st International Conference Companion on World Wide Web - WWW 12 Companion*. New York: ACM Press.

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H. 2019. Welcome to the tidyverse. *Journal of Open Source Software* **4(43)**:1686 DOI 10.21105/joss.01686.

Wilder-James E. 2018. Description of a project. Ontology Specification for DOAP. *Available at https://github.com/ewilderj/doap/wiki*.

Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, Cistac P, Rault T, Louf R, Funtowicz M, Davison J, Shleifer S, von Platen P, Ma C, Jernite Y, Plu J, Xu C, Le Scao T, Gugger S, Drame M, Lhoest Q, Rush A. 2020. Transformers: state-of-the-art natural language processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Stroudsburg: Association for Computational Linguistics, 38–45 DOI 10.18653/v1/2020.emnlp-demos.6.

Xu J, Kim S, Song M, Jeong M, Kim D, Kang J, Rousseau JF, Li X, Xu W, Torvik VI, Bu Y, Chen C, Ebeid IA, Li D, Ding Y. 2020. Building a PubMed knowledge graph. *Scientific Data* **7(1)**:205 DOI 10.1038/s41597-020-0543-2.

PeerJ Computer Science

Research artifacts and citations in computer systems papers

Eitan Frachtenberg

Computer Science, Reed College, Portland, OR, United States of America

ABSTRACT

Research in computer systems often involves the engineering, implementation, and measurement of complex systems software and data. The availability of these artifacts is critical to the reproducibility and replicability of the research results, because system software often embodies numerous implicit assumptions and parameters that are not fully documented in the research article itself. Artifact availability has also been previously associated with higher paper impact, as measured by citations counts. And yet, the sharing of research artifacts is still not as common as warranted by its importance. The primary goal of this study is to provide an exploratory statistical analysis of the artifact-sharing rates and associated factors in the research field of computer systems. To this end, we explore a cross-sectional dataset of papers from 56 contemporaneous systems conferences. In addition to extensive data on the conferences, papers, and authors, this analyze dataset includes data on the release, ongoing availability, badging, and locations of research artifacts. We combine this manually curated dataset with citation counts to evaluate the relationships between different artifact properties and citation metrics. Additionally, we revisit previous observations from other fields on the relationships between artifact properties and various other characteristics of papers, authors, and venue and apply them to this field. The overall rate of artifact sharing we find in this dataset is approximately 30%, although it varies significantly with paper, author, and conference factors, and it is closer to 43% for conferences that actively evaluated artifact sharing. Approximately 20% of all shared artifacts are no longer accessible four years after publications, predominately when hosted on personal and academic websites. Our main finding is that papers with shared artifacts averaged approximately 75% more citations than papers with none. Even after controlling for numerous confounding covariates, the release of an artifact appears to increase the citations of a systems paper by some 34%. This metric is further boosted by the open availability of the paper's text.

Subjects Data Science, Databases, Digital Libraries, World Wide Web and Web Science

Keywords Reproducible research, Computer systems, Software artifacts, Bibliometrics, Open access, Software repositories, FAIR

Submitted 16 July 2021
Accepted 21 January 2022
Published 7 February 2022

Corresponding author
Eitan Frachtenberg,
etc_26@yahoo.com

Academic editor
Daniel de Oliveira

Additional Information and Declarations can be found on page 21

DOI **10.7717/peerj-cs.887**

INTRODUCTION

Many scientific experimental results cannot be successfully repeated or reproduced, leading to the so-called "reproducibility crisis" (*Baker, 2016b*; *Van Noorden, 2015*). An experimental result is not fully established unless it can be independently reproduced (*Stodden, 2008*), and an important step towards this goal is the sharing of artifacts associated with the work, including computer code (*ACM, 2020*; *Collberg & Proebsting, 2016*; *Fehr et al., 2016*). The availability of experimental artifacts is not only

crucial for reproducibility, but it also directly contributes to the transparency, reusability, and credibility of the work (*Feitelson, 2015*). Artifacts additionally play an important role in drive toward open science, which has gained substantial momentum in computer science (CS) (*Heumüller et al., 2020*).

Given the central role of research artifacts in reproducibility, it is not surprising to find major initiatives to increase artifact sharing and evaluation. As Childers and Chrysanthis wrote in 2017:

> Experimental computer science is far from immune [from the reproducibility crisis], although it should be easier for CS than other sciences, given the emphasis on experimental artifacts, such as source code, data sets, workflows, parameters, etc. The data management community pioneered methods at ACM SIGMOD 2007 and 2008 to encourage and incentivize authors to improve their software development and experimental practices. Now, after 10 years, the broader CS community has started to adopt Artifact Evaluation (AE) to review artifacts along with papers (*Childers & Chrysanthis, 2017*).

Unfortunately, the sharing and evaluation of artifacts are still not as commonplace as warranted by their importance. One challenge in addressing this topic is that definitions and expectations for research artifacts are not always clear. The Association of Computing Machinery (ACM) defines a paper's artifact as follows:

> By "artifact" we mean a digital object that was either created by the authors to be used as part of the study or generated by the experiment itself. For example, artifacts can be software systems, scripts used to run experiments, input datasets, raw data collected in the experiment, or scripts used to analyze results (*ACM, 2020*).

This paper aims to shed some light on artifact sharing in one particular field of CS, namely *computer systems* (or "systems" for short). Systems is a large research field with numerous applications, used by some of the largest technology companies in the world. For the purpose of this study, we define systems as the study and engineering of concrete computing systems, which includes research topics such as: operating systems, computer architectures, data storage and management, compilers, parallel and distributed computing, and computer networks.

The study of these topics often involves the implementation, modification, and measurement of *system software* itself. System software is software that is not on its own a user-facing application, but rather software that manages system resources or facilitates development for the actual applications, such as compilers, operating system components, databases, and middleware. System software can be fairly complex and tightly coupled to the system it is designed to run on.

Research artifacts, and especially software artifacts, are therefore paramount to the evaluation and reproduction of systems research. Whereas in other fields of science—or even CS—research results can often be replicated from the original equations or datasets, systems software can embody countless unstated assumptions in the code and parameters. The significance is that reproducing research findings by recreating its artifacts from the

terse descriptions in a paper is often unfeasible, rendering software artifacts all that more important.

The hypothesis of this paper is therefore that because of its importance to reproducibility, artifacts sharing in systems significantly increases a paper's influence, as measured by citations. Citations are not only a widely used metric of impact for papers and researchers but also an indirect measure of the work's quality and usefulness, which presumably are both helped by the availability of artifacts. Citations may also stand in as proxy metrics for the transparency, reusability, reproducibility, and credibility of papers—if we assume that any of these qualities encourage subsequent researchers to cite the original work. The main observational goal of this paper is to evaluate the quantitative association between artifacts availability and citations in the research field of computer systems. An additional goal of this study is an exploratory data analysis of associated factors to identify relationships between the availability of research artifacts in systems research papers and other statistical properties of these papers.

Study design and main findings

To evaluate our main hypothesis, this study uses an observational, cross-sectional approach, analyzing 2,439 papers from a large subset of leading systems conferences. The study population comes from a hand-curated collection of 56 peer-reviewed systems and related conferences from a single publication year (2017). Among other characteristics, it includes manually collected data on artifact availability and paper citation counts 3.5 years from publication, as detailed in the next section. By comparing the post-hoc citations of papers with released artifacts to those with none, we find that we can reject the null hypothesis that artifact availability does not impact paper citations. Even after controlling for demographic, paper, and conference factors using a multilevel mixed-effects model, papers with artifacts still receive 34% more citations on average than papers without.

Our expansive dataset also offers the opportunity for a descriptive and correlational study of the following additional questions. These questions are ancillary to the main research question of the relationship between artifacts and citations. Nevertheless, they may interest the reader and provide a fuller context and quantitative understanding of the state of artifact sharing in the field, and are provided as secondary contributions. These questions, and a short answer to each, are listed here and are elaborated in the results section:

1. What is the ratio of papers in systems for which artifacts are available? (Approximately 30%.)
2. How many of these artifacts are actually linked from the paper? (Approximately 80%.)
3. How many of these artifacts have expired since publication? What characterizes these artifacts? (Approximately 13% can no longer be found, mostly from academic and personal host pages.)
4. What are the per-conference factors and differences that affect the ratio of artifact sharing? (The most influential factor appears to be an artifact evaluation process. Approximately 57% of papers in these six conferences shared artifacts.)

5. Does conference prestige affect artifact availability? (Papers that release artifacts tend to appear in more competitive conferences.)
6. What is the relationship between artifact accessibility and paper accessibility? (Papers with shared artifacts are also more likely to have an eprint version freely available online, and sooner than non-artifact papers.)
7. What is the relationship between artifact accessibility and paper awards? (Approximately 39% of papers with awards shared artifacts *vs.* 27% in the rest.)
8. Are there any textual properties of the paper that can predict artifact availability? (Papers that share artifacts tend to be longer and incorporate a computer system moniker in their titles.)

As a final contribution, this study provides a rich dataset of papers (*Frachtenberg, 2021*), tagged with varied metadata from multiple sources, including for the first time artifact properties (described next). Since comprehensive data on papers with artifacts is not always readily available, owing to the significant manual data collection involved, this dataset can serve as the basis of additional studies.

The rest of this paper is organized as follows. The next section presents in detail the dataset, methodology, and limitations of our study. An extensive set of descriptive and explanatory statistics is presented in the results section and then used to build a mixed-effects multilevel regression model for citation count. The discussion section presents potential implications from these findings, as well as potential threats to the validity of this analysis. These findings are then placed in historical and cross-disciplinary context in the related-work section. Finally, the concluding section summarizes the main findings of this study and suggests some future research directions.

DATA AND METHODS

The most time-consuming aspect of this study was the collection and cleaning of the data. This section describes the data selection and cleaning process for paper, artifact, and citation data.

The primary dataset we analyze comes from a hand-curated collection of 56 peer-reviewed systems and related conferences from a single publication year (2017), to reduce time-related variance. Conference papers were preferred over journal articles because in CS, and in particular, in its more applied fields such as systems, original scientific results are typically first published in peer-reviewed conferences (*Patterson, Snyder & Ullman, 1999*; *Patterson, 2004*):health, and then possibly in archival journals, sometimes years later (*Vrettas & Sanderson, 2015*). These conferences were selected to represent a large cross-section of the field, with different sizes, competitiveness, and subfields (Table 1). Such choices are necessarily subjective, based on the author's experience in the field. But they are aspirationally both wide enough to represent the field well and focused enough to distinguish it from the rest of CS.

A few of these conferences, such as MobiCom and SLE, specifically encouraged artifacts in their call-for-papers or websites. Four conferences—SC, OOPSLA, PLDI, and SLE—archived their artifacts in the ACM's digital library. In addition to general

Table 1 System conferences, including start date, number of published papers, total number of named authors, and acceptance rate.

Conference	Date	Papers	Authors	Acceptance	Conference	Date	Papers	Authors	Acceptance
ICDM	2017-11-19	72	269	0.09	PACT	2017-09-11	25	89	0.23
KDD	2017-08-15	64	237	0.09	SPAA	2017-07-24	31	84	0.24
SIGMETRICS	2017-06-05	27	101	0.13	MASCOTS	2017-09-20	20	75	0.24
SIGCOMM	2017-08-21	36	216	0.14	CCGrid	2017-05-14	72	296	0.25
SP	2017-05-22	60	287	0.14	PODC	2017-07-25	38	101	0.25
PLDI	2017-06-18	47	173	0.15	CLOUD	2017-06-25	29	110	0.26
NDSS	2017-02-26	68	327	0.16	Middleware	2017-12-11	20	91	0.26
NSDI	2017-03-27	42	203	0.16	EuroPar	2017-08-30	50	179	0.28
IMC	2017-11-01	28	124	0.16	PODS	2017-05-14	29	91	0.29
ISCA	2017-06-24	54	295	0.17	ICPP	2017-08-14	60	234	0.29
SOSP	2017-10-29	39	217	0.17	ISPASS	2017-04-24	24	98	0.30
ASPLOS	2017-04-08	56	247	0.18	Cluster	2017-09-05	65	273	0.30
CCS	2017-10-31	151	589	0.18	OOPSLA	2017-10-25	66	232	0.30
HPDC	2017-06-28	19	76	0.19	HotOS	2017-05-07	29	112	0.31
MICRO	2017-10-16	61	306	0.19	ISC	2017-06-18	22	99	0.33
MobiCom	2017-10-17	35	164	0.19	HotCloud	2017-07-10	19	64	0.33
ICAC	2017-07-18	14	46	0.19	HotI	2017-08-28	13	44	0.33
SC	2017-11-13	61	325	0.19	SYSTOR	2017-05-22	16	64	0.34
CoNEXT	2017-12-13	32	145	0.19	ICPE	2017-04-22	29	102	0.35
SIGMOD	2017-05-14	96	335	0.20	HotStorage	2017-07-10	21	94	0.36
PPoPP	2017-02-04	29	122	0.22	IISWC	2017-10-02	31	121	0.37
HPCA	2017-02-04	50	215	0.22	CIDR	2017-01-08	32	213	0.41
EuroSys	2017-04-23	41	169	0.22	VEE	2017-04-09	18	85	0.42
ATC	2017-07-12	60	279	0.22	SLE	2017-10-23	24	68	0.42
HiPC	2017-12-18	41	168	0.22	HPCC	2017-12-18	77	287	0.44
SIGIR	2017-08-07	78	264	0.22	HCW	2017-05-29	7	27	0.47
FAST	2017-02-27	27	119	0.23	SOCC	2017-09-25	45	195	Unknown
IPDPS	2017-05-29	116	447	0.23	IGSC	2017-10-23	23	83	Unknown

encouragement and archival, six conferences specifically offered to evaluate artifacts by a technical committee: OOPSLA, PACT, PLDI, PPoPP, SC, and SLE.

For each conference, we gathered various statistics from its web page, proceedings, or directly from its chairs. We also collected historical conference metrics from the websites of the ACM, the Institute of Electrical and Electronics Engineers (IEEE), and Google Scholar (GS), including past citations, age, and total publications, and downloaded all papers in PDF format. The dataset includes extensive data on the authors and the textual properties of the papers, and the relevant features are discussed in the next section.

We are also interested in measuring the post-hoc impact of each paper, as approximated by its number of citations. Citation metrics typically lag publication by a few months or years, allowing for the original papers to be discovered, read, cited, and then the citations themselves published and recorded. The time duration since these papers had been published, approximately 3.5 years, permits the analysis of their short-to-medium term impact in terms of citations. In practice, this duration is long enough that only 46 papers (1.89%) have gathered no citations yet.

For this study, the most critical piece of information on these papers is their artifacts. Unfortunately, most papers included no standardized metadata with artifact information, so it had to be collected manually from various sources, as detailed next.

The only existing form of standardized artifact metadata was for the subset of conferences organized by the ACM with artifact badge initiatives. In the proceedings page in the ACM's digital library of these conferences, special badges denote which papers made artifacts available, and which papers had artifacts evaluated (for conferences that supported either badge). In addition, the ACM digital library also serves as a repository for the artifacts, and all of these ACM papers included a link back to the appropriate web page with the artifact.

Unfortunately, most papers in this dataset were not published by the ACM or had no artifact badges. In the absence of artifact metadata or an automated way to extract artifact data, these papers required a manual scanning of the PDF text of every paper in order to identify such links. When skimming these papers, several search terms were used to assist in identifying artifacts, namely: "github", "gitlab", "bitbucket", "sourceforge", and "zenodo" for repositories; variants of "available", "open source", and "download" for links; and variations of "artifact", "reproducibility", and "will release" for indirect references. Some papers make no mention of artifacts in the text, but we can still discover associated artifacts online by searching github.com for author names, paper titles, and especially unique monikers used in the paper to identify their software.

We also recorded for each paper: whether the paper had an "artifact available" badge or "artifact evaluated" badge, whether a link to the artifact was included in the text, the actual URL for the artifact, and the latest date that this artifact was still found intact online. All of the searches for these artifacts are recent, so from the last field above we can denote the current status of an artifact as either *extant* or *expired*. From the availability of a URL, we can classify an artifact as *released* or *unreleased* (the latter denoting papers that promised an artifact but no link or repository was found). And from the host domain of the URL we can classify the location of the artifact as either an *Academic* web page, the *ACM* digital library, a *Filesharing* service such as Dropbox or Google, a specialized *Repository* such as github.com, *Other* (including .com and .org web sites), or *NA*.

In all, 722 papers in our dataset (29.6%) had an identifiable or promised artifact, predominantly as software but occasionally as data, configuration, or benchmarking files. Artifacts that had been included in previous papers or written by someone other than the paper's authors were excluded from this count. This statistic only reflects artifact availability, not quality, since evaluating artifact quality is both subjective and time-consuming. It is worth noting, however, that most of the source-code repositories in these artifacts showed

no development activity—commits, forks, or issues—after the publication of their paper, suggesting limited activity for the artifacts alone.

Data-collection procedure

The following list summarizes the data-collection process for reproducibility purposes.

1. Visit the website and proceedings of each conference and record general information about the conference: review policy, open-access, rebuttal policy, acceptance rate, program committee, *etc.*
2. Also from these sources, manually copy the following information for each paper: title, author names, and award status (as noted on the website and in proceedings).
3. Double-check all paper titles and author names by comparing conference website and post-conference proceedings. Also compare titles to GS search results and ensure all papers are (eventually) discovered by GS with the title corrected as necessary. Finally, check the same titles and author names against the Semantic Scholar database and resolve any discrepancies.
4. Download the full text of each paper in PDF format *via* institutional digital library access.
5. Record all papers with artifact badges. These are unique to the ACM conferences in our dataset and are clearly shown both in the ACM digital library and in the PDF copy of such papers.
6. Collect and record GS citation counts for each paper as close to possible to exactly 42 months after the conference's opening day. The dataset includes citation counts for each paper across multiple time points, but the analysis in this paper only uses one data point per paper, closest to the selected duration.
7. Record artifact availability and links for papers. This is likely the most time-consuming and error-prone process in the preparation of the data specific to this study and involves the following steps: Using a search tool on each document (''pdfgrep'') on each of the search terms listed above and perusing the results to identify any links or promises to artifacts; skimming or reading papers with negative results to ensure such a link was not accidentally missed; Finally, searching github.com for specific system names if a paper describes one, even if not linked directly from the paper.

Statistics

For statistical testing, group means were compared pairwise using Welch's two-sample t-test; differences between distributions of two categorical variables were tested with χ^2 test; and comparisons between two numeric properties of the same population were evaluated with Pearson's product-moment correlation. All statistical tests are reported with their p-values.

Ethics statement

All of the data for this study was collected from public online sources and therefore did not require the informed consent of the papers' authors.

Table 2 Class of artifact URLs. 'NA' locations indicate expired or unreleased URLs.

Location	Count
Repository	477
Academic	81
Other	67
ACM	44
Filesharing	6
NA	47

Code and data availability

The complete dataset and metadata are available in the supplementary material, as well as a github repository (*Frachtenberg, 2021*).

RESULTS

Descriptive statistics

Before addressing our main research question, we start with a simple characterization of the statistical distributions of artifacts in our dataset. Of the 722 papers with artifacts, we find that about 79.5% included an actual link to the artifact in the text. The ACM digital library marked 88 artifact papers (12.2%) with an "Artifact available" badge, and 89 papers (12.3%) with an "Artifact evaluated" badge. The majority of artifact papers (86.7%) still had their artifacts available for download at the time of this writing. This ratio is somewhat similar to a comparable study that found that 73% of URLs in five open-access (OA) journals were live after five years (*Saberi & Abedi, 2012*). Of the 722 papers that promised artifacts, 47 appear to have never released them. The distribution of the location of the accessible artifacts is shown in Table 2, and is dominated by Github repositories.

Looking at the differences across conferences, Fig. 1 shows the percentage of papers with artifacts per conference, ranging from 0% for ISCA, IGSC, and HCW to OOPSLA's 78.79% (mean: 27.22%, SD: 19.32%). Unsurprisingly, nearly all of the conferences where artifacts were evaluated are prominent in their relatively high artifact rates. Only PACT stands out as a conference that evaluated artifacts but had a lower-than-average overall ratio of papers with artifacts (0.24). The MobiCom conference also shows a distinctly low ratio, 0.09, despite actively encouraging artifacts. It should be noted, however, that many papers in PACT and MobiCom are hardware-related, where artifacts are often unfeasible. The same is true for a number of other conferences with low artifact ratios, such as ISCA, HPCA, and MICRO. Also worth noting is the fact that ACM conferences appear to attract many more artifacts than IEEE conferences, although the reasons likely vary on a conference-by-conference basis.

Another indicator for artifact availability is author affiliation. As observed in other systems papers, industry-affiliated authors typically face more restrictions for sharing artifacts (*Collberg & Proebsting, 2016*), likely because the artifacts hold commercial or competitive ramifications (*Ince, Hatton & Graham-Cumming, 2012*). In our dataset, only

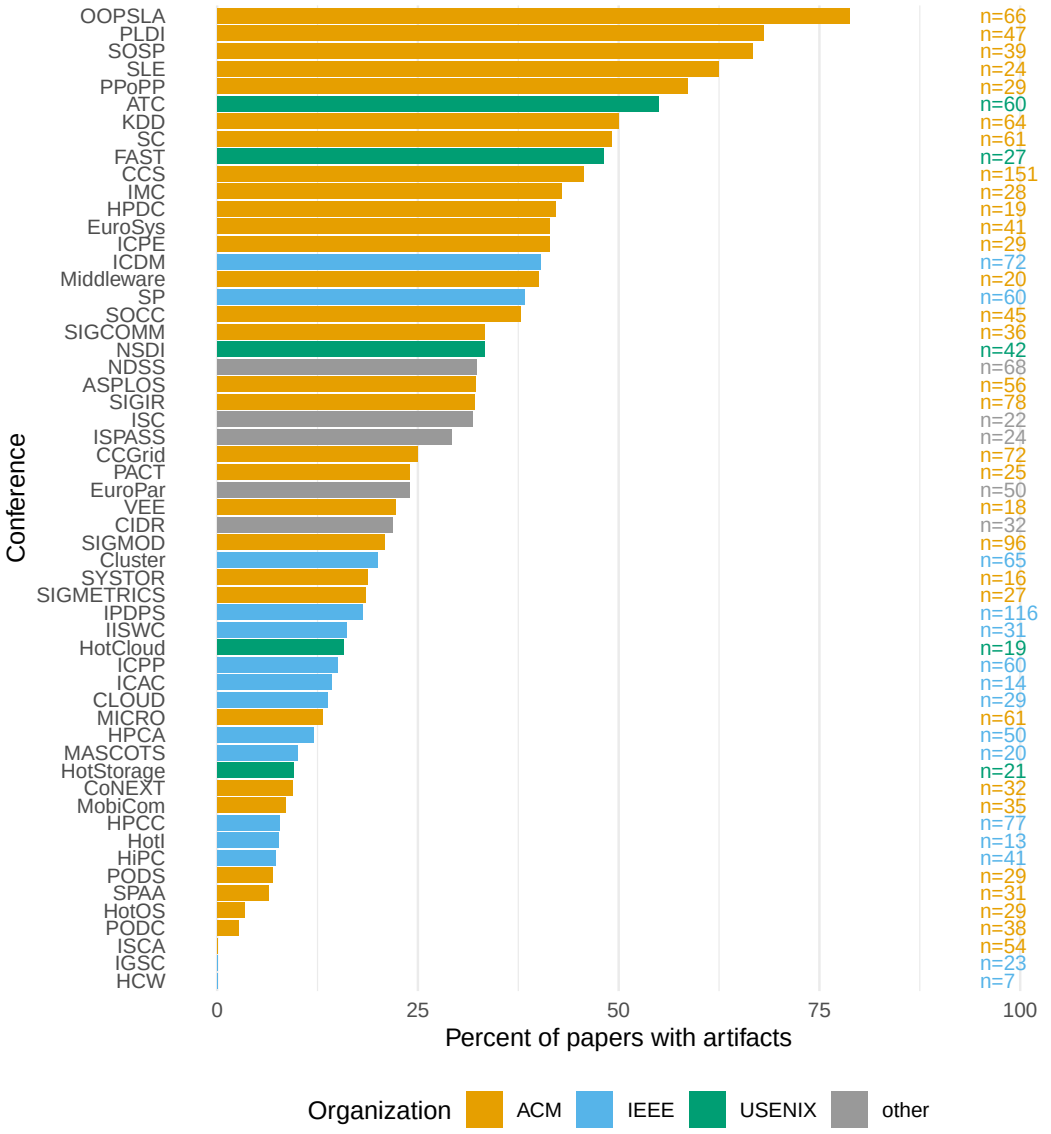


Figure 1 **Papers with artifact by conference.**

Full-size DOI: 10.7717/peerjcs.887/fig-1

19.3% of the 109 papers where all authors had an industry affiliation also released an artifact, compared to 28.1% for the other papers ($\chi^2 = 3.6$, $p = 0.06$).

Relationships to citations

Turning now to our main research hypothesis, we ask: does the open availability of an artifact affect the citations of a paper in systems? To answer this question, we look at the distribution of citations for each paper 42 months after its conference's opening day, when its proceedings presumably were published[1].

[1] At the time of this writing during summer 2021, the papers from December 2017 had been public for 3.5 years, so this 42-month duration was selected for all papers to normalize the comparison.

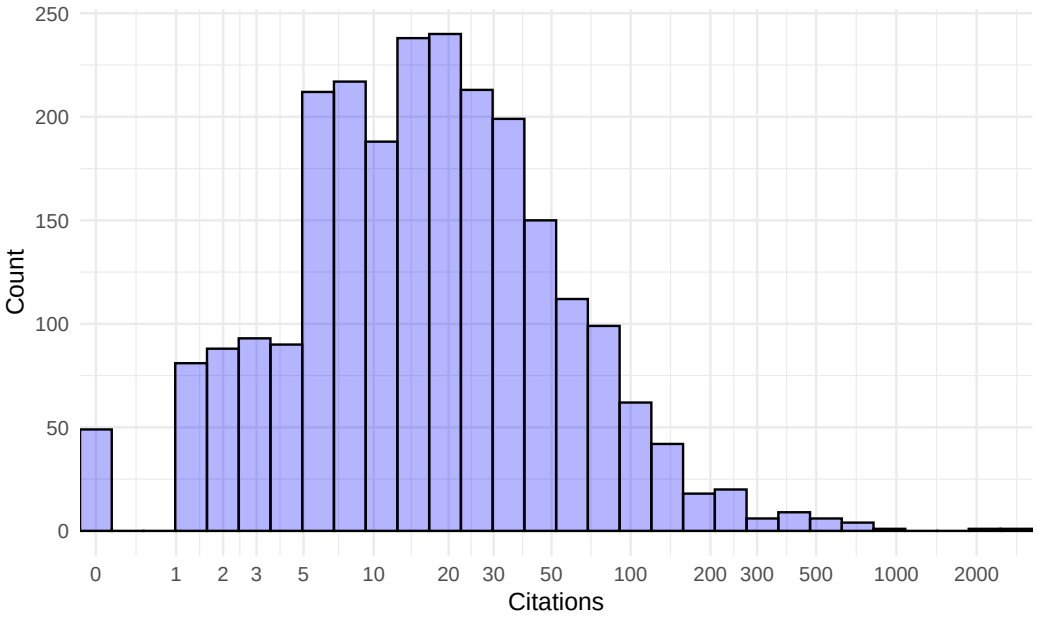


Figure 2 **Distribution of paper citations 42 months after publication (log-scale).**
Full-size DOI: 10.7717/peerjcs.887/fig-2

Figure 2 shows the overall paper distribution as a histogram, while Fig. 3 breaks down the distributions of artifact and non-artifact papers as density plots.

Citations range from none at all (49 papers) to about a thousand, with two outlier papers exceeding 2,000 citations (*Carlini & Wagner, 2017*; *Jouppi et al., 2017*). The distributions appear roughly log-normal. The mean citations per paper with artifacts released was 50.7, compared to 29.1 with none ($t = 4.07$, $p < 10^{-4}$). Since the citation distribution is so right-skewed, it makes sense to also compare the median citations with and without artifacts (25 *vs.* 13, $W = 767739$, $p < 10^{-9}$). Both statistics suggest a clear and statistically significant advantage in citations for papers that released an artifact. Likewise, the 675 papers that actually released an artifact garnered more citations than the 47 papers that did promise an artifact that could later not be found ($t = 3.82$, $p < 10^{-3}$), and extant artifacts fared better than expired ones ($t = 4.17$, $p < 10^{-4}$).

In contradistinction, some positive attributes of artifacts were actually associated with fewer citations. For example, the mean citations of the 573 papers with a linked artifact, 47, was much lower than the 71.3 mean for the 102 papers with artifacts we found using a Web search ($t = -2.02$, $p = 0.04$; $W = 22865$, $p < 10^{-3}$). Curiously, the inclusion of a link in the paper, presumably making the artifact more accessible, was associated with fewer citations.

Similarly counter-intuitive, papers that received an "Artifact evaluated" badge fared worse in citations than artifact papers who did not ($t = -3.32$, $p < 0.01$; $W = 11932.5$, $p = 0.03$). Papers who received an "Artifact available" badge did fare a little worse than artifact papers who did not ($t = -1.45$, $p = 0.15$; $W = 26050.5$, $p = 0.56$). These findings

Figure 3 **Density plot of paper citations 42 months after publication (log-scale).**
Full-size DOI: 10.7717/peerjcs.887/fig-3

Table 3 **Median citations by class of artifact URLs for extant artifacts.**

Location	Count	Median citations
Repository	477	25
Academic	81	24
Other	67	27
ACM	44	15
Filesharing	6	35

appear to contradict the premise that such badges are associated with increased artifact sharing, as has been found in other fields (*Baker, 2016a*).

Finally, we can also break down the citations per paper grouped by the type of location for the artifact and by its organization, examining medians because of the outsize effects of outliers (Table 3). The three major location categories do not show significant differences in citations, and the last two categories may be too small to ascribe statistical significance to their differences.

Accessibility

One commonly used set of principles to assess research software artifacts is termed FAIR: findability, accessibility, interoperability, and reusability (*Hong et al., 2021*; *Wilkinson et al., 2016*). We have overviewed the findability aspect of artifacts in the statistics of how many of these were linked or found *via* a Web search. The reusability and interoperability of artifacts unfortunately cannot be assessed with the current data. But we can address some of our secondary research questions by analyzing the accessibility of artifacts in depth.

As mentioned previously, 13.3% of released artifacts are already inaccessible, a mere ≈ 3.5 years after publication. Most of the artifacts in our dataset were published in code repositories, predominantly github, that do not guarantee persistent access or even universal access protocols such as digital object identifiers (DOI). However, only 2.3% of the "Repository" artifacts were inaccessible. In contrast, 22.2% of the artifacts in university pages have already expired, likely because they had been hosted by students or faculty that have since moved elsewhere. Also, a full half of the artifacts on file-sharing sites such as Dropbox or Google Drive are no longer there, possibly because these are paid services or free to a limited capacity, and can get expensive to maintain over time.

Accessibility is also closely related to the findability of the artifact, which in the absence of artifact DOIs in our dataset, we estimate by looking at the number of papers that explicitly link to their artifacts. The missing (expired) artifacts consisted of a full 31.1% of the papers with no artifact link, compared to only 8.7% for papers that linked to them ($\chi^2 = 49.15$, $p < 10^{-9}$).

Another related question to artifact accessibility is how accessible is the actual paper that introduced the artifact, which may itself be associated with higher citations (*Gargouri et al., 2010*; *McCabe & Snyder, 2015*; *McKiernan et al., 2016*; *Tahamtan, Afshar & Ahamdzadeh, 2016*). A substantial proportion of the papers (23.1%) were published in 15 open-access conferences. Other papers have also been released openly as preprints or *via* other means. One way to gauge the availability of the paper's text is to look it up on GS and see if an accessible version (eprint) is linked, which was recorded in our dataset. Of the 2,439 papers, 91.8% displayed at some point an accessible link to the full text on GS. Specifically, of the papers that released artifacts, 96.7% were associated with an eprint as well, compared to 90% of the papers with no artifacts ($\chi^2 = 29$, $p < 10^{-7}$).

Moreover, our dataset includes not only the availability of an eprint link on GS, but also the approximate duration since publication (in months) that it took GS to display this link, offering a quantitative measure of accessibility speed. It shows that for papers with artifacts, GS averaged approximately 4 months post-publication to display a link to an eprint, compared to 5.8 months for papers with no artifacts ($t = -5.48$, $p < 10^{-7}$). Both of these qualitative and quantitative differences are statistically significant, but keep in mind that the accessibility of papers and artifacts are not independent: some conferences that encouraged artifacts were also open-access, particularly those with the ACM. Another dependent covariate with accessibility is citations; several studies suggested that accessible papers are better cited (*Bernius & Hanauske, 2009*; *Niyazov et al., 2016*; *Snijder, 2016*), although others disagree (*Calver & Bradley, 2010*; *Davis & Walters, 2011*; *McCabe & Snyder, 2015*). This dependence may explain part of the higher citability of papers with artifacts, as elaborated next.

Covariate analysis

Having addressed the relationships between artifacts and citations, we can now explore relationships between additional variables from this expansive dataset.

Awards

Many conferences present competitive awards, such as "best paper", "best student paper", "community award", *etc.* Of the 2,439 total papers, 4.7% received at least one such award. Papers with artifacts are disproportionately represented in this exclusive subset (39.5% *vs.* 27.1% in non-award papers; $\chi^2 = 7.71$, $p < 0.01$).

Again, it is unclear whether this relationship is causal since the two covariates are not entirely independent. For example, a handful of awards specifically evaluated the contribution of the paper's artifact. Even if the relationship is indeed causal, its direction is also unclear, since 20% of award papers with artifacts did not link to it in the paper. It is possible that these papers released their artifacts after winning the award or because of it.

Textual properties

Some of the textual properties of papers can be estimated from their full text using simple command-line tools. Our dataset includes three such properties: the length of each paper in words, the number of references it cites, and the existence of a system's moniker in the paper's title.

The approximate paper length in words and the number of references turn out to be positively associated with the release of an artifact. Papers with artifacts average **more pages** than papers without (13.98 *vs.* 12.4; $t = 8.24$, $p < 10^{-9}$), **more words** (11757.36 *vs.* 10525.22; $t = 7.86$, $p < 10^{-9}$), and **more references** (32.31 *vs.* 28.71; $t = 5.25$, $p < 10^{-6}$). Keep in mind, however, that longer papers also correspond to more references ($r = 0.48$, $p < 10^{-9}$), and are further confounded with specific conference factors such as page limits.

As previously mentioned, many systems papers introduce a new computer system, often as software. Sometimes, these papers name their system by a moniker, and their title starts with the moniker, followed by a colon and a short description (*e.g.*, "Widget: An Even Faster Key-Value Store"). This feature is easy to extract automatically for all paper titles.

We could hypothesize that a paper that introduces a new system, especially a named system, would be more likely to include an artifact with the code for this system, quite likely with the same repository name. Our data support this hypothesis. The ratio of artifacts released in papers with a labeled title, 41.9%, is nearly double that of papers without a labeled title, 22.8% ($\chi^2 = 84.23$, $p < 10^{-9}$).

The difficulty to ascribe any causality to these textual relationships could mean that there is little insight to be gained from them. But they can clue the paper's reader to the possibility of an artifact, even if one is not linked in the paper. Indeed, they accelerated the manual search for such unlinked artifacts during the curation of the data assisted for this study.

Conference prestige

Next, we look at conference-specific covariates that could represent how well-known or competitive a conference is. In addition to textual conference factors, these conference metrics may also be associated with higher rates of artifact release.

Several proxy metrics for prestige appear to support this hypothesis. Papers with released artifacts tend to appear in conferences that average a **lower acceptance rate** (0.21 *vs.* 0.24; $t = -6.28$, $p < 10^{-9}$), **more paper submissions** (360.5 *vs.* 292.45; $t = 6.33$, $p < 10^{-9}$),

higher historical mean citations per paper (16.6 *vs.* 14.96; $t = 3.09$, $p < 0.01$), and a **higher h5-index** from GS metrics (46.04 *vs.* 41.04; $t = 6.07$, $p < 10^{-8}$). Also note that papers in conferences that offered some option for author response to peer review (often in the form of a rebuttal) were slightly more likely to include artifacts, perhaps as a response to peer review ($\chi^2 = 2.03$, $p = 0.15$).

To explain these relationships, we might hypothesize that a higher rate of artifact submission would be associated with more reputable conferences, either because artifact presence contributes to prestige, or because more rigorous conferences are also more likely to expect such artifacts. Observe, however, that some of the conferences that encourage or require artifacts are not as competitive as the others. For example, OOPSLA, with the highest artifact rate, had an acceptance rate of 0.3, and SLE, with the fourth-highest artifact rate, had an acceptance rate of 0.42. The implication here is that it may not suffice for a conference to actively encourage artifacts for it to be competitive, but a conference that already is competitive may also attract more artifacts.

Regression model

Finally, we combine all of these factors to revisit in depth our primary research interest: the effect of artifact sharing on citations. We already observed a strong statistical association between artifact release and higher eventual citations. As cautioned throughout this study, such associations are insufficient to draw causal conclusions, primarily because there are many confounding variables, most of which relating to the publishing conference. These confounding factors could provide a partial or complete statistical explanation to differences in citations beyond artifact availability.

In other words, papers published in the same conference might exhibit strong correlations that interact or interfere with our response variable. One such factor affecting paper citations is time since publication, which we control for by measuring all citations at exactly the same interval, 42 months since the conference's official start. Another crucial factor is the field of study—which we control for by focusing on a single field—while providing a wide cross-section of the field to limit the effect of statistical variability.

There are also numerous less-obvious paper-related factors that have shown positive association with citations, such as review-type studies, fewer equations, more references, statistically significant *positive* results, papers' length, number of figures and images, and even more obscure features such as the presence of punctuation marks in the title. We can attempt to control for such confounding variables when evaluating associations by using a multilevel model. To this end, we fit a linear regression model of citations as a function of artifact availability, and then add predictor variables as controls, observing their effect on the main predictor. The response variable we model for is *ln(citations)* instead of citations, because of the long tail of their distribution. We also omit the 49 papers with zero citations to improve the linear fit with the predictors.

In the baseline form, fitting a linear model of the log-transformed citations as a function of only artifact released yields an intercept (baseline log citations) of 2.6 and a slope of 0.59, meaning that releasing an artifact adds approximately 81% more citations to the paper, after exponentiation. The p-value for this predictor is exceedingly low (less than 2×10^{-16}) but

the simplistic model only explains 4.61% of the variance in citations (Adjusted $R^2 = 0.046$). The Bayesian Information Criterion (BIC) for this model is 7693.252, with 2388 degrees of freedom (df).

We can now add various paper covariates to the linear model in an attempt to get more precise estimates for the artifact released predictor, by iteratively experimenting with different predictor combinations to minimize BIC using stepwise model selection (*García-Portugués, 2021*, Ch. 3). The per-paper factors considered were: **paper length** (words), **number of coauthors**, **number of references**, **colon in the title**, **award** given, and **accessibility speed** (months to eprint[2]).

It turns out that all these paper-level factors except award given have a statistically significant effect on citations, which brings the model to an increased adjusted R^2 value of 0.285 and a BIC of 7028.07 ($df = 2{,}380$). However, the coefficient for artifact released went down to 0.35 (42% relative citation increase) with an associated p-value of 3.8×10^{-13}.

Similar to paper variables, some author-related factors such as their academic reputation, country of residence, and gender have been associated with citation count (*Tahamtan, Afshar & Ahamdzadeh, 2016*). We next enhance our linear model with the following predictor variables (omitting 451 papers with NA values):

- Whether all the coauthors with a known affiliation came from the same country (*Puuska, Muhonen & Leino, 2014*).
- Is the lead author affiliated with the United States (*Gargouri et al., 2010*; *Peng & Zhu, 2012*)?
- Whether any of the coauthors was affiliated with one of the top 50 universities per www.topuniversities.com (27% of papers) or a top company (if any author was affiliated with either (Google, Microsoft, Yahoo!, or Facebook: 18% of papers), based on the definitions of a similar study (*Tomkins, Zhang & Heavlin, 2017*).
- Whether all the coauthors with a known affiliation came from industry.
- The gender of the first author (*Frachtenberg & Kaner, 2021*).
- The sum of the total past publications of all coauthors of the paper (*Bjarnason & Sigfusdottir, 2002*).
- The maximum h-index of all coauthors (*Hurley, Ogier & Torvik, 2014*).

Only the maximum h-index and top-university affiliation had statistically significant coefficients, but hardly affected the overall model.[3] These minimal changes may not justify the increased complexity and reduced data size of the new model (because of missing data), so for the remainder of the analysis, we ignore author-related factors and proceed with the previous model.

We can now add the last level: venue factors. Conference (or journal) factors—such as the conference's own prestige and competitiveness—can have a large effect on citations, as discussed in the previous section. Although we can approximate some of these factors with some metrics in the dataset, there may also be other unknown or qualitative conference factors that we cannot model. Instead, to account for conference factors we next build a mixed-effects model, where all the previously mentioned factors become fixed effects and the conference becomes a random effect (*Roback & Legler, 2021*, Ch. 8).

[2] Papers with no eprint available at the time of this writing were assigned an arbitrary time to eprint of 1,000 months, but the regression analysis was not particularly sensitive to this choice.

[3] Note that the past publication counts and h-index are correlated ($r = 0.6$), ($p < 10^{-9}$), so one may cancel the other out.

Table 4 Estimated parameters for final multilevel mixed-effects model of ln(citations).

Factor	Coefficient	*p*-value
Intercept	1.74166	6.6e−31
Artifact released	0.29357	3.6e−10
Award given	0.00002	4.7e−02
Months to eprint	−0.00048	8.8e−09
References number	0.00907	1.3e−08
Coauthors number	0.06989	1.6e−19
Colon in title	0.13369	1.4e−03

This last model does indeed reduce the relative effect of artifact release on citations to a coefficient of 0.29 (95% confidence interval: 0.2–0.39). But this coefficient still represents a relative citation increase of about a third for papers with released artifacts (34%), which is substantial. We can approximate a *p*-value for this coefficient *via* Satterthwaite's degrees of freedom method using R's `lmerTest` package (*Kuznetsova, Brockhoff & Christensen, 2017*), which is also statistically significant at 3.5825×10^{-10}. The parameters for this final model are enumerated in Table 4. The only difference in paper-level factors is that award availability has replaced word count as a significant predictor, but realistically, both have a negligible effect on citations.

DISCUSSION

Implications

The regression model described in the preceding section showed that even with multiple controlling variables we observe a strong association between artifact release and citations. We can therefore ask, does this association allow for any causal or practical inferences? This association may still not suffice to claim causation due to hidden variables (*Lewis, 2018*), but it does support the hypothesis that releasing artifacts can indeed improve the prospects of a systems research paper to achieve wider acceptance, recognition, and scientific impact.

One implication of this model is that even if we assume no causal relation between artifact sharing and higher citation counts, the association is strong enough to justify a change in future scientometric studies of citations. Such studies often attempt to control for various confounders when attempting to explain or predict citations, and this strong link suggests that at least for experimental and data-driven sciences, the sharing of research artifacts should be included as an explanatory variable.

That said, there may be a case for a causal explanation with a clear direction after all. First, the model controls for many of the confounding variables identified in the literature, so the possibility of hidden, explanatory variables is diminished. Second, there is a clear temporal relationship between artifact sharing and citations. Artifact sharing invariably accompanies the publication of a paper, while its citations invariable follow months or years afterward. It is therefore plausible to expect that citation counts are influenced by artifact sharing and not the other way around.

If we do indeed assume causality between the two, then an important, practical implication also arises from this model, especially for authors wishing to increase their work's citations. There are numerous factors that authors cannot easily control, such as their own demographic factors, but fortunately, these turn out to have insignificant effects on citations. Even authors' choice of a venue to publish in, which does influence citations, can be constrained by paper length, scope match, dates and travel, and most importantly, the peer-review process that is completely outside of their control. But among the citation factors that authors can control, the most influential one turns out to be the sharing of research artifacts.

A causal link would then provide a simple lever for systems authors to improve their citations by an average of some 34%: share and link any available research artifacts. Presumably, authors attempting to maximize impact already work hard to achieve a careful study design, elaborate engineering effort, a well-written paper, and acceptance at a competitive conference. The additional effort of planning for and releasing their research artifact should be a relatively minor incremental effort that could improve their average citation count. If we additionally assume causality in the link between higher artifact sharing rates and acceptance to more competitive conferences, the effect on citations can be compounded.

Other potential implications of our findings mostly agree with our intuition and with previous findings in related studies, as described in the related-work. For example, all other things being equal, papers with open access and with long-lasting artifacts receive more citations.

Two factors that do not appear to have a positive impact on citations, at least in our dataset, are the receipt of artifact badges or the linking of artifacts in the paper. This is unfortunate because it implicitly discourages standardized or searchable metadata on artifacts, which is critical for studies on their effect, as described next.

Threats to validity

Perhaps the greatest challenge in performing this study or in replicating it is the fact that good metadata on research artifacts is either nonexistent or nonstandard. There is currently no automated or even manual methodology to reliably discover which papers shared artifacts, how they were they shared, and how long did they survive. There are currently several efforts underway to try to standardize artifact metadata and citation, but for this current study, the validity and scalability of the analysis hinge on the quality of the manual process of data collection.

One way to address potential human errors in data collection and tagging is to collect a sizeable dataset—as was attempted in this dataset—so that such errors disappear in the statistical noise. Although a large-enough number of artifacts was identified for statistical analysis, there likely remain untagged papers in the dataset that did actually release an artifact (false negatives). Nevertheless, there is no evidence to suggest that their number is large or that their distribution is skewed in some way as to bias statistical analyses. Moreover, since the complete dataset is (naturally) released as an artifact of this paper, it can be enhanced and corrected over time.

Additionally, there is the possibility of errors in the manual process of selecting conferences, importing data about papers and authors, disambiguating author names, and identifying the correct citation data on GS. In the data-collection process, we have been careful to cross-validate the data we input against the one found in the official proceedings of each conference, as well as the data that GS recorded, and reconciled any differences we found.

Citation metrics were collected from the GS database because it includes many metrics and allows for manual verification of the identity of each author by linking to their homepage. This database is not without its limitations, however. It does not always disambiguate author names correctly, and it tends to overcount publications and citations (*Halevi, Moed & Bar-Ilan, 2017*; *Harzing & Alakangas, 2016*; *Martin-Martin et al., 2018*; *Sugimoto & Lariviere, 2018*). The name disambiguation challenge was addressed by manually verifying the GS profiles of all researchers and ensuring that they include the papers from our dataset. Ambiguous profiles were omitted from our dataset. As for citation over-counting, note that the absolute number of citations is immaterial to this analysis, only the difference between papers with and without artifacts. Assuming GS overcounts both classes of papers in the same way, it should not materially change the conclusions we reached.

Our dataset also does not include data specific to self-citations. Although it is possible that papers with released artifacts have different self-citations characteristics, thus confounding the total citation count, there is no evidence to suggest such a difference. This possibility certainly opens up an interesting question for future research, using a citation database with reliable self-citation information (unlike GS).

RELATED WORK

This paper investigates the relationship between research artifacts and citations in the computer systems field. This relationship has been receiving increasingly more attention in recent years for CS papers in general. For example, a new study on software artifacts in CS research observed that while artifact sharing rate is increasing, the bidirectional links between artifacts and papers do not always exist or last very long, as we have also found (*Hata et al., 2021*). Some of the reasons that researchers struggle to reproduce experimental results and reuse research code from scientific papers are the continuously changing software and hardware, lack of common APIs, stochastic behavior of computer systems, and a lack of a common experimental methodology (*Fursin, 2021*), as well as copyright restrictions (*Stodden, 2008*).

Software artifacts have often been discussed in the context of their benefits for open, reusable, and reproducible science (*Hasselbring et al., 2019*). Such results have led more CS organizations and conferences to increase adoption of artifact sharing and evaluation, including a few of the conferences evaluated in this paper (*Baker, 2016b*; *Dahlgren, 2019*; *Hermann, Winter & Siegmund, 2020*; *Saucez, Iannone & Bonaventure, 2019*). One recent study examined specifically the benefit of software artifacts for higher citation counts (*Heumüller et al., 2020*). Another study looked at artifact evaluation for CS papers and

found a small but positive correlation with higher citations counts for papers between 2013 and 2016 (*Childers & Chrysanthis, 2017*).

When analyzing the relationship between artifact sharing and citations, one must be careful to consider the myriad possibilities for confounding factors, as we have in our mixed-effects model. Many such factors have been found to be associated with higher citation counts. Some examples relating to the author demographics include the authors' gender (*Frachtenberg & Kaner, 2021*; *Tahamtan, Afshar & Ahamdzadeh, 2016*), country of residence (*Gargouri et al., 2010*; *Peng & Zhu, 2012*; *Puuska, Muhonen & Leino, 2014*), affiliation (*Tomkins, Zhang & Heavlin, 2017*), and academic reputation metrics (*Hurley, Ogier & Torvik, 2014*; *Bjarnason & Sigfusdottir, 2002*). Other factors were associated with the publishing journal or conference, such as the relative quality of the article and the venue (*McCabe & Snyder, 2015*) and others still related to the papers themselves, such as characteristics of the titles and abstracts, characteristics of references, and length of paper (*Tahamtan, Afshar & Ahamdzadeh, 2016*).

Among the many paper-related factors studied in relation to citations is the paper's text availability, which our data shows to be also linked with artifact availability. there exists a rich literature examining the association between a paper's own accessibility and higher citation counts, the so-called "OA advantage" (*Bernius & Hanauske, 2009*; *Davis & Walters, 2011*; *Sotudeh, Ghasempour & Yaghtin, 2015*; *Wagner, 2010*).

For example, Gargouri et al. found that articles whose authors have supplemented subscription-based access to the publisher's version with a freely accessible self-archived version are cited significantly more than articles in the same journal and year that have not been made open (*Gargouri et al., 2010*). A few other more recent studies and reviews not only corroborated the OA advantage but also found that the proportion of OA research is increasing rapidly (*Breugelmans et al., 2018*; *Fu & Hughey, 2019*; *McKiernan et al., 2016*; *Tahamtan, Afshar & Ahamdzadeh, 2016*). The actual amount by which open access improves citations is unclear, but one recent study found the number to be approximately 18% (*Piwowar et al., 2018*), which means that higher paper accessibility on its own is not enough to explain all of the citation advantage we identified for papers with available artifacts.

Turning our attention specifically to the field of systems, we might expect that many software-based experiments should be both unimpeded and imperative to share and reproduce (*Ince, Hatton & Graham-Cumming, 2012*). But instead we find that many artifacts are not readily available or buildable (*Collberg & Proebsting, 2016*; *Freire, Bonnet & Shasha, 2012*; *Heumüller et al., 2020*; *Krishnamurthi & Vitek, 2015*). A few observational studies looked at artifact sharing rates in specific subfields of systems, such as software engineering (*Childers & Chrysanthis, 2017*; *Heumüller et al., 2020*; *Timperley et al., 2021*) and computer architecture (*Fursin & Lokhmotov, 2011*), but none that we are aware of have looked across the entire field.

Without directly comparable information on artifact availability rates in all of systems or in other fields, it is impossible to tell whether the overall rate of papers with artifacts in our dataset, 27.7%, is high or low. However, within the six conferences that evaluated artifacts, 42.86% of papers released an artifact, a very similar rate to the $\approx$ 40% rate found

in a study of of a smaller subset of systems conferences with an artifact evaluation process (*Childers & Chrysanthis, 2017*).

In general, skimming the papers in our dataset revealed that many "systems" papers do in fact describe the implementation of a new computer system, mostly in software. It is plausible that the abundance of software systems in these papers and the relative ease of releasing them as software artifacts contributes directly to this sharing rate, in addition to conference-level factors.

CONCLUSION

Several studies across disparate fields found a positive association between the sharing of research artifacts and increased citation of the research work. In this cross-sectional study of computer systems research, we also observed a strong statistical relationship between the two, although there are numerous potential confounding and explanatory variables to increased citations. Still, even when controlling for various paper-related and conference-related factors, we observe that papers with shared artifacts receive approximately one-third more citations than papers without.

Citation metrics are a controversial measure of a work's quality, impact, and importance, and perhaps should not represent the sole or primary motivation for authors to share their artifacts. Instead, authors and readers may want to focus on the clear and important benefits to science in general, and to the increased reproducibility and credibility of their work in particular. If increased citation counts are not enough to incent more systems authors to share their artifacts, perhaps conference organizers can leverage their substantial influence to motivate authors. Although artifact evaluation can represent a nontrivial additional burden on the program committee, our data show that it does promote higher rates of artifact sharing.

While many obstacles to the universal sharing of artifacts still remain, the field of computer systems does have the advantage that many—-if not most—of its artifacts come in the form of software, which is easier to share than artifacts in other experimental fields. It is therefore not surprising that we find the majority of shared and extant artifacts in computer systems hosted on github.com, a highly accessible source-code sharing platform. That said, a high artifact sharing rate is not enough for the goals of reproducible science, since many of the shared artifacts in our dataset have since expired or have been difficult to locate.

Our analysis found that both the findability and accessibility of systems artifacts can decay significantly even after only a few years, especially when said artifacts are not hosted on dedicated open and free repositories. Conference organizers could likely improve both aspects by requiring—and perhaps offering—standardized tools, techniques, and repositories, in addition to the sharing itself. The ACM has taken significant steps in this direction by not only standardizing various artifact badges but also offering its own supplementary material repository in its digital library. A few conferences in our dataset, like SC, are taking another step in this direction by also requesting a standardized artifact description appendix and review for every technical paper, including a citeable link to the research artifacts.

To evaluate the impact of such efforts, we must look beyond the findability and accessibility of artifacts, as was done in this study. In future work, this analysis can be expanded to the two remaining aspects of the FAIR principles: interoperability and reusability, possibly by incorporating input from the artifact review process itself. The hope is that as the importance and awareness of research artifacts grows in computer systems research, many more conferences will require and collect this information, facilitating not only better, reproducible research, but also a better understanding of the nuanced effects of software artifact sharing.

ACKNOWLEDGEMENTS

I wish to thank Prof. Kelly McConville of Reed College for her thoughtful and patient assistance with the statistical analysis.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

The author received no funding for this work.

Competing Interests

The authors declare there are no competing interests.

Author Contributions

- Eitan Frachtenberg conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the paper, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

All source and data are available in the Supplemental Files.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.887#supplemental-information.

REFERENCES

ACM. 2020. Artifact review and badging version 1.1. *Available at https://www.acm.org/publications/policies/artifact-review-and-badging-current*.

Baker M. 2016a. 1,500 scientists lift the lid on reproducibility. *Nature News* **533(7604)**:452 DOI 10.1038/533452a.

Baker M. 2016b. Digital badges motivate scientists to share data. *Nature News* Epub ahead of print May 12 2016 DOI 10.1038/nature.2016.19907.

Bernius S, Hanauske M. 2009. Open access to scientific literature-increasing citations as an incentive for authors to make their publications freely accessible. In: *42nd Hawaii international conference on system sciences*. Piscataway: IEEE, 1–9 DOI 10.1109/HICSS.2009.335.

Bjarnason T, Sigfusdottir ID. 2002. Nordic impact: article productivity and citation patterns in sixteen Nordic Sociology departments. *Acta Sociologica* **45(4)**:253–267 DOI 10.1177/000169930204500401.

Breugelmans JG, Roberge G, Tippett C, Durning M, Struck DB, Makanga MM. 2018. Scientific impact increases when researchers publish in open access and international collaboration: A bibliometric analysis on poverty-related disease papers. *PLOS ONE* **13(9)**:e0203156 DOI 10.1371/journal.pone.0203156.

Calver MC, Bradley JS. 2010. Patterns of citations of open access and non-open access conservation biology journal papers and book chapters. *Conservation Biology* **24(3)**:872–880 DOI 10.1111/j.1523-1739.2010.01509.x.

Carlini N, Wagner D. 2017. Towards evaluating the robustness of neural networks. In: *Proceedings of the IEEE symposium on security and privacy (SP'17)*. Piscataway: IEEE, 39–57 DOI 10.1109/SP.2017.49.

Childers BR, Chrysanthis PK. 2017. Artifact evaluation: is it a real incentive? In: *2017 IEEE 13th international conference on e-science (e-Science)*. Piscataway: IEEE, 488–489 DOI 10.1109/eScience.2017.79.

Collberg C, Proebsting TA. 2016. Repeatability in computer systems research. *Communications of the ACM* **59(3)**:62–69 DOI 10.1145/2812803.

Dahlgren E. 2019. Getting research software to work: a case study on artifact evaluation for OOPSLA 2019. *Available at https://2019.splashcon.org/getImage/orig/accpub-OOPSLA2019-licensed.pdf* .

Davis PM, Walters WH. 2011. The impact of free access to the scientific literature: a review of recent research. *Journal of the Medical Library Association: JMLA* **99(3)**:208 DOI 10.3163/1536-5050.99.3.008.

Fehr J, Heiland J, Himpe C, Saak J. 2016. Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software. *AIMS Mathematics* **3**:261–281 DOI 10.3934/Math.2016.3.261.

Feitelson DG. 2015. From repeatability to reproducibility and corroboration. *ACM SIGOPS Operating Systems Review* **49(1)**:3–11 DOI 10.1145/2723872.2723875.

Frachtenberg E. 2021. Systems conferences analysis dataset.GitHub. *Available at http://github.com/eitanf/sysconf/* DOI 10.5281/zenodo.5590574.

Frachtenberg E, Kaner R. 2021. Representation of women in HPC conferences. In: *Proceedings of the international conference for high performance computing, networking, storage, and analysis (SC'21)*. St. Louis, MO DOI 10.1145/1122445.1122456.

Freire J, Bonnet P, Shasha D. 2012. Computational reproducibility: state-of-the-art, challenges, and database research opportunities. In: *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. New York: ACM 593–596 DOI 10.1145/2213836.2213908.

Fu DY, Hughey JJ. 2019. Meta-Research: Releasing a preprint is associated with more attention and citations for the peer-reviewed article. *Elife* **8**:e52646 DOI 10.7554/eLife.52646.

Fursin G. 2021. Collective knowledge: organizing research projects as a database of reusable components and portable workflows with common interfaces. *Philosophical Transactions of the Royal Society A* **379(2197)**:20200211 DOI 10.1098/rsta.2020.0211.

Fursin G, Lokhmotov A. 2011. Artifact evaluation for reproducible quantitative research. *Available at https://www.sigarch.org/artifact-evaluation-for-reproducible-quantitative-research/.*

García-Portugués E. 2021. Notes for predictive modeling. Version 5.8.6. *Available at https://bookdown.org/egarpor/PM-UC3M/.*

Gargouri Y, Hajjem C, Larivière V, Gingras Y, Carr L, Brody T, Harnad S. 2010. Self-selected or mandated, open access increases citation impact for higher quality research. *PLOS ONE* **5(10)**:e13636 DOI 10.1371/journal.pone.0013636.

Halevi G, Moed H, Bar-Ilan J. 2017. Suitability of Google Scholar as a source of scientific information and as a source of data for scientific evaluation—review of the literature. *Journal of Informetrics* **11(3)**:823–834 DOI 10.1016/j.joi.2017.06.005.

Harzing A-W, Alakangas S. 2016. Google scholar, scopus and the web of science: a longitudinal and cross-disciplinary comparison. *Scientometrics* **106(2)**:787–804 DOI 10.1007/s11192-015-1798-9.

Hasselbring W, Carr L, Hettrick S, Packer H, Tiropanis T. 2019. FAIR and open computer science research software. ArXiv preprint. arXiv:1908.05986.

Hata H, Guo JL, Kula RG, Treude C. 2021. Science-software linkage: the challenges of traceability between scientific knowledge and software artifacts. ArXiv preprint. arXiv:2104.05891.

Hermann B, Winter S, Siegmund J. 2020. Community expectations for research artifacts and evaluation processes. In: *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. New York: ACM 469–480 DOI 10.1145/3368089.3409767.

Heumüller R, Nielebock S, Krüger J, Ortmeier F. 2020. Publish or perish, but do not forget your software artifacts. *Empirical Software Engineering* **25(6)**:4585–4616 DOI 10.1007/s10664-020-09851-6.

Hong NPC, Katz DS, Barker M, Lamprecht A-L, Martinez C, Psomopoulos FE, Harrow J, Castro LJ, Gruenpeter M, Martinez PA, Struck THA, Lee A, Loewe A, Van Werkhoven B, Jones C, Garijo D, Plomp E, Genova F, Shanahan H, Leng J, Hellstrm M, Sandstrm M, Sinha M, Kuzak M, Herterich P, Zhang Q, Islam S, Sansone S-A, Pollard T, Williams UDAA, Czerniak A, Niehues A, Fouilloux AC, Desinghu B, Goble C, Richard C, Gray C, Erdmann C, Nst D, Tartarini D, Ranguelova E, Anzt H, Todorov I, McNally J, Moldon J, Burnett J, Garrido-Snchez J, Belhajjame K, Sesink L, Hwang L, Tovani-Palone MR, Wilkinson MD, Servillat M, Liffers M, Fox M, Miljkovi N, Lynch N, Lavanchy PM, Gesing S, Stevens S, Cuesta SM, Peroni S, Soiland-Reyes S, Bakker T, Rabemanantsoa T, Sochat V, Yehudi Y. 2021. FAIR

principles for research software (FAIR4RS principles). Harwell Oxford: Research Data Alliance DOI 10.15497/RDA00065.

Hurley LA, Ogier AL, Torvik VI. 2014. Deconstructing the collaborative impact: Article and author characteristics that influence citation count. *Proceedings of the American Society for Information Science and Technology* **50(1)**:1–10 DOI 10.1002/meet.14505001070.

Ince DC, Hatton L, Graham-Cumming J. 2012. The case for open computer programs. *Nature* **482(7386)**:485–488 DOI 10.1038/nature10836.

Jouppi NP, Young C, Patil N, Patterson D, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A, Boyle R, luc Cantin P, Chao C, Clark C, Coriell J, Daley M, Dau M, Dean J, Gelb B, Ghaemmaghami TV, Gottipati R, Gulland W, Hagmann R, Ho CR, Hogberg D, Hu J, Hundt R, Hurt D, Ibarz J, Jaffey A, Jaworski A, Kaplan A, Khaitan H, Koch A, Kumar N, Lacy S, Laudon J, Law J, Le D, Leary C, Liu Z, Lucke K, Lundin A, MacKean G, Maggiore A, Mahony M, Miller K, Nagarajan R, Narayanaswami R, Ni R, Nix K, Norrie T, Omernick M, Penukonda N, Phelps A, Ross J, Ross M, Salek A, Samadiani E, Severn C, Sizikov G, Snelham M, Souter J, Steinberg D, Swing A, Tan M, Thorson G, Tian B, Toma H, Tuttle E, Vasudevan V, Walter R, Wang W, Wilcox E, Yoon DH. 2017. In-datacenter performance analysis of a tensor processing unit. In: *Proceedings of the 44th annual international symposium on computer architecture (ISCA'17)*. 1–12 DOI 10.1145/3079856.3080246.

Krishnamurthi S, Vitek J. 2015. The real software crisis: repeatability as a core value. *Communications of the ACM* **58(3)**:34–36 DOI 10.1145/2658987.

Kuznetsova A, Brockhoff PB, Christensen R. HB. 2017. lmerTest package: tests in linear mixed effects models. *Journal of Statistical Software* **82(13)**:1–26 DOI 10.18637/jss.v082.i13.

Lewis CL. 2018. The open access citation advantage: does it exist and what does it mean for libraries? *Information Technology and Libraries* **37(3)**:50–65 DOI 10.6017/ital.v37i3.10604.

Martin-Martin A, Orduna-Malea E, Thelwall M, Lopez-Cozar ED. 2018. Google scholar, web of science, and scopus: a systematic comparison of citations in 252 subject categories. *Journal of Informetrics* **12(4)**:1160–1177 DOI 10.1016/j.joi.2018.09.002.

McCabe MJ, Snyder CM. 2015. Does online availability increase citations? Theory and evidence from a panel of economics and business journals. *Review of Economics and Statistics* **97(1)**:144–165 DOI 10.1162/REST_a_00437.

McKiernan EC, Bourne PE, Brown CT, Buck S, Kenall A, Lin J, McDougall D, Nosek BA, Ram K, Soderberg CK, Spies JR, Thaney K, Updegrove A, Woo KH, Yarkoni T. 2016. How open science helps researchers succeed. *eLife* **5**:e16800 DOI 10.7554/eLife.16800.

Niyazov Y, Vogel C, Price R, Lund B, Judd D, Akil A, Mortonson M, Schwartzman J, Shron M. 2016. Open access meets discoverability: citations to articles posted to Academia.edu. *PLOS ONE* **11(2)**:e0148257 DOI 10.1371/journal.pone.0148257.

Patterson DA. 2004. The health of research conferences and the dearth of big idea papers. *Communications of the ACM* **47(12)**:23–24 DOI 10.1145/1035134.1035153.

Patterson DA, Snyder L, Ullman J. 1999. Evaluating computer scientists and engineers for promotion and tenure. Computing Research News. *Available at http://archive2.cra.org/uploads/documents/resources/bpmemos/tenure_review.pdf* .

Peng T-Q, Zhu JJ. 2012. Where you publish matters most: a multilevel analysis of factors affecting citations of internet studies. *Journal of the American Society for Information Science and Technology* **63(9)**:1789–1803 DOI 10.1002/asi.22649.

Piwowar H, Priem J, Larivière V, Alperin JP, Matthias L, Norlander B, Farley A, West J, Haustein S. 2018. The state of OA: a large-scale analysis of the prevalence and impact of Open Access articles. *PeerJ* **6**:e4375 DOI 10.7717/peerj.4375.

Puuska H-M, Muhonen R, Leino Y. 2014. International and domestic co-publishing and their citation impact in different disciplines. *Scientometrics* **98(2)**:823–839 DOI 10.1007/s11192-013-1181-7.

Roback P, Legler J. 2021. *Beyond multiple linear regression: applied generalized linear models and multilevel models in R*. New York: CRC Press.

Saberi MK, Abedi H. 2012. Accessibility and decay of web citations in five open access ISI journals. *Internet Research* **22(2)**:234–247 DOI 10.1108/10662241211214584.

Saucez D, Iannone L, Bonaventure O. 2019. Evaluating the artifacts of SIGCOMM papers. *Computer Communication Review* **49(2)**:44–47 DOI 10.1145/3336937.3336944.

Snijder R. 2016. Revisiting an open access monograph experiment: measuring citations and tweets 5 years later. *Scientometrics* **109(3)**:1855–1875 DOI 10.1007/s11192-016-2160-6.

Sotudeh H, Ghasempour Z, Yaghtin M. 2015. The citation advantage of author-pays model: the case of Springer and Elsevier OA journals. *Scientometrics* **104(2)**:581–608 DOI 10.1007/s11192-015-1607-5.

Stodden V. 2008. The legal framework for reproducible scientific research: Licensing and copyright. *Computing in Science & Engineering* **11(1)**:35–40 DOI 10.1109/MCSE.2009.19.

Sugimoto CR, Lariviere V. 2018. *Measuring research: what everyone needs to know*. Oxford, UK: Oxford University Press.

Tahamtan I, Afshar AS, Ahamdzadeh K. 2016. Factors affecting number of citations: a comprehensive review of the literature. *Scientometrics* **107(3)**:1195–1225 DOI 10.1007/s11192-016-1889-2.

Timperley CS, Herckis L, Le Goues C, Hilton M. 2021. Understanding and improving artifact sharing in software engineering research. *Empirical Software Engineering* **26(4)**:1–41 DOI 10.1007/s10664-020-09901-z.

Tomkins A, Zhang M, Heavlin WD. 2017. Reviewer bias in single-versus double-blind peer review. *Proceedings of the National Academy of Sciences* **114(48)**:12708–12713 DOI 10.1073/pnas.1707323114.

Van Noorden R. 2015. Sluggish data sharing hampers reproducibility effort. *Nature News* Epub ahead of print Jun 3 2015 DOI 10.1038/nature.2015.17694.

Vrettas G, Sanderson M. 2015. Conferences versus journals in computer science. *Journal of the Association for Information Science and Technology* **66(12)**:2674–2684 DOI 10.1002/asi.23349.

Wagner AB. 2010. Open access citation advantage: an annotated bibliography. *Issues in Science and Technology Librarianship* **60**:2 DOI 10.5062/F4Q81B0W.

Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, Blomberg N, Boiten J-W, Da Silva Santos LB, Bourne PE, Bouwman J, Brookes AJ, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo CT, Finkers R, Gonzalez-Beltran A, Gray AJ, Groth P, Goble C, Grethe JS, Heringa J, t Hoen PA, Hooft R, Kuhn T, Kok R, Kok J, Lusher SJ, Martone ME, Mons A, Packer AL, Persson B, Rocca-Serra P, Roos M, Van Schaik R, Sansone S-A, Schultes E, Sengstag T, Slater T, Strawn G, Swertz MA, Thompson M, Van der Lei J, Van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft K, Zhao J, Mons B. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* **3(1)**:160018 DOI 10.1038/sdata.2016.18.

PeerJ Computer Science

Understanding progress in software citation: a study of software citation in the CORD-19 corpus

Caifan Du[1], Johanna Cohoon[1], Patrice Lopez[2] and James Howison[1]

[1] The University of Texas at Austin, Austin, TX, United States of America
[2] Science-miner, Naves, France

ABSTRACT

In this paper, we investigate progress toward improved software citation by examining current software citation practices. We first introduce our machine learning based data pipeline that extracts software mentions from the CORD-19 corpus, a regularly updated collection of more than 280,000 scholarly articles on COVID-19 and related historical coronaviruses. We then closely examine a stratified sample of extracted software mentions from recent CORD-19 publications to understand the status of software citation. We also searched online for the mentioned software projects and their citation requests. We evaluate both practices of referencing software in publications and making software citable in comparison with earlier findings and recent advocacy recommendations. We found increased mentions of software versions, increased open source practices, and improved software accessibility. Yet, we also found a continuation of high numbers of informal mentions that did not sufficiently credit software authors. Existing software citation requests were diverse but did not match with software citation advocacy recommendations nor were they frequently followed by researchers authoring papers. Finally, we discuss implications for software citation advocacy and standard making efforts seeking to improve the situation. Our results show the diversity of software citation practices and how they differ from advocacy recommendations, provide a baseline for assessing the progress of software citation implementation, and enrich the understanding of existing challenges.

Subjects Data Science, Social Computing
Keywords Software citation, Science policy, Scholarly communication

Submitted 8 September 2021
Accepted 7 June 2022
Published 25 July 2022

Corresponding author
Caifan Du, cfdu@utexas.edu

Academic editor
Silvio Peroni

Additional Information and Declarations can be found on page 30

DOI **10.7717/peerj-cs.1022**

INTRODUCTION

Software is crucial to research, but its visibility in scholarly records is problematic, undermining research policy goals (*Mayernik et al., 2017*; *Howison & Bullard, 2016*; *Bouquin et al., 2020*). These goals include facilitating more verifiable and reproducible research by explicitly referencing software in scholarly communications (*Howison & Bullard, 2016*), ensuring sufficient credit for research software work within the scientific reputation economy (*Bouquin et al., 2020*; *Howison & Herbsleb, 2011*), and tracking the research impact of software for decisions of funding and support (*Katz & Smith, 2015*; *Mayernik et al., 2017*; *Allen, Teuben & Ryan, 2018*). Making software visible in scholarly communication and evaluation can enable these goals. It is also instrumental to incentivize

high quality software work that buttresses more robust and useful research. Thus, there is a need to improve the visibility of software in research. Corresponding advocacy efforts have been gaining traction recently, pushing toward machine- and human-actionable software citations in scholarly communications.

A body of research has investigated software visibility in the scholarly record, with results largely confirming the frustration research software practitioners express (*e.g.*, *Howison & Bullard, 2016*; *Pan et al., 2018*; *Bouquin et al., 2020*). For instance, in contrast with the well established practice of citing publications, Howison and Bullard found through their examination of biology articles published between 2000 and 2010 that software citation "practices are varied and appear relatively ad hoc" (*Howison & Bullard, 2016*). They found 43% of mentions of software were "informal" mentions without bibliographical references, largely noting the software "name-only"; 39% of mentions appeared in bibliographies; 19% of mentions were presented in a "like instrument" manner which gave the software name and its (usually commercial) publisher with an address. Only 28.5% of the mentions were found to provide version information; only 6% of those versions could be found online. Similarly, *Pan et al. (2018)* traced specific cases of software through 481 papers and found "researchers mention and cite the tools in diverse ways, many of which fall short of a traditional formal citation". Finally, Bouquin and colleagues studied practices in the Astronomy literature, examining publisher-provided XML for over 76,000 articles published between 1995 and 2018 (*Bouquin et al., 2020*). Using a list of nine known software packages, they highlighted the "variation" in how these packages were mentioned. They identified diverse "software aliases" in different locations within papers, highlighting false positives due to name ambiguity (*e.g.*, a package named after a planet). They concluded that while software is valued, the inability to "systematically identify software citations" due to their variability can lead "people to doubt the value of citing software" and the authors highlight the importance of advocacy efforts to standardize practices.

These studies have also noted that it is important that software providers indicate how they would like to be credited within papers. Howison and Bullard found only 18% of mentioned packages provided a "request for citation", but these requests appeared to be effective: 68% of the mentions of those packages followed those requests. Bouquin and colleagues identified "preferred citations" statements made by software providers online for all their studied packages, reporting that these were often requesting multiple preferred citations or sometimes made inconsistent citation requests across different locations.

Advocacy around software citation has called for new (or clarified) practices among software providers, end-user researchers, and publication venue editors. The FORCE11 organization has been one venue for this work through the Software Citation Working Group (and its follow-up initiatives), leading to the publication of clear recommendations (*Smith, Katz & Niemeyer, 2016*; *Katz et al., 2019*; *Katz et al., 2021*; *Chue Hong et al., 2019*). This advocacy has also connected with similar issues in the sphere of research data, including DataCite (*Brase, Lautenschlager & Sens, 2015*) and FAIR principles (*Hong et al., 2021*). Additional work has focused on the accessibility of software metadata, such as through software catalogs (*Allen & Schmidt, 2014*; *Monteil et al., 2020*; *Muench et al., 2020*). Other efforts have encouraged unambiguous and machine-actionable

formats of "requests for citation", for instance, the CITATION.cff file (*Druskat et al., 2021*).

Finally, discovery techniques also have advanced for improving software visibility. Prototype systems such as Depsy (*Piwowar & Priem, 2016*), CiteAs (*Du et al., 2021b*), and the recently added software citation discovery feature on GitHub (*GitHub, 2021*) increase the chance that research software is identified and cited. Publication mining has also sought to identify software reported in research papers. *Krüger & Schindler (2020)* reviewed 18 studies that use different techniques to extract software mentions from articles. Machine learning (ML) approaches are the most likely to scale such discoveries. Accordingly, corpora with manual annotation of software mentions have been developed (*e.g.*, *Schindler et al., 2021*; *Du et al., 2021a*). Supervised ML-based software extraction over large collections with acceptable computational performance is now possible (*Lopez et al., 2021a*; *Lopez et al., 2021b*; *Schindler et al., 2022*; *Wade & Williams, 2021*). Community efforts titled "Habeus Corpus" was launched and continue to investigate these emerging collections of extracted software mentions (*Habeas Corpus, 2021*).

The growing capability to extract software mentions from publications at large scale now allows us to better assess how software is mentioned in scholarly records. Thus, in this study, we identify the baseline, opportunities, and challenges for the ongoing effort of making software visible in scholarship. For this purpose, we created a combined annotation scheme based on empirical descriptions of software citation practices and recommendations from advocacy. We used that scheme to annotate a stratified sample of a recently released collection of software mentions automatically extracted from the CORD-19 set of open access publications (*Lopez et al., 2021a*). In this way, we systematically document and analyze existing practices for software citation. Our annotation results enable us to understand the current status of software citation implementation, provide a baseline for future assessment, and compare with previous findings and advocacy recommendations. Through this analysis, we identify changes, challenges, and pathways for software citation implementation and advocacy.

RESEARCH QUESTIONS

1. How is software mentioned in recently published literature? Have these patterns changed from previous studies? How do current patterns compare to recently published guidelines for software citation?
2. How widespread are requests for citation? What form do they take? How do these requests compare to recently published guidance?

DATA & METHODS

To answer these research questions, we first obtained a sample of software mentions extracted from recent literature, leveraging a ML-based software entity recognition pipeline. Then, we assembled an annotation scheme built upon both past research and existing recommendations for software citation. We also annotated relevant practices that enable

the citation of software by looking for and examining the online records of mentioned software.

CORD-19 dataset

We used CORD-19, the COVID-19 Open Research Dataset (*Wang et al., 2020*), to obtain software mentions in recent publications. The CORD-19 was initially released in March 2020 by the Allen Institute for AI as a large-scale collection of publications and preprints on COVID-19 and past related coronavirus research. The initial release included about 28,000 papers; the dataset has been growing through regular updates. We used the 22 March 2021 release in the CORD-19 release history as our base corpus for software mention extraction, downloaded from https://www.semanticscholar.org/cord19/download. Out of its 490,904 bibliographical entries, this release includes 274,400 publications that can be uniquely identified by a distinct DOI (Digital Object Identifier) and 238,283 publications that can be uniquely identified by a distinct PubMed ID.[1] For reasons discussed below, we did not use the released extracted article content in JSON format, but harvested the open access PDFs instead using the identifiers of articles in the dataset as our starting point.

[1]Note these two sets have large overlaps. We present these numbers because there is no perfect global identifier in this corpus release.

The CORD-19 dataset is not noise-free (*Kanakia et al., 2020*). According to the metadata released along with the dataset, the earliest publication dates back to 1800s. Given our purpose of understanding the very recent software citation practices, we first extracted software mentions from the full corpus, then concentrated on the extracted mentions from articles published since 2016 for detailed analysis, the same year during which the Software Citation Principles (*Smith, Katz & Niemeyer, 2016*) was published. Metadata in the CORD-19 dataset release indicate that 80% of its contents were published after 2020, and 87% were published after 2016. This is because the contents of CORD-19 primarily focus on COVID-19, which emerged only in 2019. Thus, its large number of publications in recent years provide a rich base for us to investigate recent software citation practices.

Software mention extraction

We harvested the open access versions of these articles using the article metadata in the CORD-19 release, including both PDF and structured XML formats where available. This choice allows for more complete and reliable full-text extraction of software using our Softcite pipeline. The main components of the Softcite pipeline include three pieces of software: a full-text PDF harvester (*Article Dataset Builder, 2021*), a machine learning library for extracting structured content from scholarly PDFs (*GROBID, 2021*), and a software mention recognizer powered by a set of machine learning and deep learning models (*Softcite Software Mention Recognizer, 2021*).

Using the Softcite pipeline, we harvested more full-text articles than those released in the CORD-19 JSON corpus. Software mentions then were extracted from these reharvested open access publications. The extraction method is described in detail in (*Lopez et al., 2021b*). In short, the pipeline obtains PDFs, structures them using GROBID (*GROBID, 2021*; *Lopez, 2009*), and runs the software mention recognizer (*Softcite Software Mention Recognizer, 2021*) based on a fine-tuned SciBERT+CRF model (*Beltagy, Lo & Cohan, 2019*) trained on the softcite-dataset (*Du et al., 2021a*).

While *Wade & Williams (2021)* also published software mention extractions from CORD-19 based on the softcite-dataset, our extraction enhances the performance in a number of ways, mainly: (1) a more complete number of full-text articles in PDF were obtained, along with their DOI metadata retrieved from CrossRef API; (2) additional techniques to cope with the extreme label imbalance caused by the sparsity of software mentions in publications; (3) extraction of additional attributes (version, URL, publisher, context of mention); and (4) entity disambiguation, including normalization and using Wikidata to identify false positives (*e.g.*, the mention of algorithms but not a particular software instantiation). Finally, the pipeline also attaches in-text and bibliographic references to software mentions when available; these references are disambiguated against and matched with their registered CrossRef DOI. This final step allows us to more efficiently examine formal citation of software. Overall, the Softcite pipeline demonstrated good performance when recognizing mentions of software, its version, publisher, and/or URL mentioned together in text (average f1-score 79.1), with acceptable computational performance for processing very large collections of literature in PDF (evaluation reported in *Lopez et al., 2021b*).

The softcite-dataset that underlies our extraction pipeline of software mentions is human-curated annotations of biomedical and economic literature, with the majority of annotated software mentions identified in biomedical research publications. The shared focus on biomedicine therefore makes this training set line up well with the examination of the CORD-19 papers.

Descriptive statistics of extracted software

We published the full extraction results as the Softcite-CORD-19 dataset under CC-BY−4.0 (*Lopez et al., 2021a*). In this study, we used published version 0.2.1 (*Lopez et al., 2021a*), which is based on the CORD-19 dataset release on 22 March 2021. This version contains total 295,609 mentions of software, including their semantic and layout details, bibliographical references linking to the in-text software mentions (N = 55,407), and metadata of all the documents in which software mentions are recognized (N = 76,448; out of total 211,213 re-harvested open access full-text documents).

Figure 1 shows an overall breakdown of all the 295,609 software mentions in the Softcite-CORD-19 dataset and what (and how many) details are mentioned along in their article context. These extraction results are conditional on the extraction performance of our pipeline (*Lopez et al., 2021b*). About 50% of the extracted software mentions are solely names of the software without further details; 35% provide a version; 21% mention their publisher; and 9% have a URL given in the text.

Stratified sampling

To get an overview of how software has been mentioned in recent publications, we took software mentions extracted from the 61,175 articles (87%) published since 2016 in the Softcite-CORD-19 dataset as our sample frame. This sample frame contains 250,163 extracted software mentions. Due to the time when COVID-19 emerged, the sample frame is skewed toward software mentions in publications after 2019; software mentions from articles published since 2020 totals 85% of the extracted mentions in our sample frame.

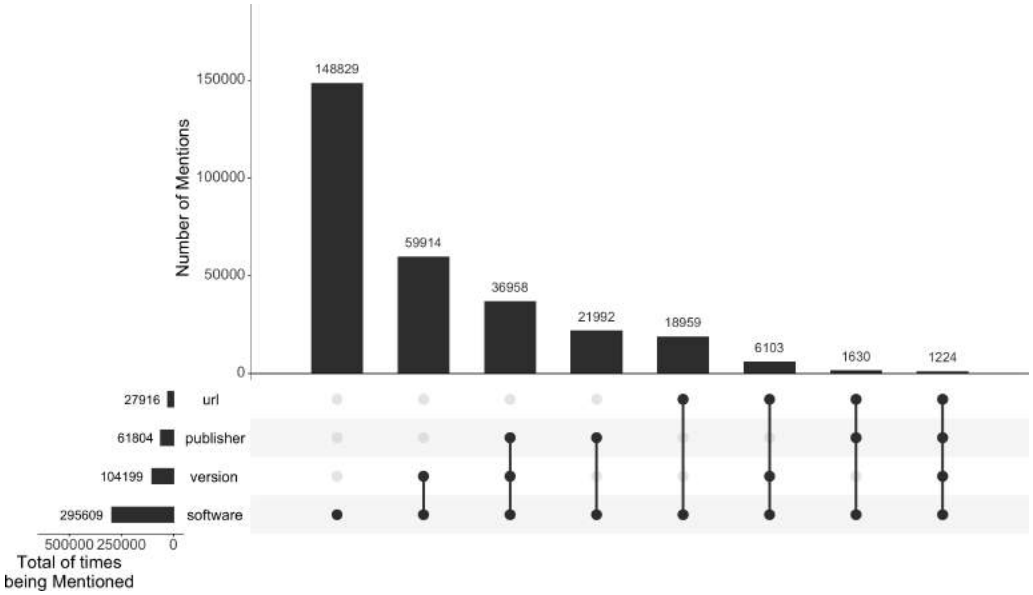


Figure 1 Number of extracted software mentions and their associated details in the Softcite-CORD-19 dataset. The figure was created using UpSetR (*Conway, Lex & Gehlenborg, 2017*).
Full-size DOI: 10.7717/peerjcs.1022/fig-1

We systematically constructed a random sample of software mentions from our sample frame, stratified by the *impact factor* of the article's publication venue as well as the article's *mention density*.

Impact factor

Because a scientific reader's attention is often concentrated on a few publication venues (*Bradford, 1934*; *Brookes, 1985*), these venues may have an outsized effect on their perception of the scientific reporting practices. The journal impact factor, developed by the Institute of Scientific Information (ISI) and annually calculated and published by the ISI Web of Science (WoS), is calculated based on the citations to one journal within a given period of time. Because citation to one journal is an outcome of one's scientific attention, we used the journal impact factor as the proxy of collective scientific attention. We matched the 6,997 distinct journal titles in the Softcite-CORD-19 dataset (version 0.2.1) using CrossRef DOI metadata to the 12,312 indexed journals in the ISI WoS 2020 Journal Citation Report. 78% of the publications (N = 47,959) in the sample frame were identified as articles from a venue with an indexed journal impact factor.

We then divided the publications in the sample frame into different strata based on the range that their journal impact factor ranking falls in: 1–10, 11–100, 101–1,000, 1,001–12,982, and a "No impact factor" group for those articles from venues without an indexed impact factor.[2] This stratification balanced the coverage of articles from journals that receive different levels of attention in our sample. This choice also enabled us to examine software citation practices in comparison with the prior analysis of 90 biological publications from 2000 to 2010 in (*Howison & Bullard, 2016*). In that study, 90 articles from journals indexed in the 2010 ISI WoS biology-related subject headings were

[2]There are also numerous preprints in the CORD-19 corpus.

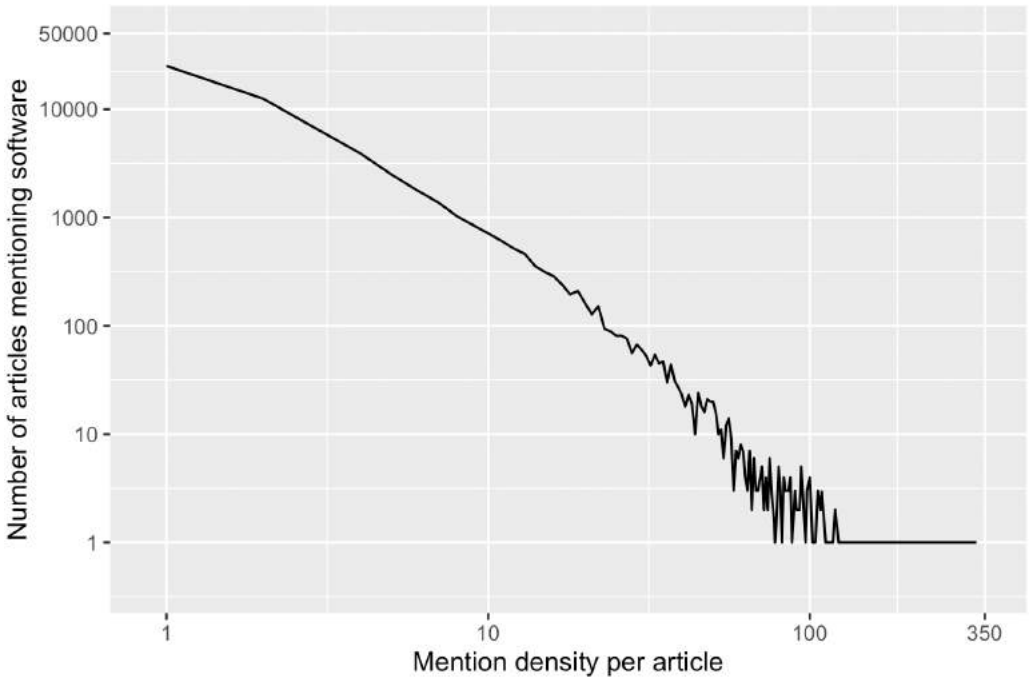


Figure 2 In our sample frame, number of articles mentioning software varies by mention density per article. 25,116 articles in the sample frame (41%) had only one software mention, while one particular article had the most software mention ($N = 330$). Only 11% of the articles in the sample frame had more than 9 software mentions.

Full-size DOI: 10.7717/peerjcs.1022/fig-2

randomly sampled by a 3-tier journal impact stratification: journals with impact factor ranked 1 through 10, then those ranked 11–100, finally those ranked 111–1,455.

Mention density

The number of software mentions extracted per article, *i.e.*, the *mention density*, varied significantly in the sample frame. The average mention density is 4.1 with a standard deviation of 7.2, and ranges from one to 330 mentions per article. In Fig. 2, this variance in our sample frame is further illustrated. Because of this variation, we also stratified our sample by mention density.

The variation of mention density across articles could be the result of changing practices of mentioning software over time, different genres of publications across venues and domains, requirements of journals, and/or writing conventions of authors, and so forth. Figure 3 demonstrates the distribution of mention density in different impact strata: Articles in the top impact factor stratum have a narrower distribution of mention density. Articles in lower/no journal impact factor strata have more articles with more than eight software mentions; these strata also have outlier articles where more than 100 software mentions are recognized. To ensure representation of these different distributions, we grouped the articles in our sample frame into three subsets by their mention density. The resulting sample frame and the number of articles in each stratum are summarized in

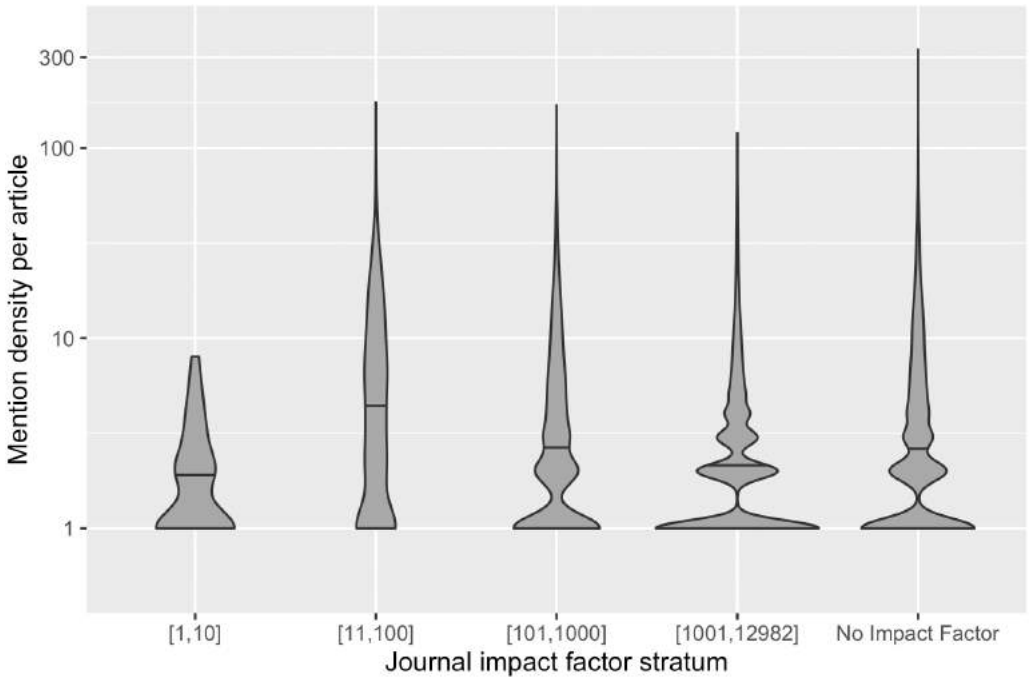


Figure 3 The distribution of mention density in each journal impact factor stratum. The horizontal line within each violin plot denotes the median mention density in the distribution.
Full-size DOI: 10.7717/peerjcs.1022/fig-3

Table 1 Number of articles and their percentage in the sample frame from each sample stratum.

Mention density impact strata	[0, 1] (41.06%)	[2, 8] (48.13%)	[9, 350] (10.82%)
[1, 10] (0.05%)	17	16	0
[11, 100] (0.95%)	179	259	142
[101, 1,000] (9.31%)	2,201	2,708	784
[1,001, 12,982] (51.14%)	13,606	15,291	2,385
No Impact Factor (38.56%)	9,113	11,168	3,306

Table 1, which also shows the uneven distribution of articles across different strata. Our stratification therefore supports more balanced sampling.

We examined annotation results in each stratum after annotation. However, we did not find substantial differences across any strata, so we do not report results broken down by these strata in this article.

We randomly sampled 15 articles from each stratum. Because there is no article in the stratum with the highest mention density ([9,350]) and highest journal impact rank ([1,10]), we skipped this stratum and obtained a sample of 210 articles from the remaining 14 strata. Next, we randomly sampled one software mention from all the extracted mentions of each article. Given that the CORD-19 dataset is a somewhat noisy corpus, when we encountered an article that is not a research article (*e.g.*, scientific news postings), we moved to the next article in the same sample stratum and randomly sampled an

extracted mention from this article to replace the original one. Accordingly, we gained a sample of 210 extracted software mentions. The scripts for implementing this stratified sampling procedure as well as all the analyses and figures discussed in the following sections can be found in https://github.com/caifand/cord19-sw-analysis.

Annotation

We developed a coding scheme to manually validate and annotate the sampled extraction results. Based on empirical descriptions of software citation practices and recommendations from advocacy, this coding scheme allows for capture of current practices for software citation and comparison to advocated-for best practices (*Smith, Katz & Niemeyer, 2016*; *Katz et al., 2019*; *Hong et al., 2021*). The full coding scheme contains 57 codes. Some codes were annotated based only on the content of the mention and its original article, while others, such as those regarding the access and archiving status of software, required annotators to conduct web searches and locate online presences of the mentioned software.

The first codes in our scheme validate the extracted mentions and their details. These codes require examination of the extracted mention content (sentences from original full-texts that mentioned software) and their accompanying bibliographical items. The original PDF publications were also examined to confirm whether the extracted results were consistent with the source article. For any problematic software extraction results, we manually corrected the extraction and annotated them accordingly. If it was not found in the corresponding PDF, we randomly sampled another mention from the same article, or moved to the next article in the sample frame and randomly sampled one of its mentions if the original article had only a single extracted mention. Throughout the sample annotation process, we found 5% of the automatically extracted software mentions are false positives (95% CI [0.03, 0.09]).

Figure 4 presents the remainder of our codes. We first replicated the coding scheme applied by *Howison & Bullard (2016)*, including codes about the in-text software mentions and their bibliographical entries, the functions of these software mentions, the access to the mentioned software, and whether the software citation aligns with a discoverable citation request (codes A1–B1, B3–C3, D1–D5, & E1). We then added codes E2–E14 to specify the format, contents, and location of citation requests. Next, we identify whether these codes meet the advocacy recommendations for software citation, particularly those discussed in *Smith, Katz & Niemeyer (2016)* and *Katz et al. (2019)*. This mapping from empirical descriptions to advocacy recommendations allows us to compare annotated practices with advocacy recommendations.

Here we explain how the codes are mapped from the scheme of empirical descriptions (*Howison & Bullard, 2016*) to specific advocacy recommendations, as the crosswalk in Fig. 4 shows. For example, if a software mention includes the name and version of the software, credits its creator(s), and provides a URL to facilitate the access (codes A1–A4), then the basic requirements of the Software Citation Principles (*Smith, Katz & Niemeyer, 2016*) are met. If the mention enables the access to the software, no matter it is open source or closed source, then it further conforms with later recommendations (codes D2–D5; particularly, *Katz et al. (2019)* have specified citation expectations for closed source software as the code

Themes	Code		Explanation	Howison & Bullard (2016)	Smith et al. (2016)	Katz et al. (2019)	FAIR4RS (2021)
Software mention	A1	Software name	The name of the mentioned software is mentioned	✓	✓	✓	
	A2	Version	Specific version number or release date is mentioned	✓	✓	✓	
	A3	Publisher	Publisher or creator of software is mentioned	✓	✓	✓	
	A4	URL	A web link is mentioned in text with software (a resolvable identifier is counted)	✓	✓	✓	
	A5	"Like instrument" mention	e.g., …were performed using the Prism 3.0 software (GraphPad Software, San Diego, CA, USA)	✓			
	A6	Software configuration details	e.g., operation environment; parameter settings	✓			
Software reference	B1	Cite to software publication	A "software publication" or a publication substantially discusses the technical aspects of the software	✓			
	B2	Cite to software	The software itself, instead of any publications, is referenced in bibliography		✓	✓	
	B3	Cite to domain publication	A publication that discusses domain science topics, including methods or models but not their software instantiation	✓			
	B4	Cite to user manual/guide	The documentation of the software is referenced	✓			
	B5	Cite to a project	A project name or website is referenced	✓			
Citation function	C1	Identifiable	The mention/citation enables identifying the software as a distinct identity (e.g., with a unique name; online information available from reliable sources)	✓			
	C2	Findable	A stable presence of the software in any relevant form (e.g., website; downloads; working repositories; documentation) can be found online	✓			
	C3	Findable version	A stable presence of the mentioned software version can be found online	✓	✓	✓	
	C4	Referenced with unique and persistent identifier	A possible form of unique and persistent identifier such as DOI, SWHID, ARK, Handle, NBN, or PURL is referenced as the software itself		✓	✓	
	C5	Referenced with a commit hash	A git commit hash is referenced for a snapshot of software			✓	
Software access	D1	No access	No findable information online about how to access the software	✓			
	D2	Proprietary	A license fee has to be paid for accessing the software	✓		✓	
	D3	Free access	Software can be accessed without a payment	✓	✓	✓	
	D4	Source code accessible	The source code of the software is available online	✓	✓	✓	
	D5	Permission to modify	The publisher has given permission to users for modifying the software (e.g., open source licensed or public domain software)	✓	✓	✓	
	D6	Open source	The software has a standard open source license		✓	✓	✓
Citation request	E1	Matches citation request	The software is mentioned in a way that matches the citation request by its publisher	✓			
	E2	Plain text citation request	The publisher has requested preferred citation in plain text style				
	E3	BibTex citation request	The publisher has provided a preferred citation in BibTex format				
	E4	Citation request in repository README	The publisher has requested preferred citation in the README of a working repository (e.g., on GitHub/GitLab)				
	E5	Citation request on web page	The publisher has requested preferred citation on a webpage (e.g., project website; on a software catalog page)				
	E6	CITATION file	A CITATION file is provided for requesting preferred citation				
	E7	CITATION.cff	A CITATION.cff file is provided for requesting preferred citation		✓	✓	
	E8	CodeMeta	A CodeMeta file is provided for requesting preferred citation		✓	✓	
	E9	Domain-specific citation metadata	Domain-specifc citation metadata is provided such as a R CITATION/DESCRIPTION file				✓
	E10	Request to cite software	The software publisher requests to cite the software itself		✓	✓	
	E11	Request to cite software publication	The software publisher requests to cite a software publication				
	E12	Request to cite domain publication	The software publisher requests to cite a domain science publication				
	E13	Request to cite a project	The software publisher requests to cite a project rather than a software product				
	E14	Request to cite other research product	The software publisher requests to cite other non-software products such as dataset				
Standard compliance	F1	Archived	The software has at least one version archived in a archival repository (e.g., Zenodo; Figshare; institutional repository)			✓	
	F2	Unique and persistent identifier	The software itself has a unique and persistent identifier (e.g., DOI, Handle, ARK, NBN, PURL, SWHID)		✓	✓	✓
	F3	Metadata available	The software has publicly accessible metadata, including both citation metadata and package metadata		✓	✓	✓

Figure 4 **A crosswalk between the software extraction coding scheme and advocacy recommendations.**
Full-size DOI: 10.7717/peerjcs.1022/fig-4

D2 concerns). If a mentioned piece of software has an available citation request online for the mentioned software and it requests to cite the software artifact rather than any other complementary artifact (such as a publication or user manual) and if specific formats of citation metadata (*e.g.*, CITATION.cff, CodeMeta) are adopted, these practices also comply with the advocacy recommendations (codes E7, E8, E10). In cases where a citation request is declared by domain-specific metadata, such as by an R DESCRIPTION/CITATION file,

it does not conform with those software citation advocacy recommendations, but meets one of the FAIR Principles for Research Software (FAIR4RS) that the software metadata should be *findable*. While the FAIR4RS Principles are not primarily advocated for software citation implementation, we included them for reference.

To better examine the extent to which existing citation practices align with advocacy recommendations, we added codes that are primarily derived from advocacy recommendations (codes B2, C4–C5, D6, F1–F3). These codes concern whether the software artifact itself is referenced in the bibliography with a unique and/or persistent identifier and whether the software artifact is archived, registered with an persistent identifier, or has available metadata online. These emphases from current advocacy recommendations ensure that the software artifact can be referenced like traditional research products such as scholarly publications. In particular, valid form of metadata available (codes E7–E9, F2 & F3) and/or a unique and persistent identifier make the software in accordance with the FAIR4RS "Findable" principle (codes F1, F2, and/or F3). If the software code follows standard open source (code D6), then it is "reusable" in the sense of the FAIR4RS definition (specifically, see FAIR4RS principle R1.1 in *Hong et al., 2021*). Given our emphasis on software citation, we considered only the aspects of FAIR4RS related to software citation.

To confirm the inter-annotator reliability of the coding scheme, two authors initially annotated a sub-sample (10% of the full sample). They achieved 93.3% percentage agreement across all the codes in the full coding scheme; discussions resolved the remaining disagreements. Later, a third author joined as an annotator using the validated coding scheme. Questions emerging from the annotation process were then discussed among annotators to reach consensus and refine the coding scheme accordingly. Finally, after the annotation was finished, one author wrote a script to validate the logical constraints and consistency between coding results, and re-annotated when violations of logical constraints were identified.

RESULTS

As our annotation scheme is extended from previous empirical descriptions of software citation practices, we are able to compare our results with earlier findings from *Howison & Bullard (2016)* to see if software citation practices have changed. The annotation scheme also allows us to compare existing practices with advocacy recommendations and understand whether they converge or deviate. In this section, we report the annotation results with these considerations. We first describe the forms that software mentions take in our sample and the functions of scholarly citation they are able to realize. Next, we discuss the extent to which the mentioned software is citable as the advocacy has recommended by examining their metadata availability, archiving status, and persistent identification. Finally, we present findings about how software citation has been requested by software creators and publishers. Our annotation was conducted in August 2021 and all the annotations about software online presences correspond to results of web searches conducted then. We report all the proportions of coded categories with 95% confidence interval. These results were calculated using the `prop.test` function in the base R package stats (*R Core Team, 2019*).

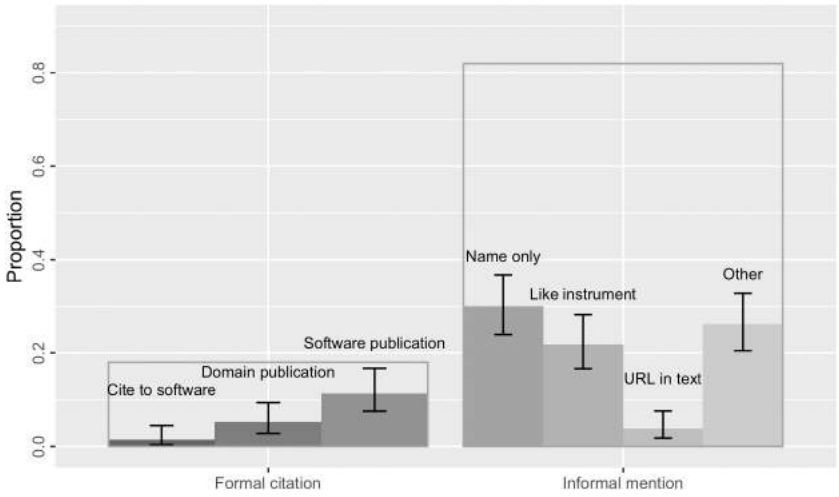


Figure 5 18% of the software mentions in the sample were formal citations (the transparent bar on the left) of either the software itself, a domain publication, or a software publication. The remaining 82% were informal mentions without bibliographical references, including those mentioning software name only, following a "like instrument" style, providing a URL in text, or other forms of casual mentions that might refer to a version and/or software creator. Error bars show the 95% confidence interval estimates.
Full-size DOI: 10.7717/peerjcs.1022/fig-5

How software is mentioned

Forms of software mention

Overall, 82% of the software mentions in our sample ($N = 172$; 95% [0.76, 0.87]) were informal without a bibliographical reference (Fig. 5). Most of these informal mentions only gave the name of the software; sometimes they additionally referred to a version, software creator or publisher, and/or a URL. 30% of the mentions ($N = 63$; [0.24, 0.37]) only referred to the software by its name (Fig. 5). Because our annotation procedures rely on the results of the software mention recognizer, it is possible that we have missed mentions that provided no name (*e.g.*, "using a program we wrote"). However, we do not expect these would have been substantial, given that *Howison & Bullard (2016)* only found 1% ([0, 0.04]) unnamed software mentions in their corpus.

Consistent with findings from *Howison & Bullard (2016)*, we found software was sometimes mentioned "like instrument" (22% of the sample, $N = 46$, [0.17, 0.28]) in that it is similar to how researchers conventionally reference scientific instruments provided by vendors or manufacturers when authoring academic publications. This kind of mentions specify the software name, the name of its provider, and often the geographical location of the provider (*e.g.*, "STATA, StataCorp, College Station, TX"). 4% of the mentions ($N = 9$; [0.02, 0.08]) in the sample provided a URL in text for software access. These results are indistinguishable from the findings in *Howison & Bullard (2016)* that 19% ([0.14, 0.24]) of the software mentions identified in the biology literature sample were in the "like instrument" style and 5% ([0.03, 0.08]) gave a URL.

In contrast to findings from *Howison & Bullard (2016)*, we did not find any software mentions citing a software manual or its project. The majority of formal citations still cited a publication—either a domain science publication (5%, $N = 11$, [0.02, 0.09]), or, more

commonly, a software publication (11%, $N = 24$, [0.08, 0.17]). In Howison & Bullard's sample, even more software mentions cited a publication (37%; [0.31, 0.43]).

Software Citation Principles (*Smith, Katz & Niemeyer, 2016*) suggest that the software artifact itself should be cited as a first-class research product; any relevant publications should be cited as companions. In this case, we found three formal citations of the software artifact in bibliography (1.4%; [0.004, 0.05]), which was not found in *Howison & Bullard (2016)*, but none of them included an identifier such as a persistent DOI or a commit hash (*Katz et al., 2019*). Thus, none of these software mentions or citations met the Software Citation Principles' goal of *unique identification* or *persistence*. The identification of the mentioned software, especially when a specific version is involved, mainly relies on whether there is a URL referenced in text, otherwise the academic readers would need to make use of any clue revealed by the software mention to look for the software.

Functions of software mentions

Traditional scholarly citation allows for the identification, access, and subsequently verifying and building upon the cited work (*Howison & Bullard, 2016*). Software citation advocacy seeks to enable these functions by recommending best practices for referencing software in scholarly work (*Katz et al., 2021*). *Howison & Bullard (2016)* found that informal software mentions could still function to some extent in scholarly communications. We assessed and annotated the enabled functions of mentions in our sample in accordance.

A software mention was annotated as "identifiable" if the information given distinguished the software as a distinct entity. It was then annotated as "findable" when the mention facilitated the online search and discovery of the software. 96% ([0.92, 0.98]) of the software mentions were both identifiable and findable, enabling the annotator to discover a distinct piece of software *via* web search (Fig. 6). These results suggest an improvement on findable software from Howison & Bullard's results: they found 93% ([0.88, 0.96]) of software mentions supported the successful identification of software but fewer (86%; [0.80, 0.90]) enabled online discovery. Overall, it is positive that we were able to identify and find almost all of the software even though 82% of all the identified mentions were informal. This finding implies that most software has online presence(s).

We saw that 46% ([0.39, 0.53]) of the software mentions specified a version and 43% ([0.36, 0.50]) of the software mentions had a findable version online, an increase over Howison & Bullard's findings (28%, [0.22, 0.35], and 5%, [0.03, 0.10], respectively). 48% ([0.41, 0.56]) of the informal software mentions and 35% ([0.21, 0.53]) of the formal citations identified specific versions (Fig. 7). 46% ([0.38, 0.53]) of the informal mentions led to findable versions online and so did the 30% ([0.16, 0.47]) of the formal citations. The annotation results also suggest that 78% ([0.61, 0.90]) of the formal citations and 35% ([0.28, 0.42]) of the informal mentions identified authorship and thus were able to give credit. We interpret a likely reason for the difference between existing formal citations and informal mentions in their strength to give credit and to identify versions: 92% ([0.77, 0.98]) of the formal software citations reference a publication, indicating clear authorship; but a publication, even a "software paper", is not often version specific. Current software

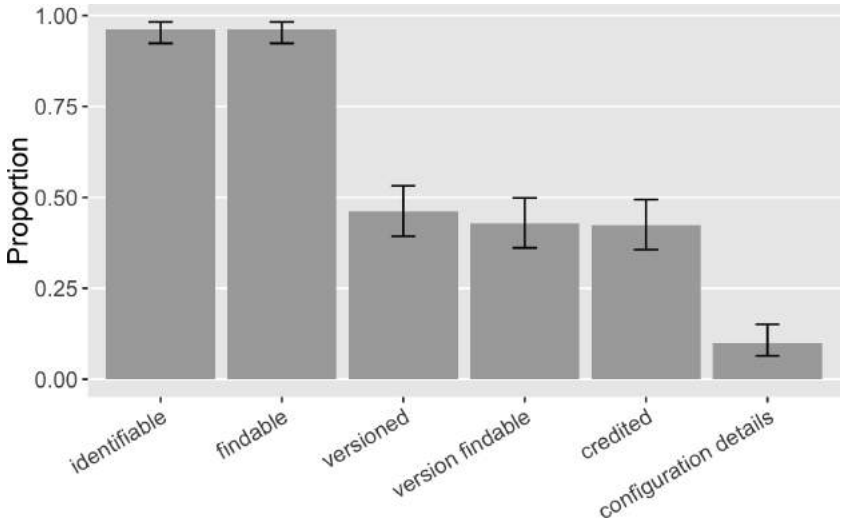


Figure 6 **Functions of software mentions in our sample (error bars show 95% CI): Do software mentions enable the identification and discovery of software and its versions? Do software mentions credit its contributors? Do software mentions provide further configuration detail that facilitates reuse?**
Full-size DOI: 10.7717/peerjcs.1022/fig-6

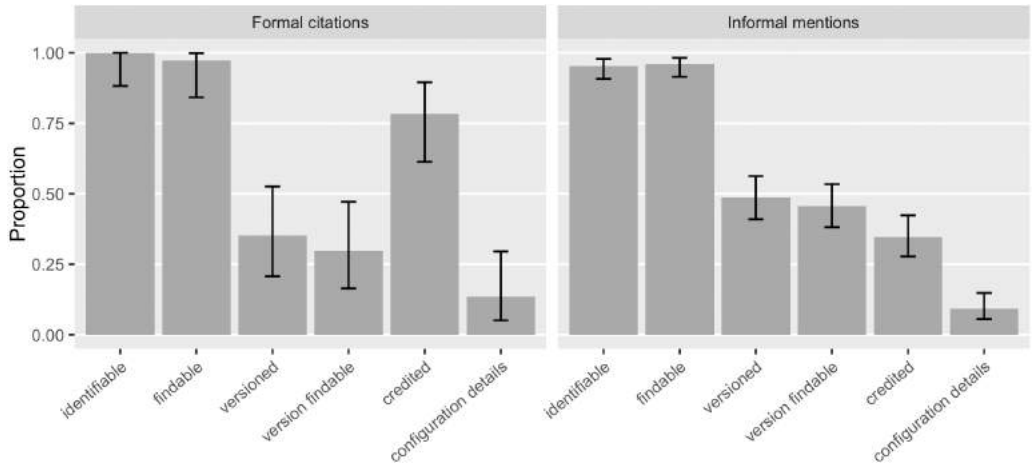


Figure 7 **How do existing formal citations and informal mentions of software function? In our sample, informal mentions tended to provide more specific versions; and formal citations credited better.** Most formal citations in our sample referenced a publication rather than the software artifact (Error bars show 95% CI).
Full-size DOI: 10.7717/peerjcs.1022/fig-7

citation advocacy has also recognized this: the credit given by publication is often one-off and static, not sufficient to account for the dynamic authorship of the actual software work across versions (*Katz et al., 2019*; *Katz & Smith, 2015*).

Proper crediting has been a strong motivation for software citation advocacy. In our sample, 42% ([0.36, 0.49]) of the software mentions recognized the creators or publishers of the software. In contrast, *Howison & Bullard (2016)* found around 77% ([0.70, 0.83]) of the mentions recognized creators or publishers. This is possibly driven by the 37%

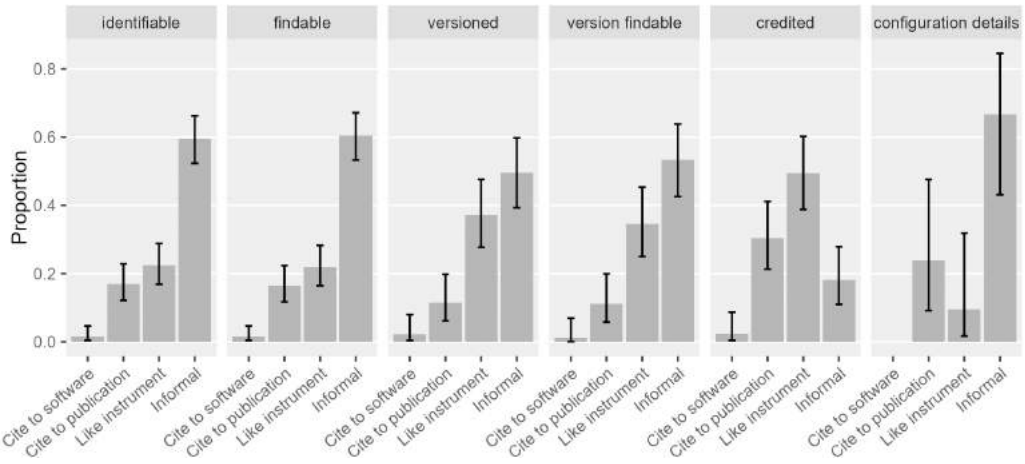


Figure 8 **Types of software mentions that serve different functions.** Software mentions that give credit were mostly driven by "like instrument" mentions and formal citations to a publication (Error bars show 95% CI).

Full-size DOI: 10.7717/peerjcs.1022/fig-8

of the software mentions in their sample that cited a publication while only 17% in ours did. When looking more closely (Fig. 8), crediting mentions in our sample were mostly "like instrument" mentions (49% of all the crediting mentions; [0.39, 0.60]), and secondarily, formal citation of articles (30% of all the crediting mentions; [0.21, 0.41]). As we will discuss in the later section, "like instrument" mentions mostly credited proprietary software publishers, who are in less need of scholarly credit. In cases of software citation, information about the proprietary software publisher probably provides more accountability for the software involved and thus contributes to the integrity of scientific communication.

Finally, 10% ([0.06, 0.15]) of the mentions gave some additional detail about the actual configuration of mentioned software, usually specifying an operation environment or parameter settings. Such detail could facilitate the verification of the scientific method employed and the reuse of mentioned software.

Another key condition of reuse is whether the mentioned software is accessible and retrievable. Facilitating access is one advocated function of software citation (*Smith, Katz & Niemeyer, 2016*) as well as the conventional concern of scholarly citation (*Howison & Bullard, 2016*). For the 155 distinct pieces of software mentioned in our sample, 97% ([0.92, 0.99]) was accessible online, 68% ([0.60, 0.75]) of the mentioned software had free access, 47% ([0.39, 0.55]) had source code available, and 43% ([0.35, 0.51]) had both source code available and permission to modify it, such as an open source license, or a statement of waived copyrights (as the "source code modifiable" category in Fig. 9). In general, mentioned software in this sample was more accessible and actionable than that in *Howison & Bullard (2016)*, where 79% ([0.71, 0.85]) of the mentioned software was accessible, 47% ([0.38, 0.56]) had free access, 32% ([0.24, 0.40]) had source code available, and 20% ([0.14, 0.27]) had modifiable source code.

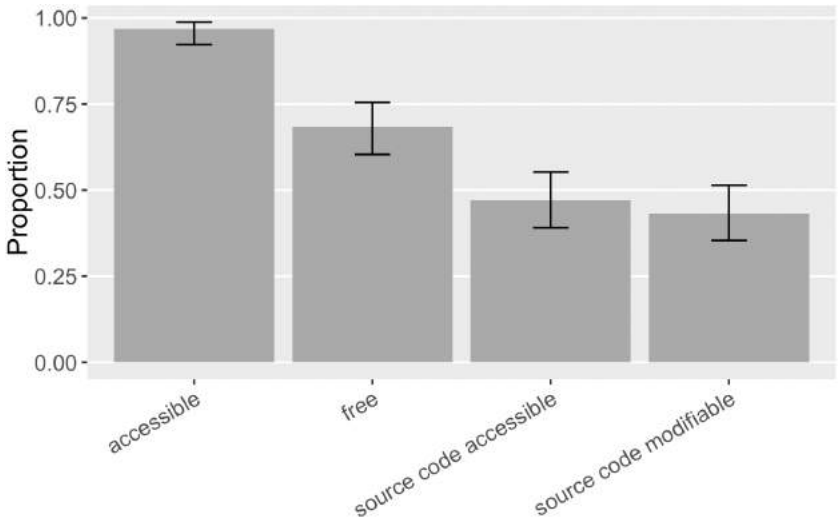


Figure 9 Accessibility of the software mentioned (error bars show 95% CI). Accessible software includes software with restricted or paid access. Free access software includes those with open source code available, or those in the form of executable or web services. Modifiable source code means the permission to modify the source code is formally granted.

Full-size DOI: 10.7717/peerjcs.1022/fig-9

Software metadata, archiving, and identification

Advocates' recommendations, such as *Katz et al. (2019)*, recognize that sufficient citation metadata, software archiving, and unique and persistent identifiers are needed to cite the software artifact in a human- and machine-actionable manner (*Smith, Katz & Niemeyer, 2016*; *Katz et al., 2019*). These practices are essential for software and its citation in scholarly literature to become as visible as traditional academic publications, such as being identifiable and indexable by discovery tools. We therefore examined whether the 155 distinct pieces of software mentioned in our sample meet these advocacy recommendations.

Metadata availability was assessed by looking for publicly accessible metadata, including those primarily focused on software citation (*i.e.,* CITATION.cff (*Druskat et al., 2021*) and CodeMeta (*Jones et al., 2017*)) and metadata stored in public information registries[3] such as bio.tools (*Ison et al., 2016*). We also annotated the availability of language-specific software metadata such as R DESCRIPTION/CITATION files, Python `setup.py` files, and Java Maven `pom.xml` files, *etc.*, considering that these are interoperable with CodeMeta through crosswalks.

We searched for archival copies of mentioned software within Zenodo (*European Organization For Nuclear Research and OpenAIRE, 2013*), Figshare (https://figshare.com), and Software Heritage (*Cosmo & Zacchiroli, 2017*) with their within-site search feature. We considered searching institutional or domain-specific repositories but it is infeasible to create an exhausted list of repositories for manual search. We also experimented with using search engines like Google; it was neither effective to identify software archived in specific archival repositories.

We assessed the use of unique and persistent identifiers by searching for DOIs, ARKs (Archival Resource Keys), PURLs (Persistent URLs), and NBNs (National Bibliography

[3] We did not annotate metadata only associated with a minted DOI as the unique and persistent identifier category covers this possibility. Hence, we focus on other kinds of interoperable metadata.

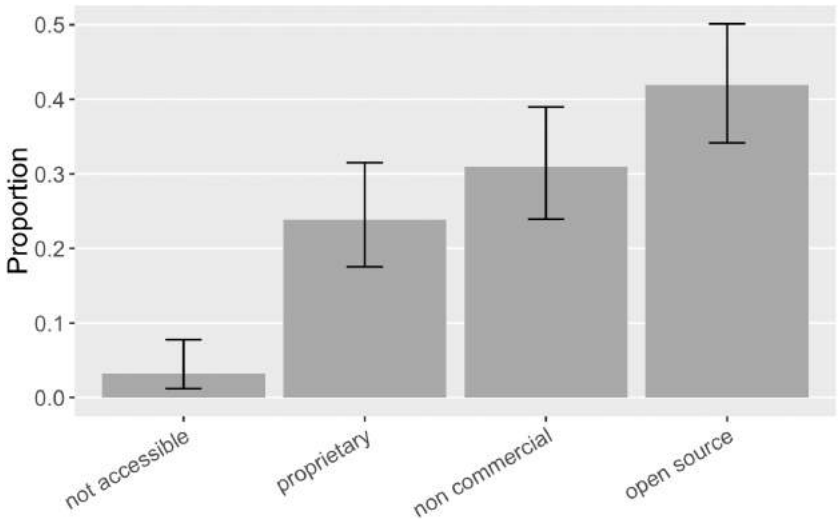


Figure 10 The property rights of the software mentioned (error bars show 95% CI). Five software mentions did not have an accessible record online, and their licensing status consequently could not be identified.

Full-size DOI: 10.7717/peerjcs.1022/fig-10

Numbers). We chose this list to focus on globally used and interoperable identifiers. We considered other identifiers, including RRID, ASCL ID, and swMATH ID, some of which are very useful in specific domains; but limited annotation labor turned our focus on those globally used and indexed across systems of digital objects. Additionally, we did not include SWHID (Software Heritage ID) (*Software Heritage Development Documentation, 2021*) because those are automatically generated once it is archived in Software Heritage, already accounted for in our annotation about archiving.

As the advocacy recommendation efforts recognize, unique and persistent identification, metadata accessibility, and archiving mechanisms can vary a lot between open source and closed source software (*Katz et al., 2019*). We thus examine the software in our sample with different property rights respectively. Particularly, we annotated closed source software with some kind of paywall as "proprietary", software with a standard open source license as "open source", and free access software without standard open source license as "non-commercial" (*e.g.*, public domain software or software with source code available but not following standard open source practices). As Fig. 10 shows, in our sample, five cases were not accessible and thus we cannot identify their licensing status; 24% ([0.18, 0.32]) of the mentioned software was proprietary while more was open source (42% of the mentioned software; [0.34, 0.50]); in-between is non-commercial software (31% of the mentioned software; [0.24, 0.39]). In Howison & Bullard's sample, the corresponding proportion of proprietary, non-commercial, and open source software were 32% ([0.24, 0.40]), 27% ([0.21, 0.36]), and 20% ([0.14, 0.27]). Therefore, we observe a larger portion of open source software mentioned in this sample.

Overall, 30% ([0.23, 0.38]) of the mentioned software had at least one archived copy within Zenodo, Figshare, or Software Heritage (Fig. 11). While archiving in these

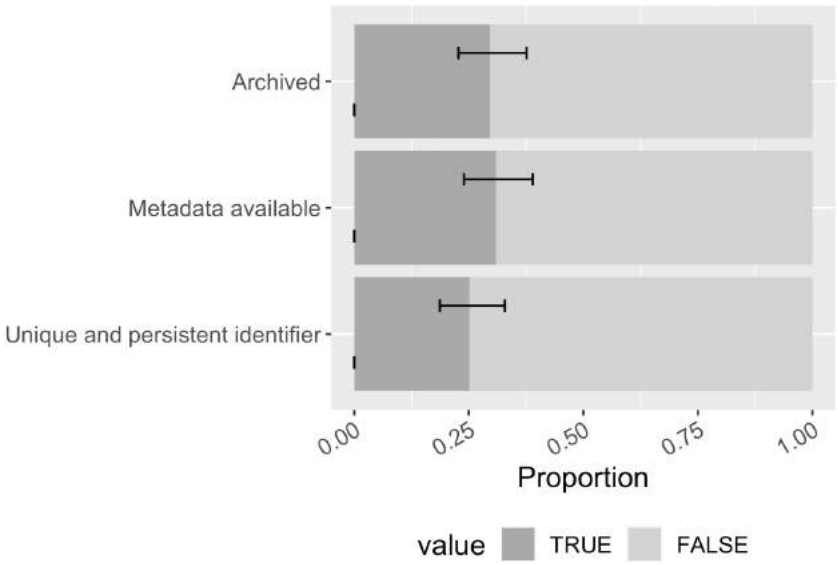


Figure 11 **Software archiving status, metadata availability, and persistent identification status in the sample (error bars show 95% CI).**

Full-size DOI: 10.7717/peerjcs.1022/fig-11

repositories generates interoperable metadata for the archived item, slightly more software in the sample had publicly available metadata (31% of the mentioned software; [0.24, 0.39]), implying some software had metadata created elsewhere. 25% ([0.19, 0.33]) of the mentioned software had at least one unique and persistent identifier found among possibilities of DOI, ARK, PURL, and NBN; the most found identifier is DOI.

When examining across software with different kinds of property rights, we found that 68% ([0.55, 0.78]) of the open source software in the sample was archived in one of those aforementioned repositories (Fig. 12). During annotation, we noticed that this is largely driven by those software projects with a GitHub repository archived in the Software Heritage. Some non-commercial software in our sample with a source code repository on GitHub was also found archived by the Software Heritage. Only 4% of the non-commercial software was archived while no proprietary software in the sample was found archived. Although it is in theory possible to archive proprietary software in a closed source archive and only expose the unique identifier and metadata for discoverable records.

We have found publicly available metadata for 16% ([0.07, 0.33]) of the proprietary software in our sample, mostly stored in a public registry (*e.g.*, bio.tools). 17% ([0.08, 0.31]) of the non-commercial software had publicly accessible metadata, so as 52% ([0.40, 0.65]) of the open source software. But these available metadata are not oriented to citation. We found no CITATION.cff or CodeMeta in our sample.

Open source software was more likely than non-commercial software to have a unique and persistent identifier (58%, [0.46, 0.70] *vs.* 2%, [0.001, 0.12]); proprietary software had none at all.

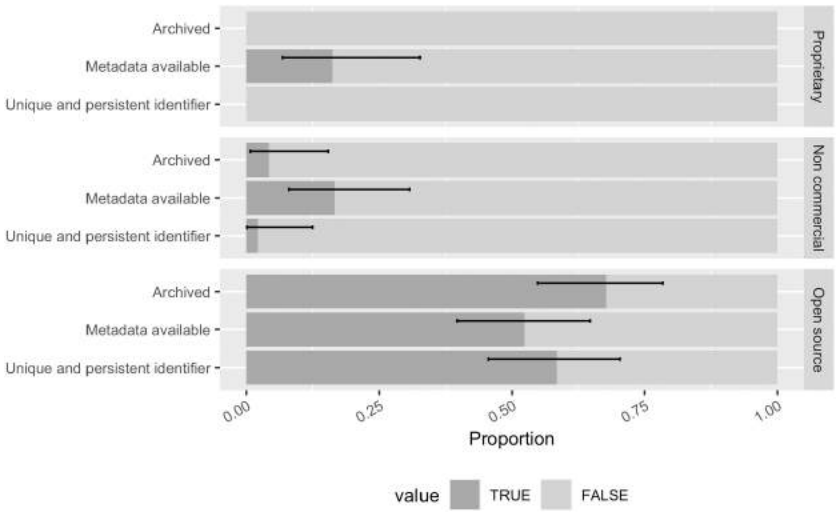


Figure 12 The archiving, metadata, and persistent identification status of software with different property rights in the sample (error bars show 95% CI). More than half of the open source software was archived and had available metadata and a persistent identifier. Proprietary software in the sample did not have any identified archival copies or persistent identifiers.

Full-size DOI: 10.7717/peerjcs.1022/fig-12

Taken together, slightly more than half of the open source software in our sample met advocacy recommendations for citation. Closed source software, including non-commercial software without open source code available and proprietary software, indeed rarely have a unique and persistent identifier neither being frequently archived. It is not surprising given that archiving a piece of software is the primary way to obtain a unique and persistent identifier. The lack of the persistent identifier perhaps chiefly reduces the indexing potential of the closed source software. It is still valuable to reference proprietary software used for research in manuscripts especially when its routines constitute part of the research procedures.

How citation is requested

Past research has shown that software creators and publishers seek scholarly credit by having their software cited (*Howison & Herbsleb, 2011*). *Howison & Bullard (2016)* found 18% ([0.13, 0.30]) of the mentioned software in their sample made a specific request for their software work to be cited, usually in the online presences of the software such as on a project web page. *Bouquin et al. (2020)* also looked for the preferred citations requested by software projects in their sample, motivated by examining the quality of these sources of citation information. Findings by these studies suggest that public citation requests do not necessarily prioritize the software artifact itself. Instead, software authors may request citations reference a software publication because those publications are more immediately compatible with the current system of scholarly citation, reputation, and impact.

We are interested in understanding how common citation requests are, how software projects make specific citation requests, to what extent they orient researchers' citation behavior, and whether they conform to the best practices recommended by the software

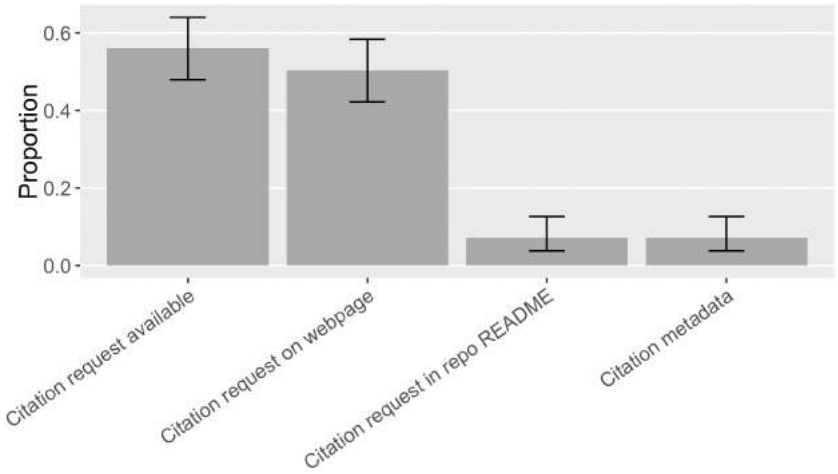


Figure 13 Locations of software citation request in the sample (error bars show 95% CI). Citation metadata includes formats like CITATION file and domain-specific citation metadata such as R CITATION/DESCRIPTION. No CITATION.cff or CodeMeta was found.
Full-size DOI: 10.7717/peerjcs.1022/fig-13

citation advocacy. We therefore annotated the format and location of citation requests and what citation target was requested by searching for specific software's citation request and looking through the online presence(s) of the software in our sample. Overall, we found 87 out of the 155 pieces of software (56%, [0.48, 0.64]) had at least one citation request findable online (Fig. 13); but only 13% ([0.09, 0.19]) of the sampled mentions followed these citation requests. *Howison & Bullard (2016)*, in comparison, found 18% ([0.13, 0.30]) of the software in their sample made citation requests and 7%([0.04, 0.11]) of the software mentions followed these citation requests.

While we found an increase in citation requests and citations that follow them over Howison and Bullard's findings, citations that follow recommendations were still fairly limited, varying by the type of software in question. 46% ([0.30, 0.63]) of the proprietary software in our sample had at least one citation request while 2% ([0.004, 0.09]) of the proprietary software mentions matched the requested citation. Half of the non-commercial software (50%, [0.36, 0.64]) in the sample requested software citation; 13% ([0.06, 0,.26]) of the non-commercial software mentions matched the actual citation request. 70% ([0.58, 0.81]) of the open source software made citation request and 26% ([0.17, 0.38]) of their mentions matched the request (see Fig. 14). In general, open source software projects made more citation requests, and their requests were more followed.

Locations of citation requests

Half of the software in the sample (50%, [0.42, 0.58]) had a preferred citation specified on a web page, mostly located on a software project website. Occasionally, we found a software catalog online suggesting a citation for its software entries (*e.g.*, Bioconductor (https://www.bioconductor.org/) and ASCL (*Nemiroff & Wallin, 1999*) do so). 7% ([0.04, 0.13]) of the software requested software citation in the README file of a source code repository. Another 7% ([0.04, 0.13]) had a metadata file available

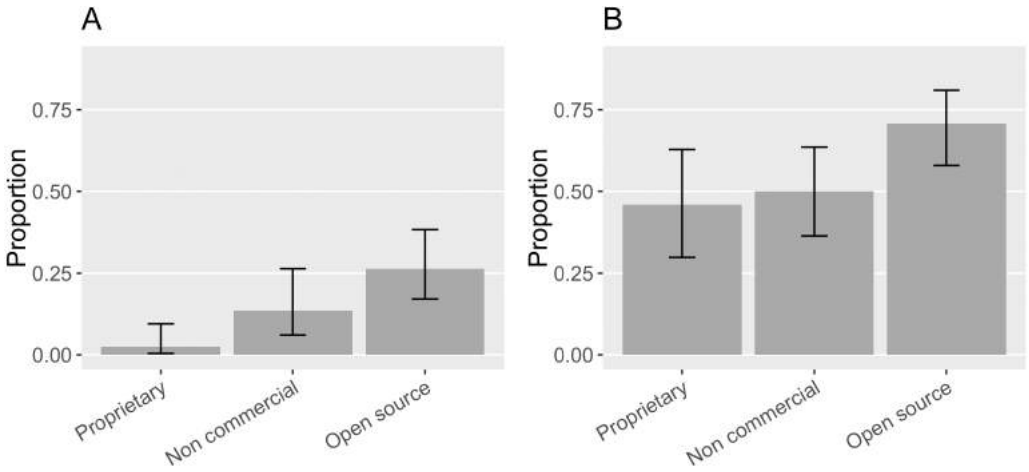


Figure 14 **(A) Proportion of the mentioned software with different property rights in the sample that had citation request(s). (B) Proportion of the mentions of software with different property rights that matched the citation request (Error bars show 95% CI).**

Full-size DOI: 10.7717/peerjcs.1022/fig-14

providing needed information for citation, especially the domain-specific format R CITATION/DESCRIPTION was found to specify a citation request (Fig. 13).

Formats of citation requests

Software projects in our sample use different formats to make citation requests. It was most common (54%, [0.45, 0.62]) to give a suggested citation in plain text format so that users can copy and paste it into their manuscript (Fig. 15). The next most popular format was a BibTeX formatted citation entry that can be used in the LaTeX document preparation system; but, at 7% ([0.04, 0.13]) of our sample, even this second-most-poplar citation request format was rare. Domain-specific citation metadata format, exclusively the R CITATION/DESCRIPTION file, was used by 6% ([0.03, 0.11]) of the cases in our sample; considering that it is a common expectation of the R language community, it may be a convenient choice for software developers to make a citation request. Finally, two pieces of software in our sample (1%, [0.002, 0.05]) used a CITATION file. These were initially embraced by advocates for crediting research software work (*Wilson, 2013*); because CITATION files are not machine-readable, advocates now prefer structured citation metadata like Citation File Format (CFF) and CodeMeta (*Katz et al., 2019*). We did not find any CITATION.cff or CodeMeta.json files, indicating these newer approaches have yet to replace older ones.

Objects of citation requests

If a request was made, it was the most common to ask that users cite a software publication (32%, [0.25, 0.4]). 20% ([0.14, 0.27]) of the software in our sample requested to cite the software artifact itself, in accordance with advocates' recommendations (*Smith, Katz & Niemeyer, 2016*). 12% ([0.07, 0.18]) of the software in the sample requested that a domain science publication be cited. Two software projects requested the project itself be cited. One

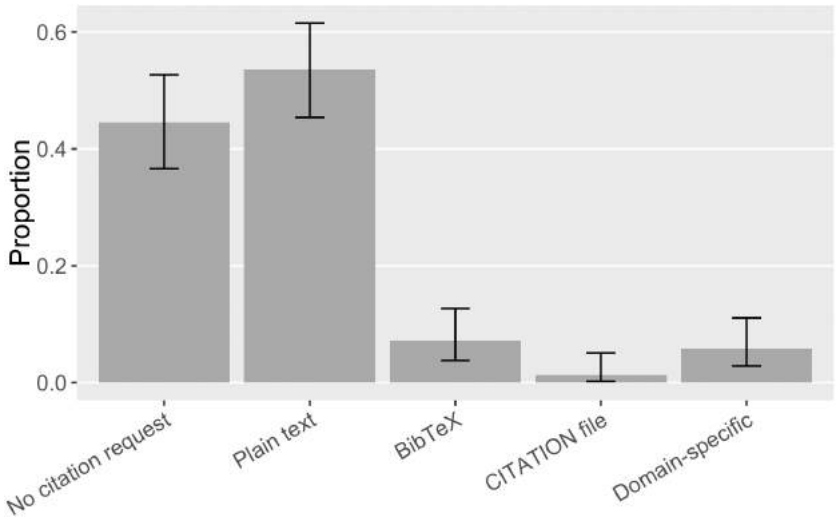


Figure 15 **Formats of software citation requests in the sample (error bars show 95% CI).**
Full-size DOI: 10.7717/peerjcs.1022/fig-15

project requested that a dataset be cited; in this specific case, the software is the byproduct of the dataset.

It is worthwhile to look closely at these citation requests with respect to different software property rights because citation preferences vary (Fig. 16A). Non-commercial and open source software seemed to prefer a publication citation the most: 38% ([0.24, 0.53]) of the non-commercial software in the sample requested to cite a software publication while 46% ([0.34, 0.59]) of the open source software requested so; 17% ([0.08, 0.30]) of the non-commercial software and 15% ([0.08, 0.27]) of the open source software requested that users cite a domain science publication. 4% ([0.007, 0.15]) of the non-commercial software and 20% ([0.11, 0.32]) of the open source software requested that the software itself be the target of the citation.

In contrast, only one proprietary software project in our sample requested that users cite a software publication (2%, [0.001, 0.16]); 43% ([0.28, 0.60]) of the proprietary software requested that authors cite the software itself; and the remaining 46% ([0.30, 0.63]) of the proprietary software in the sample did not have a citation request available. This is likely because proprietary software publishers have more incentives to promote their software product but have less motive to engage in publishing about the science relevant to their software. Meanwhile, for open source and non-commercial software projects, citing their publication can more effectively demonstrate their scientific impact in the way most relevant to those evaluating their careers.

We accordingly examine how the software with different kinds of property rights in our sample is actually mentioned in publications (Fig. 16B). None of the mentions of proprietary software in our sample came with any bibliographic references. Most commonly they were referenced in the "like instrument" style (52%, [0.41, 0.63]), following the practices of referring to scientific instruments and their vendors. Yet, both informal and "like instrument" mentions ensure that the actual software artifact is referenced in publications.

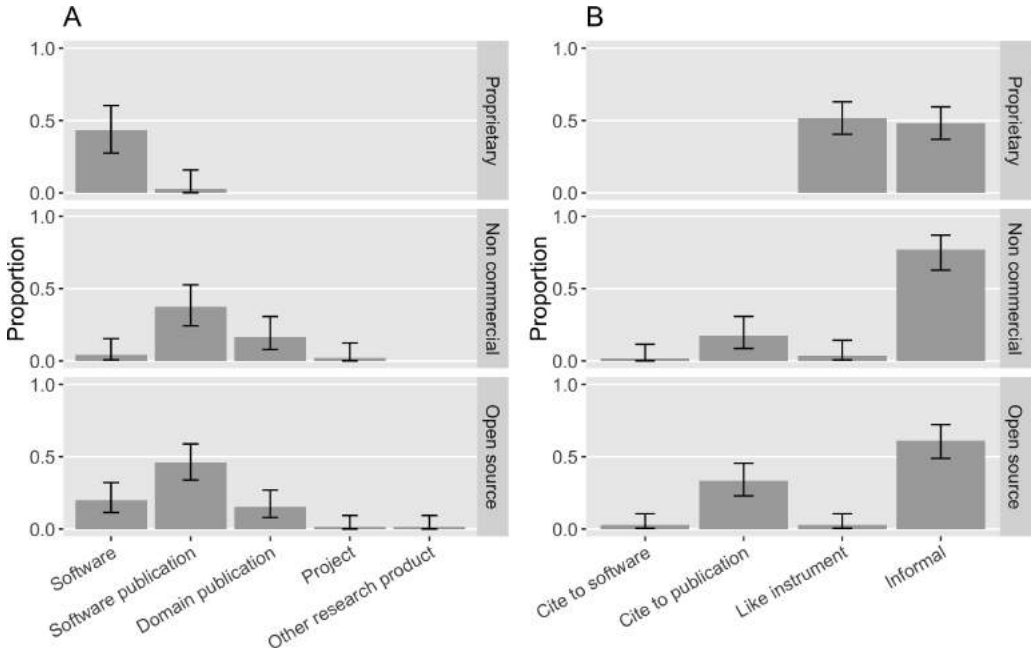


Figure 16 **(A) What software projects with different property rights request to cite in the sample. (B) How software with different property rights is mentioned in the sample (Error bars show 95% CI).**
Full-size DOI: 10.7717/peerjcs.1022/fig-16

17% ([0.09, 0.30]) of the non-commercial software mentions and 33% ([0.23, 0.46]) of the open source software mentions cited a publication; both were rarely mentioned like instruments in our sample: Only 4% ([0.006, 0.14]) of non-commercial software mentions and 3% ([0.004, 0.10]) of open source software mentions were instrument-like. Despite frequent requests to cite publications, 78% ([0.63, 0.87]) of the non-commercial software mentions and 61% ([0.49, 0.72]) of the open source software mentions were informal. Although 20% ([0.11, 0.32]) of the open source software in our sample requested that users cite the software artifact, only 3% of the open source mentions did so.

Comparison with prior findings

Finally, in Fig. 17 (on page 24), our annotation results are summarized in comparison with results from *Howison & Bullard (2016)*. Given that the two sets of findings come from two independent samples, we conducted a two-sample significance test to compare the proportions calculated from annotations, using the `prop.test` function (*Newcombe, 1998*) in base R. When the difference between the proportion values across the two samples is tested as significant with $p < .05$, we have more statistical support to conclude a notable change. These statistically significant changes are highlighted in Fig. 17. The *In-text URL* and *Cite to software* categories have too few positive results (both under ten) to be an adequate sample for such significance testing.

Overall, we found more informal mentions and less citations to publications for referencing software in our sample. The mentioned software were more findable and more frequently referenced with a specific version, and these versions were also more accessible.

Found proportions in the sample	Prior results in Howison & Bullard (2016) (N=286)	Current results (N=210)
Types of software mentions		
Informal mentions without bibliographical references	56% (50%-62%)	↗ 82% (76%-87%)
In-text name only	31% (26%-37%)	30% (24%-37%)
Instrument-like	19% (14%-24%)	22% (17%-28%)
In-text URL	5% (3%-8%)	4% (2%-8%)
Cite to publication	37% (31%-43%)	↘ 17% (12%-23%)
Cite to software	0%	1% (0.4%-4%)
Functions of software mentions		
Software identifiable	93% (88%-96%)	96% (92%-98%)
Software findable	86% (80%-90%)	↗ 96% (92%-98%)
Versioned software mention	28% (22%-35%)	↗ 46% (39%-53%)
Version findable	5% (3%-10%)	↗ 43% (36%-50%)
Credited	77% (70%-83%)	↘ 42% (36%-49%)
Configuration details provided	Not provided	10% (6%-15%)
Requested citation		
Citation matches request	7% (4%-11%)	↗ 13% (9%-19%)
Found proportions in the sample	Software projects in Howison & Bulard (2016) (N=146)	Software projects in current sample (N=155)
Property rights of mentioned software		
Not accessible	21% (15%-29%)	↘ 3% (1%-8%)
Proprietary	32% (24%-40%)	24% (18%-32%)
Non-commercial	27% (21%-36%)	31% (24%-39%)
Open source licensed	20% (14%-27%)	↗ 42% (34%-50%)
Accessibility of mentioned software		
Free access	47% (39%-56%)	↗ 68% (60%-75%)
Source code available	32% (24%-40%)	↗ 47% (39%-55%)
Source code modifiable	20% (14%-27%)	↗ 43% (35%-51%)
Citation request		
Citation requested	18% (13%-30%)	↗ 56% (48%-64%)

Figure 17 Comparing current results with previous findings by *Howison & Bullard (2016)*. Statistically significant differences are highlighted in colors, with orange denoting an increase and blue denoting a decrease, correspondingly.

Full-size DOI: 10.7717/peerjcs.1022/fig-17

However, proper crediting of software contributors did not seem to have improved. We see that mentioned software were more accessible; more mentioned software followed standard open source practices and provided modifiable source code. An increasing amount of software requested a preferred citation, but the actual mentions found did not always follow these requests.

DISCUSSION

In this section, we assess the limitations of our findings, discuss explanations for why we found the changes that we found, and discuss challenges for advocacy we perceived through our study.

Limitations

In this paper, we compare our findings with those from previous publications, but our ability to compare and ascribe differences to changes in practices over time is subject to limitations. The sample in *Howison & Bullard (2016)* was taken from articles published between 2001 and 2010 in journals indexed in biology-related subject headings by the Web of Science; the sample in this study was constructed from papers published since 2016 from the CORD-19 corpus, covering perhaps more diverse venues that include many biology journals but only including content topically relevant to coronaviruses. Further, Howison and Bullard randomly selected articles assessing all mentions within those articles, whereas this study first used machine learning to identify mentions, then randomly selected mentions for annotation. Both approaches work and shed light on our understanding of how software is cited and citable in scholarly communication.

We also compare our findings with recommendations from software citation advocates. We report these findings as a baseline for future comparison, rather than an assessment of advocacy success, because it is likely that the time frame of publications chosen is insufficient for assessing the impact of advocacy. While we chose publication dates that came after advocacy recommendations, publication timelines can be long and we do not know when these articles were drafted; some may have been drafted prior to the publication of advocacy guidelines. Second, we do not know whether the authors, editors, or venues were exposed to advocacy at all; publishing and promoting articles and principles about software citation does not mean they are widely read. Our results should be understood as relatively contemporaneous with the emergence of new software citation recommendations.

Finally, our findings about citation requests only reflect the moment of data collection and annotation. These online records are subject to constant change; this means that the citation request may have not been present when a publication that cited software was authored. Nonetheless, earlier findings were subject to this same limitation so we think the comparison between studies is useful.

Possible explanations of changes observed

We found that mentioned software were more available and accessible, included mention of specific versions, and their source code was more frequently available, particularly as standard open source. We reason that the increase of open source and accessible source

code has followed the overall rise of open source and particularly the availability of hosted platforms for software development such as GitHub. Additional pressure may also have come from funders, which increasingly signal support for open source code.

We also reason that the observed increase in the mention of software version numbers could result from both the developers' practice and the paper authors' practice. On the development side, more software has been produced and shared for research use. Software production in general has also increasingly followed the practices of versioning, including semantic versioning (*e.g.*, *Decan & Mens, 2021*) as a well established open source practice that is reinforced by code hosting platforms (*e.g.*, GitHub "releases" and git tags). On the paper author side, it seems likely that the increased prevalence of software means that authors of research papers have more awareness, and versioning may be more visible when researchers install and update software through packaging systems (including the work of resolving incompatibilities between versions or dependencies!). They may need to access online help forums such as Stack Overflow and found the version could be crucial for asking and searching for solutions. These software related practices can thus increase the saliency of versioning experienced by authors, leaving impressions of what is important to the scientific understanding of their work as well as reproducibility.

We were struck by the increase in software citation requests observed. We specifically investigated whether this overall increase was consistent across types of software, suspecting that proprietary software may make more explicit requests. However, we did not find statistically significant differences between the proportions of proprietary, non-commercial, and open source software that make citation requests, nor did we find differences with respect to how well those requests are followed by paper authors. Thus, we conclude that the practice of making citation requests is adopted by more overall. Software authors might notice prominent requests by others and then be motivated and educated to add their own. The presence of templates for language-specific features such as the citation() method in R, and the prevalence and visibility of CITATION files at the top level of code repositories may also influence software authors' choices and behaviors. This raises hopes for software visibility; recent promising efforts include GitHub moving citation request support to the "front page" of repositories (*GitHub, 2021*).

Challenges for advocacy

Our findings suggest that standards making and advocacy efforts should take existing practices into account, charting a course from current practices to hoped-for futures. For example, for researchers using software, we argue that the "like instrument" approach should be taken as a starting point for recommendations, such as re-purposing the location field for software repositories rather than a meaningless geographic location of a software publisher.

Our results about software archiving, general metadata availability, and persistent identification also have further clarified directions in which advocacy efforts can propose specific changes for different classes of software in terms of their property rights. We found over half of the open source software was archived and uniquely identified, with metadata available to fulfill citation needs. Some open source software projects were also aware

of requesting citation to the software artifact directly, but largely they were still cited by their companion publications. In such cases, citing an available publication is probably a convenient choice, which is more compatible with a researcher's conventional authoring workflow. As *Bouquin et al. (2020)* found that citation requests could be inconsistent when multiple online presences of software exist, recommendations can suggest concrete steps for making them actionable communications of citation expectations. The recommendations in *Katz et al. (2021)* & *Katz et al. (2019)* are a key step to ensuring that guidance begins at current practices for different kinds of software.

Another challenge rests with the citation requests of proprietary software (such as SPSS). Our observations of these requests included many that we thought almost unusable or incompatible with reference systems, including long-winded legalistic requests filled with® and ™ symbols and disclaimers of warranty; they seemed to be written by lawyers rather than specialists of scholarly communication. Advocacy in this area might therefore need to address lawyers or encourage developers to take ownership of these requests. An alternative might be for citation style guides to provide translation principles for these sorts of requests.

The persistence of non-machine readable citation requests (*e.g.*, free-text CITATION files) might also be a starting point for recommendations, such as providing migration paths and perhaps automatic synchronization between manual and machine-readable requests, meeting software producers where they are. Recent efforts are moving in this direction, including highlighting built-in language features for citation and advocating for more languages to include these (*e.g.*, *Katz et al., 2021*).

Maintenance costs are a likely challenge which advocacy should address. While the first step of adoption is not easy, as the efforts by *Allen (2021)* showed, making software and its citation human- and machine-actionable in the long term requires the upkeep of these formats and practices. One technical reason why the software artifact is recommended over its publication(s) as a citation target is that software is a very dynamic object. A single publication at one time cannot credit all those who contributed to it throughout its lifecycle. However, formal citation of the software artifact in a both machine- and human-actionable manner also raises the requirement for the upkeep of citation metadata and persistent identifier as the software artifact keeps to evolve. *Bouquin et al. (2020)* also found that the discoverable citation requests could become outdated when new releases of the software come out. It is likely that a non-trivial amount of regular work is needed for a software project to keep themselves citable and communicate the up-to-date citation expectations to their users.

Advocacy around persistent identifiers could also directly address the question of automated identifiers *vs.* manually created identifiers. This became clear to us as we reasoned around how to include the Software Heritage identifier in our annotation. The current archiving mechanism of Software Heritage is designed as a response to concerns of computational reproducibility (*Alliez et al., 2020*; *Cosmo, Gruenpeter & Zacchiroli, 2020*). By its endeavor to archive all software source codes (*Di Cosmo, 2018*), Software Heritage archives source codes crawled from major code hosting platforms. Software Heritage archives copies, with automatically generated identifiers and metadata, which can be cited

as part of the research workflow reported in publications. They are open to researchers' citation use and technically fulfill the advocacy recommendations for software citation; although they do not necessarily reflect the software creators' preference. In contrast, the manual creation of identifiers is a costly exercise that indicates that someone thinks them worth using. Language-specific software metadata and metadata stored in third party registries are also alternative sources of citation information, and advocacy might consider how researchers can be guided to use these available resources to cite software. However, in face of multiple sources, researchers need to be able to identify the appropriate record. We found, when searching for identifiers in systems that create them automatically, it was very difficult for users to identify the canonical archived repository for a package as distinguished from archived repositories of end-user code that forked the canonical repository (or even repositories that merely used the code). These issues are parallel to the recent discussion about non-creator-instigated software identification (*Katz, Bouquin & Chue Hong, 2019*), commonly concerning appropriate use of third party created software identification information.

The absence of persistent identifiers for proprietary software, whether manually or automatically created, also suggests a need for advocacy to address this specifically. Current infrastructures and policies that support software archiving and identification mechanism are designed primarily for open source code (*e.g.*, *Research Data Alliance/FORCE11 Software Source Code Identification Working Group et al., 2020*). As with citation metadata and requests, it may be easier for third parties to create and maintain these, rather than relying on influencing commercial software publishers. Indeed, software registries have already stored public accessible metadata and even suggested citations for proprietary software. Nonetheless, proprietary software are closed source and reuse would depend on purchase and be limited. The right for third-party verification and replication of the embodied methods or any future work directly built upon them is simply not granted. Neither do their commercial publishers need academic credit. As *Alliez et al. (2020)* and *Cosmo, Gruenpeter & Zacchiroli (2020)* have well distinguished the need for citation from that for reference, citation of the overall proprietary software project may be sufficient and the aspiration of referencing the specific software artifact may be unneeded.

Finally, our experience during the data collection for this study mirrored the reality in the age of "data deluge": While it is promising that a variety of software metadata are growing, accurate and comprehensive metadata retrieval is not straightforward for either humans or machines. Software publishers may post their citation requests across online locations using different formats and request citations of different publications for a single piece of software across its version history. This adds the challenge of identifying linkages between software and publications additional to the challenges of software identification (*Hata et al., 2021*).

CONCLUSION

In this study, we examined a sample of software mentions automatically extracted from PDFs of a large corpus of coronavirus research published since 2016. We manually examined

a stratified random sample, validating the extraction infrastructure with a false positive rate of 5%. We annotated our validated software mentions using an existing coding scheme extended with recommendations from recent advocacy, demonstrating agreement in its use among multiple annotators. In this way, we examined how software is mentioned, what functions they realize, and to what extent they conform to advocacy recommendations for software citation practices. In addition, we searched online to find and assess data about software packages, including if and how they make citation requests.

We found improvements when compared with prior studies of software mentions. We found increased mentions of software versions, increased adoption of open source software, and improved software accessibility. We also found over half of the open source software was archived, uniquely identified, and had metadata available, ready for citation needs. Finally we found a greater proportion of projects made a specific citation request.

On the other hand, other practices had not improved, or even moved in the other direction, compared to previous studies and advocates' recommendations. We found only a few formal citations directly to the software artifact and very little use of persistent identifiers. Crediting the authors of software in the text was still rare, mostly due to informal mentions. Worse, most mentions that do credit authors were of proprietary software, which are less likely to need to receive credit in order to keep maintaining the tool. Citation requests have potential to improve this situation, but we found these to be followed only in few cases.

Organically established practices may provide appropriate starting points for advocacy. For example, existing practices for software citation differ between proprietary, non-commercial, and open source software. Our findings emphasize the long-standing practice of "like instrument" mentions primarily for proprietary software, as well as divergent practices in the identification, archiving, and metadata provision of software with different property rights. Finally, they also differ in the common locations and widely used formats for citation requests. Advocacy may be more effective by leveraging these existing practices to minimize the behavioral change needed.

Future research suggests itself in three areas: publication type, decisions about software citation, and examining change over time in relation to advocacy.

Different publication types may have different patterns of software mentions. While we stratified the sample by impact factor and did not find any significant differences, we did not separate by article type. For example, it is perhaps worthwhile to separate publication venues specializing in publishing "software papers" that describe a software application or an algorithm; their software mention and citation patterns could be different from other domain science publications. Research could also separate pre-prints, or work to identify the differences among more granular fields and sub-fields.

Future research is needed to understand decisions about software citation and their pathway to publications. Citation requests do not yet seem effective, nor do we know much about the decisions leading to this ineffectiveness. For example, requests may not be visible to authors, or not visible at the right time. Even efforts to follow requests in a manuscript may be undermined by style guides, journal instructions, reviewers, or editors. Advocacy may not align well with the pressures involved during the production and publishing of

articles. Studying collections of drafts over time through the article creation process could create greater understanding of the manner in which mentions and reference lists are created and open up new locations or emphases for advocacy.[4]

Finally, future research should examine change over time, seeking evidence about effective advocacy. We hope that our extraction infrastructure, sampling approach, and content analysis scheme can be useful for comparable studies. Researchers can process future collections of PDFs produced by different groups, stratify, randomly sample mentions, and annotate and compare results to observe change. These observations could be focused on specific advocacy efforts, including micro efforts, such as comparing the publications of those exposed to specific training or interventions with those unexposed. Groups could be individuals, classes of individuals (such as software producers or early-career scholars), or larger groups such as users of specific software, classes of software or techniques, or specific fields. Assessments of interventions should be designed to give sufficient time for interventions to operate; timing of citation requests can be then compared to the period in which a study was authored and published.

Understanding patterns of software citation from research publications over time can inform advocacy and policy-making efforts to improve the visibility and rewards of software work in science. Our study suggests that software citation practices have not changed substantially in the past five years. Advocating for change takes time and we hope these results can provide a baseline against which to measure change in the future. Nonetheless, we think it possible that the behavior change required to implement new forms of visibility for software in publications may be too complex for quick uptake. Automated solutions that do not require behavioral change, such as entity extraction from PDFs to build software impact indexes, clearly have a role to play, both as a resource for improving advocacy and as a fallback for visibility where publication practices are slow to change.

[4] For example, in preparation of this publication, reviewers pointed out a missing URL in a dataset entry in our reference list. Even with our heightened attention given the topic of this paper, this error made it into the PDF; inquiry showed that the initial URL entry in the BibTeX had become masked in the toolchain from BibTeX through to the PeerJ LaTeX template and citation style specified for PDF rendering. We used a work-around in BibTeX to make the URL reference visible.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was supported by the Alfred P. Sloan Foundation (Award Number: 2016-7209) and the Gordon and Betty Moore Foundation (Grant Number: 8622). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Alfred P. Sloan Foundation: 2016-7209.
Gordon and Betty Moore Foundation: 8622.

Competing Interests

Patrice Lopez is employed by science-miner.

Author Contributions

- Caifan Du conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Johanna Cohoon performed the experiments, analyzed the data, authored or reviewed drafts of the article, and approved the final draft.
- Patrice Lopez performed the computation work, authored or reviewed drafts of the article, built the data pipeline and extraction systems, and approved the final draft.
- James Howison conceived and designed the experiments, performed the experiments, analyzed the data, authored or reviewed drafts of the article, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

The raw data is available at Zenodo: Patrice Lopez, Caifan Du, Hannah Cohoon & James Howison. (2021). Softcite software mention extraction from the CORD-19 publications (0.3.0) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5235661.

The analysis code is available at GitHub: https://github.com/caifand/cord19-sw-analysis.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.1022#supplemental-information.

REFERENCES

Allen A. 2021. Citation method, please? a case study in astrophysics. ArXiv preprint. arXiv:2111.12574.

Allen A, Schmidt J. 2014. Looking before leaping: creating a software registry. ArXiv preprint. arXiv:1407.5378.

Allen A, Teuben PJ, Ryan PW. 2018. Schroedinger's code: a preliminary study on research source code availability and link persistence in astrophysics. *The Astrophysical Journal Supplement Series* **236(1)**:10 DOI 10.3847/1538-4365/aab764.

Alliez P, Cosmo RD, Guedj B, Girault A, Hacid M-S, Legrand A, Rougier N. 2020. Attributing and referencing (research) software: best practices and outlook from Inria. *Computing in Science Engineering* **22(1)**:39–52.

Article Dataset Builder. 2020–2021. *Available at https://github.com/kermitt2/article-dataset-builder*.

Beltagy I, Lo K, Cohan A. 2019. Scibert: a pretrained language model for scientific text. ArXiv preprint. arXiv:1903.10676.

Bouquin DR, Chivvis DA, Henneken E, Lockhart K, Muench A, Koch J. 2020. Credit lost: two decades of software citation in astronomy. *The Astrophysical Journal Supplement Series* **249(1)**:8 DOI 10.3847/1538-4365/ab7be6.

Bradford SC. 1934. Sources of information on specific subjects. *Engineering* **137**:85–86.

Brase J, Lautenschlager M, Sens I. 2015. The tenth anniversary of assigning DOI names to scientific data and a five year history of DataCite. *D-Lib Magazine* **21(1/2)**:01brase DOI 10.1045/january2015-brase.

Brookes B. 1985. "Sources of information on specific subjects" by sc bradford. *Journal of Information Science* **10(4)**:173–175 DOI 10.1177/016555158501000406.

Chue Hong NP, Allen A, Gonzalez-Beltran A, De Waard A, Smith AM, Robinson C, Jones C, Bouquin D, Katz DS, Kennedy D, Ryder G, Hausman J, Hwang L, Jones MB, Harrison M, Crosas M, Wu M, Löwe P, Haines R, Edmunds S, Stall S, Swaminathan S, Druskat S, Crick T, Morrell T, Pollard T. 2019. Software citation checklist for authors. Technical report. Zenodo. *Available at https://zenodo.org/record/3479199*.

Conway JR, Lex A, Gehlenborg N. 2017. Upsetr: an r package for the visualization of intersecting sets and their properties. *Bioinformatics* **33(18)**:2938–2940 DOI 10.1093/bioinformatics/btx364.

Cosmo RD, Gruenpeter M, Zacchiroli S. 2020. Referencing source code artifacts: a separate concern in software citation. *Computing in Science Engineering* **22(2)**:33–43 DOI 10.1109/MCSE.2019.2963148.

Cosmo RD, Zacchiroli S. 2017. Software heritage: why and how to preserve software source code. In: *iPRES 2017: 14th international conference on digital preservation, Kyoto, Japan*.

Decan A, Mens T. 2021. What do package dependencies tell us about semantic versioning? *IEEE Transactions on Software Engineering* **47(6)**:1226–1240 DOI 10.1109/TSE.2019.2918315.

Di Cosmo R. 2018. Software heritage: why and how we collect, preserve and share all the software source code. In: *2018 IEEE/ACM 40th international conference on software engineering: software engineering in society (ICSE-SEIS)*. Piscataway: IEEE, 2–2.

Druskat S, Hong C, Haines R, Baker J. 2021. Citation File Format (CFF)—Specifications. *Available at https://citation-file-format.github.io/*.

Du C, Cohoon J, Lopez P, Howison J. 2021a. Softcite dataset: a dataset of software mentions in biomedical and economic research publications. *Journal of the Association for Information Science and Technology* **72(7)**:870–884 DOI 10.1002/asi.24454.

Du C, Cohoon J, Priem J, Piwowar HA, Meyer C, Howison J. 2021b. CiteAs: better software through sociotechnical change for better software citation. In: *CSCW '21 companion, virtual event*.

European Organization For Nuclear Research and OpenAIRE. 2013. Zenodo. *Available at https://about.zenodo.org*.

GitHub. 2021. About CITATION files. *Available at https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-repository-on-github/about-citation-files*.

GROBID. 2008–2021. GROBID. *Available at https://github.com/kermitt2/grobid*.

Habeas Corpus. 2021. Habeas Corpus. *Available at https://github.com/softwaresaved/habeas-corpus*.

Hata H, Guo JL, Kula RG, Treude C. 2021. Science-software linkage: the challenges of traceability between scientific knowledge and software artifacts. ArXiv preprint. arXiv:2104.05891.

Hong NPC, Katz DS, Barker M, Lamprecht AL, Martinez C, Psomopoulos FE, Harrow J, Castro LJ, Gruenpeter M, Martinez PA, Honeyman T, Struck A, Lee A, Loewe A, Van Werkhoven B, Jones C, Garijo D, Plomp E, Genova F, Shanahan H, Leng J, Hellström M, Sinha M, Kuzak M, Herterich P, Zhang Q, Islam S, Sansone S-A, Pollard T, Atmojo UD, Williams A, Czerniak A, Niehues A, Fouilloux AC, Desinghu B, Richard C, Gray C, Erdmann C, Nüst D, Tartarini D, Anzt H, Todorov I, McNally J, Moldon J, Burnett J, Belhajjame K, Sesink L, Hwang L, Roberto M, Wilkinson MD, Servillat M, Liffers M, Fox M, Lynch N, Lavanchy PM, Gesing S, Stevens S, Cuesta M, Peroni S, Soiland-Reyes S, Bakker T, Rabemanantsoa T, Sochat V, Yehudi Y. 2021. FAIR principles for research software (FAIR4RS Principles). *Research Data Alliance* DOI 10.15497/RDA00065.

Howison J, Bullard J. 2016. Software in the scientific literature: problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology* **67(9)**:2137–2155 DOI 10.1002/asi.23538.

Howison J, Herbsleb JD. 2011. Scientific software production: incentives and collaboration. In: *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. New York: ACM, 513–522.

Ison J, Rapacki K, Ménager H, Kalaš M, Rydza E, Chmura P, Anthon C, Beard N, Berka K, Bolser D, Booth T, Bretaudeau A, Brezovsky J, Casadio R, Cesareni G, Coppens F, Cornell M, Cuccuru G, Davidsen K, Vedova GD, Dogan T, Doppelt-Azeroual O, Emery L, Gasteiger E, Gatter T, Goldberg T, Grosjean M, Grüning B, Helmer-Citterich M, Ienasescu H, Ioannidis V, Jespersen MC, Jimenez R, Juty N, Juvan P, Koch M, Laibe C, Li J-W, Licata L, Mareuil F, Mičetić I, Friborg RM, Moretti S, Morris C, Möller S, Nenadic A, Peterson H, Profiti G, Rice P, Romano P, Roncaglia P, Saidi R, Schafferhans A, Schwämmle V, Smith C, Sperotto MM, Stockinger H, Vařeková RS, Tosatto SCE, Torre VDL, Uva P, Via A, Yachdav G, Zambelli F, Vriend G, Rost B, Parkinson H, Løngreen P, Brunak S. 2016. Tools and data services registry: a community effort to document bioinformatics resources. *Nucleic Acids Research* **44(D1)**:D38–D47 DOI 10.1093/nar/gkv1116.

Jones MB, Boettiger C, Mayes AC, Smith A, Slaughter P, Niemeyer K, Gil Y, Fenner M, Nowak K, Hahnel M, Coy L, Allen A, Crosas M, Sands A, Chu. Hong N, Cruse P, Katz DS, Goble C. 2017. Codemeta: an exchange schema for software metadata. version 2.0. *KNB Data Repository* DOI 10.5063/schema/codemeta-2.0.

Kanakia A, Wang K, Dong Y, Xie B, Lo K, Shen Z, Wang LL, Huang C, Eide D, Kohlmeier S, Wu C-H. 2020. Mitigating biases in CORD-19 for analyzing COVID-19 literature. *Frontiers in Research Metrics and Analytics* **5**:596624 DOI 10.3389/frma.2020.596624.

Katz DS, Bouquin D, Chue Hong N. 2019. Towards software non-creator-instigated identification (NCI) and citation. *Available at https://danielskatzblog.wordpress.com/2019/03/05/towards-software-non-creator-instigated-identification-nci-and-citation/*.

Katz DS, Bouquin D, Hong NPC, Hausman J, Jones C, Chivvis D, Clark T, Crosas M, Druskat S, Fenner M, Gillespie T, Gonzalez-Beltran A, Gruenpeter M, Habermann T, Haines R, Harrison M, Henneken E, Hwang L, Jones MB, Kelly AA, Kennedy DN, Leinweber K, Rios F, Robinson CB, Todorov I, Wu M, Zhang Q. 2019. Software citation implementation challenges. ArXiv preprint. arXiv:1905.08674.

Katz DS, Hong NPC, Clark T, Muench A, Stall S, Bouquin D, Cannon M, Edmunds S, Faez T, Feeney P, Fenner M, Friedman M, Grenier G, Harrison M, Heber J, Leary A, MacCallum C, Murray H, Pastrana E, Perry K, Schuster D, Stockhause M, Yeston J. 2021. Recognizing the value of software: a software citation guide. *F1000Research* **9**:1257 DOI 10.12688/f1000research.26932.2.

Katz DS, Smith AM. 2015. Transitive credit and json-ld. *Journal of Open Research Software* **3(1)**:e7 DOI 10.5334/jors.by.

Krüger F, Schindler D. 2020. A literature review on methods for the extraction of usage statements of software and data. *Computing in Science & Engineering* **22(1)**:26–38.

Lopez P. 2009. Grobid: combining automatic bibliographic data recognition and term extraction for scholarship publications. In: Agosti M, Borbinha J, Kapidakis S, Papatheodorou C, Tsakonas G, eds. *Research and advanced technology for digital libraries. ECDL 2009. Lecture notes in computer science, vol 5714*. Berlin, Heidelberg: Springer, 473–474 DOI 10.1007/978-3-642-04346-8_62.

Lopez P, Du C, Cohoon H, Howison J. 2021a. Softcite software mention extraction from the CORD-19 publications. DOI 10.5281/zenodo.5140437.

Lopez P, Du C, Cohoon J, Ram K, Howison J. 2021b. Mining software entities in scientific literature: document-level ner for an extremely imbalance and large-scale task. In: *Proceedings of the 30th ACM international conference on information and knowledge management (CIKM '21)*. New York: ACM.

Mayernik MS, Hart DL, Maull KE, Weber NM. 2017. Assessing and tracing the outcomes and impact of research infrastructures. *Journal of the Association for Information Science and Technology* **68(6)**:1341–1359 DOI 10.1002/asi.23721.

Monteil A, Gonzalez-Beltran A, Ioannidis A, Allen A, Lee A, Bandrowski A, Wilson BE, Mecum B, Du CF, Robinson C, Garijo D, Katz DS, Long D, Milliken G, Ménager H, Hausman J, Spaaks JH, Fenlon K, Vanderbilt K, Hwang L, Davis L, Fenner M, Crusoe MR, Hucka M, Wu M, Hong NC, Teuben P, Stall S, Druskat S, Carnevale T, Morrell T. 2020. Nine best practices for research software registries and repositories: a concise guide. ArXiv preprint. arXiv:2012.13117.

Muench A, Accomazzi A, Hol. Nielsen L, Blanco-Cuaresma S, Henneken EA, Ioannidis-Pantopikos A, Nowak K, Steffen J. 2020. Asclepias: an infrastructure project to improve software citation across astronomy. In: *Astronomical data analysis software and systems XXVII ADS. 522*. 711.

Nemiroff R, Wallin J. 1999. The astrophysics source code library: http://www.ascl.net. *Bulletin of the American Astronomical Society* **31**:885.

Newcombe RG. 1998. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine* **17(8)**:873–890 DOI 10.1002/(SICI)1097-0258(19980430)17:8<873::AID-SIM779>3.0.CO;2-I.

Pan X, Yan E, Cui M, Hua W. 2018. Examining the usage, citation, and diffusion patterns of bibliometric mapping software: a comparative study of three tools. *Journal of Informetrics* **12(2)**:481–493 DOI 10.1016/j.joi.2018.03.005.

Piwowar HA, Priem J. 2016. Depsy: valuing the software that powers science. GitHub. *Available at https://github.com/Impactstory/depsy-research*.

R Core Team. 2019. R: a language and environment for statistical computing. Vienna: R Foundation for Statistical Computing. *Available at https://www.r-project.org*.

Research Data Alliance/FORCE11 Software Source Code Identification Working Group, Allen A, Bandrowski A, Chan P, Di Cosmo R, Fenner M, Garcia L, Gruenpeter M, Jones CM, Katz DS, Kunze J, Schubotz M, Todorov IT. 2020. Software source code identification use cases and identifier schemes for persistent software source code identification. *Research Data Alliance* DOI 10.15497/RDA00053.

Schindler D, Bensmann F, Dietze S, Krüger F. 2021. SoMeSci—Software Mentions in Science. Type: dataset. DOI 10.5281/zenodo.4968738.

Schindler D, Bensmann F, Dietze S, Krüger F. 2022. The role of software in science: a knowledge graph-based analysis of software mentions in pubmed central. *PeerJ Computer Science* **8**:e835 DOI 10.7717/peerj-cs.835.

Smith AM, Katz DS, Niemeyer KE. 2016. Software citation principles. *PeerJ Computer Science* **2**:e86 DOI 10.7717/peerj-cs.86.

Softcite Software Mention Recognizer. 2018–2021. Softcite software mention recognition service. *Available at https://github.com/ourresearch/software-mentions*.

Software Heritage Development Documentation. 2021. SoftWare Heritage persistent IDentifiers (SWHIDs)—Software Heritage-Development Documentation documentation. *Available at https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html* (accessed on 25 Novemeber 2021).

Wade AD, Williams I. 2021. CORD-19 Software Mentions. Zenodo. *Available at https://zenodo.org/record/4582776*.

Wang LL, Lo K, Chandrasekhar Y, Reas R, Yang J, Burdick D, Eide D, Funk K, Katsis Y, Kinney RM, Li Y, Liu Z, Merrill W, Mooney P, Murdick DA, Rishi D, Sheehan J, Shen Z, Stilson B, Wade AD, Wang K, Wang NXR, Wilhelm C, Xie B, Raymond DM, Weld DS, Etzioni O, Kohlmeier S. 2020. CORD-19: the COVID-19 open research dataset. In: *Proceedings of the 1st workshop on NLP for COVID-19 at ACL 2020, online*. Columbus: Association for Computational Linguistics.

Wilson R. 2013. Encouraging citation of software—introducing CITATION files. *Available at https://www.software.ac.uk/blog/2016-10-06-encouraging-citation-software-introducing-citation-files*.

PeerJ Computer Science

Nine best practices for research software registries and repositories

Daniel Garijo[1], Hervé Ménager[2], Lorraine Hwang[3], Ana Trisovic[4], Michael Hucka[5], Thomas Morrell[5], Alice Allen[6], Task Force on Best Practices for Software Registries[7], and SciCodes Consortium[8]

[1] Universidad Politécnica de Madrid, Madrid, Spain
[2] Institut Pasteur, Université Paris Cité, Bioinformatics and Biostatistics Hub, Paris, France
[3] University of California, Davis, Davis, California, United States
[4] Harvard University, Boston, Massachusetts, United States
[5] California Institute of Technology, Pasadena, California, United States
[6] University of Maryland, College Park, MD, United States
[7] FORCE11 Software Citation Implementation Working Group
[8] Consortium of Scientific Software Registries and Repositories

ABSTRACT

Scientific software registries and repositories improve software findability and research transparency, provide information for software citations, and foster preservation of computational methods in a wide range of disciplines. Registries and repositories play a critical role by supporting research reproducibility and replicability, but developing them takes effort and few guidelines are available to help prospective creators of these resources. To address this need, the FORCE11 Software Citation Implementation Working Group convened a Task Force to distill the experiences of the managers of existing resources in setting expectations for all stakeholders. In this article, we describe the resultant best practices which include defining the scope, policies, and rules that govern individual registries and repositories, along with the background, examples, and collaborative work that went into their development. We believe that establishing specific policies such as those presented here will help other scientific software registries and repositories better serve their users and their disciplines.

Subjects Computer Education, Databases, Digital Libraries
Keywords Best practices, Research software repository, Research software registry, Software metadata, Repository policies, Research software registry guidelines

Submitted 28 September 2021
Accepted 9 June 2022
Published 8 August 2022

Corresponding author
Daniel Garijo, daniel.garijo@upm.es

Academic editor
Varun Gupta

Additional Information and Declarations can be found on page 24

DOI **10.7717/peerj-cs.1023**

INTRODUCTION

Research software is an essential constituent in scientific investigations (*Wilson et al., 2014*; *Momcheva & Tollerud, 2015*; *Hettrick, 2018*; *Lamprecht et al., 2020*), as it is often used to transform and prepare data, perform novel analyses on data, automate manual processes, and visualize results reported in scientific publications (*Howison & Herbsleb, 2011*). Research software is thus crucial for reproducibility and has been recognized by the scientific community as a research product in its own right—one that should be properly described, accessible, and credited by others (*Smith, Katz & Niemeyer, 2016*; *Chue Hong et al., 2021*). As a result of the increasing importance of computational methods, communities such as Research Data Alliance (RDA) (*Berman & Crosas, 2020*) (https://www.rd-alliance.org/) and FORCE11 (*Bourne et al., 2012*) (https://www.force11.org/)

emerged to enable collaboration and establish best practices. Numerous software services that enable open community development of and access to research source code, such as GitHub (https://github.com/) and GitLab (https://gitlab.com), appeared and found a role in science. General-purpose repositories, such as Zenodo (*CERN & OpenAIRE, 2013*) and FigShare (*Thelwall & Kousha, 2016*), have expanded their scope beyond data to include software, and new repositories, such as Software Heritage (*Di Cosmo & Zacchiroli, 2017*), have been developed specifically for software. A large number of domain-specific research software registries and repositories have emerged for different scientific disciplines to ensure dissemination and reuse among their communities (*Gentleman et al., 2004*; *Peckham, Hutton & Norris, 2013*; *Greuel & Sperber, 2014*; *Allen & Schmidt, 2015*; *Gil, Ratnakar & Garijo, 2015*; *Gil et al., 2016*).

Research software registries are typically indexes or catalogs of software metadata, without any code stored in them; while in *research software repositories*, software is both indexed *and* stored (*Lamprecht et al., 2020*). Both types of resource improve software discoverability and research transparency, provide information for software citations, and foster preservation of computational methods that might otherwise be lost over time, thereby supporting research reproducibility and replicability. Many provide or are integrated with other services, including indexing and archival services, that can be leveraged by librarians, digital archivists, journal editors and publishers, and researchers alike.

Transparency of the processes under which registries and repositories operate helps build trust with their user communities (*Yakel et al., 2013*; *Frank et al., 2017*). However, many domain research software resources have been developed independently, and thus policies amongst such resources are often heterogeneous and some may be omitted. Having specific policies in place ensures that users and administrators have reference documents to help define a shared understanding of the scope, practices, and rules that govern these resources.

Though recommendations and best practices for many aspects of science have been developed, no best practices existed that addressed the operations of software registries and repositories. To address this need, a Best Practices for Software Registries Task Force was proposed in June 2018 to the FORCE11 Software Citation Implementation Working Group (SCIWG) (https://github.com/force11/force11-sciwg). In seeking to improve the services software resources provide, software repository maintainers came together to learn from each other and promote interoperability. Both common practices and missing practices unfolded in these exchanges. These practices led to the development of nine best practices that set expectations for both users and maintainers of the resource by defining management of its contents and allowed usages as well as clarifying positions on sensitive issues such as attribution.

In this article, we expand on our pre-print "Nine Best Practices for Research Software Registries and Repositories: A Concise Guide" (*Task Force on Best Practices for Software Registries et al., 2020*) to describe our best practices and their development. Our guidelines are actionable, have a general purpose, and reflect the discussion of a community of more than 30 experts who handle over 14 resources (registries or

repositories) across different scientific domains. Each guideline provides a rationale, suggestions, and examples based on existing repositories or registries. To reduce repetition, we refer to registries and repositories collectively as "resources."

The remainder of the article is structured as follows. We first describe background and related efforts in "Background", followed by the methodology we used when structuring the discussion for creating the guidelines (Methodology). We then describe the nine best practices in "Best Practices for Repositories and Registries", followed by a discussion (Discussion). "Conclusions" concludes the article by summarizing current efforts to continue the adoption of the proposed practices. Those who contributed to the development of this article are listed in Appendix A, and links to example policies are given in Appendix B. Appendix C provides updated information about resources that have participated in crafting the best practices and an overview of their main attributes.

BACKGROUND

In the last decade, much was written about a reproducibility crisis in science (*Baker, 2016*) stemming in large part from the lack of training in programming skills and the unavailability of computational resources used in publications (*Merali, 2010*; *Peng, 2011*; *Morin et al., 2012*). On these grounds, national and international governments have increased their interest in releasing artifacts of publicly-funded research to the public (*Office of Science & Technology Policy, 2016*; *Directorate-General for Research & Innovation (European Commission), 2018*; *Australian Research Council, 2018*; *Chen et al., 2019*; *Ministère de l'Enseignement supérieur, de la Recherche et de l'Innovation, 2021*), and scientists have appealed to colleagues in their field to release software to improve research transparency (*Weiner et al., 2009*; *Barnes, 2010*; *Ince, Hatton & Graham-Cumming, 2012*) and efficiency (*Grosbol & Tody, 2010*). Open Science initiatives such as RDA and FORCE11 have emerged as a response to these calls for greater transparency and reproducibility. Journals introduced policies encouraging (or even requiring) that data and software be openly available to others (*Editorial Staff, 2019*; *Fox et al., 2021*). New tools have been developed to facilitate depositing research data and software in a repository (*Baruch, 2007*; *CERN & OpenAIRE, 2013*; *Di Cosmo & Zacchiroli, 2017*; *Clyburne-Sherin, Fei & Green, 2019*; *Brinckman et al., 2019*; *Trisovic et al., 2020*) and consequently, make them citable so authors and other contributors gain recognition and credit for their work (*Soito & Hwang, 2017*; *Du et al., 2021*).

Support for disseminating research outputs has been proposed with FAIR and FAIR4RS principles that state shared digital artifacts, such as data and software, should be Findable, Accessible, Interoperable, and Reusable (*Wilkinson et al., 2016*; *Lamprecht et al., 2020*; *Katz, Gruenpeter & Honeyman, 2021*; *Chue Hong et al., 2021*). Conforming with the FAIR principles for published software (*Lamprecht et al., 2020*) requires facilitating its discoverability, preferably in domain-specific resources (*Jiménez et al., 2017*). These resources should contain machine-readable metadata to improve the discoverability (Findable) and accessibility (Accessible) of research software through search engines or from within the resource itself. Furthering interoperability in FAIR is aided through the adoption of community standards *e.g.*, schema.org (*Guha, Brickley & Macbeth, 2016*) or

the ability to translate from one resource to another. The CodeMeta initiative (*Jones et al., 2017*) achieves this translation by creating a "Rosetta Stone" which maps the metadata terms used by each resource to a common schema. The CodeMeta schema (https://codemeta.github.io/) is an extension of schema.org which adds ten new fields to represent software-specific metadata. To date, CodeMeta has been adopted for representing software metadata by many repositories (https://hal.inria.fr/hal-01897934v3/codemeta).

As the usage of computational methods continues to grow, recommendations for improving research software have been proposed (*Stodden et al., 2016*) in many areas of science and software, as can be seen by the series of "Ten Simple Rules" articles offered by PLOS (*Dashnow, Lonsdale & Bourne, 2014*), sites such as AstroBetter (https://www.astrobetter.com/), courses to improve skills such as those offered by The Carpentries (https://carpentries.org/), and attempts to measure the adoption of recognized best practices (*Serban et al., 2020*; *Trisovic et al., 2022*). Our quest for best practices complements these efforts by providing guides to the specific needs of research software registries and repositories.

METHODOLOGY

The best practices presented in this article were developed by an international Task Force of the FORCE11 Software Citation Implementation Working Group (SCIWG). The Task Force was proposed in June 2018 by author Alice Allen, with the goal of developing a list of best practices for software registries and repositories. Working Group members and a broader group of managers of domain specific software resources formed the inaugural group. The resulting Task Force members were primarily managers and editors of resources from Europe, United States, and Australia. Due to the range in time zones, the Task Force held two meetings 7 h apart, with the expectation that, except for the meeting chair, participants would attend one of the two meetings. We generally refer to two meetings on the same day with the singular "meeting" in the discussions to follow.

The inaugural Task Force meeting (February, 2019) was attended by 18 people representing 14 different resources. Participants introduced themselves and provided some basic information about their resources, including repository name, starting year, number of records, and scope (discipline-specific or general purpose), as well as services provided by each resource (*e.g.*, support of software citation, software deposits, and DOI minting). Table 1 presents an overview of the collected responses, which highlight the efforts of the Task Force chairs to bring together both discipline-specific and general purpose resources. The "Other" category indicates that the answer needed clarifying text (*e.g.*, for the question "is the repository actively curated?" some repositories are not manually curated, but have validation checks). Appendix C provides additional information on the questions asked to resource managers (Table C.1) and their responses (Tables C.2–C.4).

During the inaugural Task Force meeting, the chair laid out the goal of the Task Force, and the group was invited to brainstorm to identify commonalities for building a list of best practices. Participants also shared challenges they had faced in running their resources

Table 1 Overview of the information shared by the 14 resources which participated in the first Task Force meeting.

Question	#Yes	#No	#Other
Is the resource discipline-specific?	6	8	0
Does the resource accept software only?	8	6	0
Does the resource require a software deposit?	2	12	0
Does the resource accept software deposits?	10	4	0
Can the resource mint DOIs?	6	8	0
Is the resource actively curated?	10	1	3
Can the resource be used to cite software?	11	2	1

and policies they had enacted to manage these resources. The result of the brainstorming and discussion was a list of ideas collected in a common document.

Starting in May 2019 and continuing through the rest of 2019, the Task Force met on the third Thursday of each month and followed an iterative process to discuss, add to, and group ideas; refine and clarify the ideas into different practices, and define the practices more precisely. It was clear from the onset that, though our resources have goals in common, they are also very diverse and would be best served by best practices that were descriptive rather than prescriptive. We reached consensus on whether a practice should be a *best* practice through discussion and informal voting. Each best practice was given a title and a list of questions or needs that it addressed.

Our initial plan aimed at holding two Task Force meetings on the same day each month, in order to follow a common agenda with independent discussions built upon the previous month's meeting. However, the later meeting was often advantaged by the earlier discussion. For instance, if the early meeting developed a list of examples for one of the guidelines, the late meeting then refined and added to the list. Hence, discussions were only duplicated when needed, *e.g.*, where there was no consensus in the early group, and often proceeded in different directions according to the group's expertise and interest. Though we had not anticipated this, we found that holding two meetings each month on the same day accelerated the work, as work done in the second meeting of the day generally continued rather than repeating work done in the first meeting.

The resulting consensus from the meetings produced a list of the most broadly applicable practices, which became the initial list of best practices participants drew from during a two-day workshop, funded by the Sloan Foundation and held at the University of Maryland College Park, in November, 2019 (Scientific Software Registry Collaboration Workshop). A goal of the workshop was to develop the final recommendations on best practices for repositories and registries to the FORCE11 SCIWG. The workshop included participants outside the Task Force resulting in a broader set of contributions to the final list. In 2020, this group made additional refinements to the best practices during virtual meetings and through online collaborative writing producing in the guidelines described in the next section. The Task Force then transitioned into the SciCodes consortium (http://scicodes.net). SciCodes is a permanent community for research

software registries and repositories with a particular focus on these best practices. SciCodes continued to collect information about involved registries and repositories, which are listed in Appendix C. We also include some analysis of the number of entries and date of creation of member resources. Appendix A lists the people who participated in these efforts.

BEST PRACTICES FOR REPOSITORIES AND REGISTRIES

Our recommendations are provided as nine separate policies or statements, each presented below with an explanation as to why we recommend the practice, what the practice describes, and specific considerations to take into account. The last paragraph of each best practice includes one or two examples and a link to Appendix B, which contains many examples from different registries and repositories.

These nine best practices, though not an exhaustive list, are applicable to the varied resources represented in the Task Force, so are likely to be broadly applicable to other scientific software repositories and registries. We believe that adopting these practices will help document, guide, and preserve these resources, and put them in a stronger position to serve their disciplines, users, and communities[1].

[1] Please note that the information provided in this article does not constitute legal advice.

Provide a public scope statement

The landscape of research software is diverse and complex due to the overlap between scientific domains, the variety of technical properties and environments, and the additional considerations resulting from funding, authors' affiliation, or intellectual property. A scope statement clarifies the type of software contained in the repository or indexed in the registry. Precisely defining a scope, therefore, helps those users of the resource who are looking for software to better understand the results they obtained.

Moreover, given that many of these resources accept submission of software packages, providing a precise and accessible definition will help researchers determine whether they should register or deposit software, and curators by making clear what is out of scope for the resource. Overall, a public scope manages the expectations of the potential depositor as well as the software seeker. It informs both what the resource does and does not contain.

The scope statement should describe:

- What is accepted, and acceptable, based on criteria covering scientific discipline, technical characteristics, and administrative properties
- What is not accepted, *i.e.*, characteristics that preclude their incorporation in the resource
- Notable exceptions to these rules, if any

Particular criteria of relevance include the scientific community being served and the types of software listed in the registry or stored in the repository, such as source code, compiled executables, or software containers. The scope statement may also include criteria that must be satisfied by accepted software, such as whether certain software quality metrics must be fulfilled or whether a software project must be used in published

research. Availability criteria can be considered, such as whether the code has to be publicly available, be in the public domain and/or have a license from a predefined set, or whether software registered in another registry or repository will be accepted.

An illustrating example of such a scope statement is the editorial policy (https://ascl.net/wordpress/submissions/editiorial-policy/) published by the Astrophysics Source Code Library (ASCL) (*Allen et al., 2013*), which states that it includes only software source code used in published astronomy and astrophysics research articles, and specifically excludes software available only as a binary or web service. Though the ASCL's focus is on research documented in peer-reviewed journals, its policy also explicitly states that it accepts source code used in successful theses. Other examples of scope statements can be found in Appendix B.

Provide guidance for users

Users accessing a resource to search for entries and browse or retrieve the description(s) of one or more software entries have to understand how to perform such actions. Although this guideline potentially applies to many public online resources, especially research databases, the potential complexity of the stored metadata and the curation mechanisms can seriously impede the understandability and usage of software registries and repositories.

User guidance material may include:

- How to perform common user tasks, such as searching the resource, or accessing the details of an entry
- Answers to questions that are often asked or can be anticipated, *e.g.*, with Frequently Asked Questions or tips and tricks pages
- Who to contact for questions or help

A separate section in these guidelines on the *Conditions of use policy* covers terms of use of the resource and how best to cite records in a resource and the resource itself.

Guidance for users who wish to contribute software is covered in the next section, *Provide guidance to software contributors*.

When writing guidelines for users, it is advisable to identify the types of users your resource has or could potentially have and corresponding use cases. Guidance itself should be offered in multiple forms, such as in-field prompts, linked explanations, and completed examples. Any machine-readable access, such as an API, should be fully described directly in the interface or by providing a pointer to existing documentation, and should specify which formats are supported (*e.g.*, JSON-LD, XML) through content negotiation if it is enabled.

Examples of such elements include, for instance, the bio.tools registry (*Ison et al., 2019*) API user guide (https://biotools.readthedocs.io/en/latest/api_usage_guide.html), or the ORNL DAAC (*ORNL, 2013*) instructions for data providers (https://daac.ornl.gov/submit/). Additional examples of user guidance can be found in Appendix B.

Provide guidance to software contributors

Most software registries and repositories rely on a community model, whereby external contributors will provide software entries to the resource. The scope statement will already have explained *what* is accepted and what is not; the contributor policy addresses *who* can add or change software entries and the processes involved.

The contributor policy should therefore describe:

- Who can or cannot submit entries and/or metadata
- Required and optional metadata expected for deposited software
- Review process, if any
- Curation process, if any
- Procedures for updates (*e.g.*, who can do it, when it is done, how is it done)

Topics to consider when writing a contributor policy include whether the author(s) of a software entry will be contacted if the contributor is not also an author and whether contact is a condition or side-effect of the submission. Additionally, a contributor policy should specify how persistent identifiers are assigned (if used) and should state that depositors must comply with all applicable laws and not be intentionally malicious.

Such material is provided in resources such as the Computational Infrastructure for Geodynamics (*Hwang & Kellogg, 2017*) software contribution checklist (https://github.com/geodynamics/best_practices/blob/master/ContributingChecklist.md#contributing-software) and the CoMSES Net Computational Model Library (*Janssen et al., 2008*) model archival tutorial (https://forum.comses.net/t/archiving-your-model-1-gettingstarted/7377). Additional examples of guidance for software contributors can be found in Appendix B.

Establish an authorship policy

Because research software is often a research product, it is important to report authorship accurately, as it allows for proper scholarly credit and other types of attributions (*Smith, Katz & Niemeyer, 2016*). However, even though authorship should be defined at the level of a given project, it can prove complicated to determine (*Alliez et al., 2019*). Roles in software development can widely vary as contributors change with time and versions, and contributions are difficult to gauge beyond the "commit," giving rise to complex situations. In this context, establishing a dedicated policy ensures that people are given due credit for their work. The policy also serves as a document that administrators can turn to in case disputes arise and allows proactive problem mitigation, rather than having to resort to reactive interpretation. Furthermore, having an authorship policy mirrors similar policies by journals and publishers and thus is part of a larger trend. Note that the authorship policy will be communicated at least partially to users through guidance provided to software contributors. Resource maintainers should ensure this policy remains consistent with the citation policies for the registry or repository (usually, the citation requirements for each piece of research software are under the authority of its owners).

The authorship policy should specify:

- How authorship is determined *e.g.*, a stated criteria by the contributors and/or the resource
- Policies around making changes to authorship
- The conflict resolution processes adopted to handle authorship disputes

When defining an authorship policy, resource maintainers should take into consideration whether those who are not coders, such as software testers or documentation maintainers, will be identified or credited as authors, as well as criteria for ordering the list of authors in cases of multiple authors, and how the resource handles large numbers of authors and group or consortium authorship. Resources may also include guidelines about how changes to authorship will be handled so each author receives proper credit for their contribution. Guidelines can help facilitate determining every contributors' role. In particular, the use of a credit vocabulary, such as the Contributor Roles Taxonomy (*Allen, O'Connell & Kiermer, 2019*), to describe authors' contributions should be considered for this purpose (http://credit.niso.org/).

An example of authorship policy is provided in the Ethics Guidelines (https://joss.theoj.org/about#ethics) and the submission guide authorship section (https://joss.readthedocs.io/en/latest/submitting.html#authorship) of the *Journal of Open Source Software* (*Katz, Niemeyer & Smith, 2018*), which provides rules for inclusion in the authors list. Additional examples of authorship policies can be found in Appendix B.

Document and share your metadata schema

The structure and semantics of the information stored in registries and repositories is sometimes complex, which can hinder the clarity, discovery, and reuse of the entries included in these resources. Publicly posting the metadata schema used for the entries helps individual and organizational users interested in a resource's information understand the structure and properties of the deposited information. The metadata structure helps to inform users how to interact with or ingest records in the resource. A metadata schema mapped to other schemas and an API specification can improve the interoperability between registries and repositories.

This practice should specify:

- The schema used and its version number. If a standard or community schema, such as CodeMeta (*Jones et al., 2017*) or schema.org (*Guha, Brickley & Macbeth, 2016*) is used, the resource should reference its documentation or official website. If a custom schema is used, formal documentation such as a description of the schema and/or a data dictionary should be provided.
- Expected metadata when submitting software, including which fields are required and which are optional, and the format of the content in each field.

To improve the readability of the metadata schema and facilitate its translation to other standards, resources may provide a mapping (from the metadata schema used in the

resource) to published standard schemas, through the form of a "cross-walk" (*e.g.*, the CodeMeta cross-walk (https://codemeta.github.io/crosswalk/)) and include an example entry from the repository that illustrates all the fields of the metadata schema. For instance, extensive documentation (https://biotoolsschema.readthedocs.io/en/latest/) is available for the biotoolsSchema (*Ison et al., 2021*) format, which is used in the bio.tools registry. Another example is the OntoSoft vocabulary (http://ontosoft.org/software), used by the OntoSoft registry (*Gil, Ratnakar & Garijo, 2015*; *Gil et al., 2016*) and available in both machine-readable and human readable formats. Additional examples of metadata schemas can be found in Appendix B.

Stipulate conditions of use

The *conditions of use* document the terms under which users may use the contents provided by a website. In the case of software registries and repositories, these conditions should specifically state how the metadata regarding the entities of a resource can be used, attributed, and/or cited, and provide information about the licenses used for the code and binaries. This policy can forestall potential liabilities and difficulties that may arise, such as claims of damage for misinterpretation or misapplication of metadata. In addition, the conditions of use should clearly state how the metadata can and cannot be used, including for commercial purposes and in aggregate form.

This document should include:

- Legal disclaimers about the responsibility and liability borne by the registry or repository
- License and copyright information, both for individual entries and for the registry or repository as a whole
- Conditions for the use of the metadata, including prohibitions, if any
- Preferred format for citing software entries
- Preferred format for attributing or citing the resource itself

When writing conditions of use, resource maintainers might consider what license governs the metadata, if licensing requirements apply for findings and/or derivatives of the resource, and whether there are differences in the terms and license for commercial *vs* noncommercial use. Restrictions on the use of the metadata may also be included, as well as a statement to the effect that the registry or repository makes no guarantees about completeness and is not liable for any damages that could arise from the use of the information. Technical restrictions, such as conditions of use of the API (if one is available), may also be mentioned.

Conditions of use can be found for instance for DOE CODE (*Ensor et al., 2017*), which in addition to the general conditions of use (https://www.osti.gov/disclaim) specifies that the rules for usage of the hosted code (https://www.osti.gov/doecode/faq#are-there-restrictions) are defined by their respective licenses. Additional examples of conditions of use policies can be found in Appendix B.

State a privacy policy

Privacy policies define how personal data about users are stored, processed, exchanged or removed. Having a privacy policy demonstrates a strong commitment to the privacy of users of the registry or repository and allows the resource to comply with the legal requirement of many countries in addition to those a home institution and/or funding agencies may impose.

The privacy policy of a resource should describe:

- What information is collected and how long it is retained
- How the information, especially any personal data, is used
- Whether tracking is done, what is tracked, and how (*e.g.*, Google Analytics)
- Whether cookies are used

When writing a privacy policy, the specific personal data which are collected should be detailed, as well as the justification for their resource, and whether these data are sold and shared. Additionally, one should list explicitly the third-party tools used to collect analytic information and potentially reference their privacy policies. If users can receive emails as a result of visiting or downloading content, such potential solicitations or notifications should be announced. Measures taken to protect users' privacy and whether the resource complies with the *European Union Directive on General Data Protection Regulation* (https://gdpr-info.eu/) (GDPR) or other local laws, if applicable, should be explained[2]. As a precaution, the statement can reserve the right to make changes to this privacy policy. Finally, a mechanism by which users can request the removal of such information should be described.

For example, the SciCrunch's (*Grethe et al., 2014*) privacy policy (https://scicrunch.org/page/privacy) details what kind of personal information is collected, how it is collected, and how it may be reused, including by third-party websites through the use of cookies. Additional examples of privacy policies can be found in Appendix B.

[2] In the case of GDPR, the regulation applies to all European user personal data, even if the resource is not located in Europe.

Provide a retention policy

Many software registries and repositories aim to facilitate the discovery and accessibility of the objects they describe, *e.g.*, enabling search and citation, by making the corresponding records permanently accessible. However, for various reasons, even in such cases maintainers and curators may have to remove records. Common examples include removing entries that are outdated, no longer meet the scope of the registry, or are found to be in violation of policies. The resource should therefore document retention goals and procedures so that users and depositors are aware of them.

The retention policy should describe:

- The length of time metadata and/or files are expected to be retained;
- Under what conditions metadata and/or files are removed;
- Who has the responsibility and ability to remove information;
- Procedures to request that metadata and/or files be removed.

The policy should take into account whether best practices for persistent identifiers are followed, including resolvability, retention, and non-reuse of those identifiers. The retention time provided by the resource should not be too prescriptive (*e.g.*, "for the next 10 years"), but rather it should fit within the context of the underlying organization(s) and its funding. This policy should also state who is allowed to edit metadata, delete records, or delete files, and how these changes are performed to preserve the broader consistency of the registry. Finally, the process by which data may be taken offline and archived as well as the process for its possible retrieval should be thoroughly documented.

As an example, Bioconductor (*Gentleman et al., 2004*) has a deprecation process through which software packages are removed if they cannot be successfully built or tested, or upon specific request from the package maintainer. Their policy (https://bioconductor.org/developers/package-end-of-life/) specifies who initiates this process and under which circumstances, as well as the successive steps that lead to the removal of the package. Additional examples of retention policies can be found in Appendix B.

Disclose your end-of-life policy

Despite their usefulness, the long-term maintenance, sustainability, and persistence of online scientific resources remains a challenge, and published web services or databases can disappear after a few years (*Veretnik, Fink & Bourne, 2008*; *Kern, Fehlmann & Keller, 2020*). Sharing a clear end-of-life policy increases trust in the community served by a registry or repository. It demonstrates a thoughtful commitment to users by informing them that provisions for the resource have been considered should the resource close or otherwise end its services for its described artifacts. Such a policy sets expectations and provides reassurance as to how long the records within the registry will be findable and accessible in the future.

This policy should describe:

- Under what circumstances the resource might end its services;
- What consequences would result from closure;
- What will happen to the metadata and/or the software artifacts contained in the resource in the event of closure;
- If long-term preservation is expected, where metadata and/or software artifacts will be migrated for preservation;
- How a migration will be funded.

Publishing an end-of-life policy is an opportunity to consider, in the event a resource is closed, whether the records will remain available, and if so, how and for whom, and under which conditions, such as archived status or "read-only." The restrictions applicable to this policy, if any, should be considered and detailed. Establishing a formal agreement or memorandum of understanding with another registry, repository, or institution to receive and preserve the data or project, if applicable, might help to prepare for such a liability.

Examples of such policies include the Zenodo end-of-life policy (https://help.zenodo.org/), which states that if Zenodo ceases its services, the data hosted in the resource will be

migrated and the DOIs provided would be updated to resolve to the new location (currently unspecified). Additional examples of end-of-life policies can be found in Appendix B.

A summary of the practices presented in this section can be found in Table 2.

DISCUSSION

The best practices described above serve as a guide for repositories and registries to provide better service to their users, ranging from software developers and researchers to publishers and search engines, and enable greater transparency about the operation of their described resources. Implementing our practices provides users with significant information about *how* different resources operate, while preserving important institutional knowledge, standardizing expectations, and guiding user interactions.

For instance, a public scope statement and guidance for users may directly impact usability and, thus, the popularity of the repository. Resources including tools with a simple design and unambiguous commands, as well as infographic guides or video tutorials, ease the learning curve for new users. The guidance for software contributions, conditions of use, and sharing the metadata schema used may help eager users contribute new functionality or tools, which may also help in creating a community around a resource. A privacy policy has become a requirement across geographic boundaries and legal jurisdictions. An authorship policy is critical in facilitating collaborative work among researchers and minimizing the chances for disputes. Finally, retention and end-of-life policies increase the trust and integrity of a repository service.

Policies affecting a single community or domain were deliberately omitted when developing the best practices. First, an exhaustive list would have been a barrier to adoption and not applicable to every repository since each has a different perspective, audience, and motivation that drives policy development for their organization. Second, best practices that regulate the content of a resource are typically domain-specific to the artifact and left to resources to stipulate based on their needs. Participants in the 2019 Scientific Software Registry Collaboration Workshop were surprised to find that only four metadata elements were shared by all represented resources[3]. The diversity of our resources precludes prescriptive requirements, such as requiring specific metadata for records, so these were also deliberately omitted in the proposed best practices.

[3] The elements were: software name, description, keywords, and URL.

Hence, we focused on broadly applicable practices considered important by various resources. For example, amongst the participating registries and repositories, very few had codes of conduct that govern the behavior of community members. Codes of conduct are warranted if resources are run as part of a community, especially if comments and reviews are solicited for deposits. In contrast, a code of conduct would be less useful for resources whose primary purpose is to make software and software metadata available for reuse. However, this does not negate their importance and their inclusion as best practices in other arenas concerning software.

As noted by the FAIR4RS movement, software is different than data, motivating the need for a separate effort to address software resources (*Lamprecht et al., 2020*; *Katz et al., 2016*). Even so, there are some similarities, and our effort complements and aligns well

Table 2 Summary of the best practices with recommendations and examples.

Practice, description and examples	Recommendations
1. Provide a public scope statement	• What is accepted, and acceptable, based on criteria covering scientific discipline, technical characteristics, and administrative properties
Informs both software depositor and resource seeker what the collection does and does not contain.	• What is not accepted, *i.e.*, characteristics that preclude their incorporation in the resource
Example: ASCL editorial policy.	• Notable exceptions to these rules, if any
2. Provide guidance for users	• How to perform common user tasks, like searching for collection, or accessing the details of an entry
Helps users accessing a resource understand how to perform tasks like searching, browsing, and retrieving software entries.	• Answers to questions that are often asked or can be anticipated
Example: bio.tools registry API user guide.	• Point of contact for help and questions
3. Provide guidance to software contributors	• Who can or cannot submit entries and/or metadata
Specifies who can add or change software entries and explains the necessary processes.	• Required and optional metadata expected from software contributors
Example: Computational Infrastructure for Geodynamics contribution checklist.	• Procedures for updates, review process, curation process
4. Establish an authorship policy	• How authorship is determined *e.g.*, a stated criteria by the contributors and/or the resource
Ensures that contributors are given due credit for their work and to resolve disputes in case of conflict.	• Policies around making changes to authorship
Example: JOSS authorship policy.	• Define the conflict resolution processes
5. Document and share your metadata schema	• Specify the used schema and its version number. Add reference to its documentation or official website. If a custom schema is used, provide documentation.
Revealing the metadata schema used helps users understand the structure and properties of the deposited information.	• Expected metadata when submitting software
Example: OntoSoft vocabulary from the OntoSoft registry.	
6. Stipulate conditions of use	• Legal disclaimers about the responsibility and liability borne by the resource
Documents the terms under which users may use the provided resources, including metadata and software.	• License and copyright information, both for individual entries and for the resource as a whole
Example: DOE CODE acceptable use policy.	• Conditions for the use of the metadata, including prohibitions, if any
	• Preferred format for citing software entries; preferred format for attributing or citing the resource itself
7. State a privacy policy	• What information is collected and how long it is retained
Defines how personal data about users are stored, processed, exchanged, or removed.	• How the information, especially any personal data, is used
Example: SciCrunch's privacy policy.	• Whether tracking is done, what is tracked, and how; whether cookies are used
8. Provide a retention policy	• The length of time metadata and/or files are expected to be retained
Helps both users and depositors understand and anticipate retention goals and procedures.	• Under what conditions metadata and/or files are removed
Example: Bioconductor package deprecation.	• Who has the responsibility and ability to remove information; procedures to request that metadata and/or files be removed
9. Disclose end-of-life policy	• Circumstances under which the resource might end its services
Informs both users and depositors of how long the records within the resource will be findable and accessible in the future.	• What consequences would result from closure
Example: Zenodo end-of-life policy.	• What will happen to the metadata and/or the software artifacts contained in the resource in the event of closure
	• If long-term preservation is expected, where metadata and/or software artifacts will be migrated for preservation; how a migration will be funded

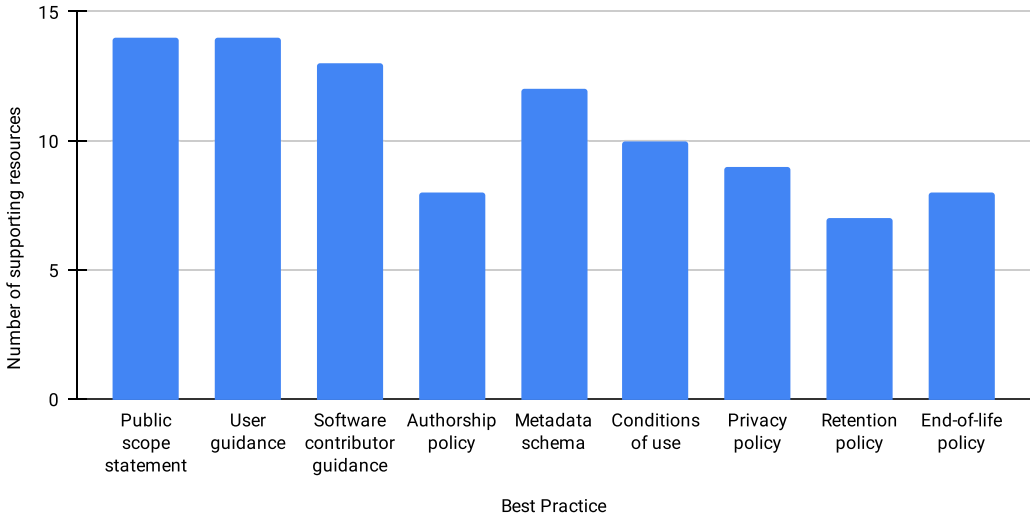


Figure 1 **Number of resources supporting each best practice, out of 14 resources.**
Full-size DOI: 10.7717/peerj-cs.1023/fig-1

with recent guidelines developed in parallel to increase the transparency, responsibility, user focus, sustainability, and technology of data repositories. For example, both the TRUST Principles (*Lin et al., 2020*) and CoreTrustSeal Requirements (*CoreTrustSeal, 2019*) call for a repository to provide information on its scope and list the terms of use of its metadata to be considered compliant with TRUST or CoreTrustSeal, which aligns with our practices "*Provide a public scope statement*" and "*Stipulate conditions of use*". CoreTrustSeal and TRUST also require that a repository consider continuity of access, which we have expressed as the practice to "*Disclosing your end-of-life policy*". Our best practices differ in that they do not address, for example, staffing needs nor professional development for staff, as CoreTrustSeal requires, nor do our practices address protections against cyber or physical security threats, as the TRUST principles suggest. Inward-facing policies, such as documenting internal workflows and practices, are generally good in reducing operational risks, but internal management practices were considered out of scope of our guidelines.

Figure 1 shows the number of resources that support (partially or in their totality) each best practice. Though we see the proposed best practices as critical, many of the repositories that have actively participated in the discussions (14 resources in total) have yet to implement every one of them. We have observed that the first three practices (providing public scope statement, add guidance for users and for software contributors) have the widest adoption, while the retention, end-of-life, and authorship policy the least. Understanding the lag in the implementation across all of the best practices requires further engagement with the community.

Improving the adoption of our guidelines is one of the goals of SciCodes (http://scicodes.net), a recent consortium of scientific software registries and repositories. SciCodes evolved from the Task Force as a permanent community to continue the dialogue and share information between domains, including sharing of tools and ideas. SciCodes

has also prioritized improving software citation (complementary to the efforts of the FORCE11 SCIWG) and tracking the impact of metadata and interoperability. In addition, SciCodes aims to understand barriers to implementing policies, ensure consistency between various best practices, and continue advocacy for software support by continuing dialogue between registries, repositories, researchers, and other stakeholders.

CONCLUSIONS

The dissemination and preservation of research material, where repositories and registries play a key role, lies at the heart of scientific advancement. This article introduces nine best practices for research software registries and repositories. The practices are an outcome of a Task Force of the FORCE11 Software Citation Implementation Working Group and reflect the discussion, collaborative experiences, and consensus of over 30 experts and 14 resources.

The best practices are non-prescriptive, broadly applicable, and include examples and guidelines for their adoption by a community. They specify establishing the working domain (scope) and guidance for both users and software contributors, address legal concerns with privacy, use, and authorship policies, enhance usability by encouraging metadata sharing, and set expectations with retention and end-of-life policies. However, we believe additional work is needed to raise awareness and adoption across resources from different scientific disciplines. Through the SciCodes consortium, our goal is to continue implementing these practices more uniformly in our own registries and repositories and reduce the burdens of adoption. In addition to completing the adoption of these best practices, SciCodes will address topics such as tracking the impact of good metadata, improving interoperability between registries, and making our metadata discoverable by search engines and services such as Google Scholar, ORCID, and discipline indexers.

APPENDIX A: CONTRIBUTORS

The following people contributed to the development of this article through participation in the Best Practices Task Force meetings, 2019 Scientific Software Registry Collaboration Workshop, and/or SciCodes Consortium meetings:

Alain Monteil, Inria, HAL; Software Heritage

Alejandra Gonzalez-Beltran, Science and Technology Facilities Council, UK Research and Innovation, Science and Technology Facilities Council

Alexandros Ioannidis, CERN, Zenodo

Alice Allen, University of Maryland, College Park, Astrophysics Source Code Library

Allen Lee, Arizona State University, CoMSES Net Computational Model Library

Ana Trisovic, Harvard University, DataVerse

Anita Bandrowski, UCSD, SciCrunch

Bruce E. Wilson, Oak Ridge National Laboratory, ORNL Distributed Active Archive Center for Biogeochemical Dynamics

Bryce Mecum, NCEAS, UC Santa Barbara, CodeMeta
Caifan Du, iSchool, University of Texas at Austin, CiteAs
Carly Robinson, US Department of Energy, Office of Scientific and Technical Information, DOE CODE
Daniel Garijo, Universidad Politécnica de Madrid (formerly at Information Sciences Institute, University of Southern California), Ontosoft
Daniel S. Katz, University of Illinois at Urbana-Champaign, Associate EiC for JOSS, FORCE11 Software Citation Implementation Working Group, co-chair
David Long, Brigham Young University, IEEE GRS Remote Sensing Code Library
Genevieve Milliken, NYU Bobst Library, IASGE
Hervé Ménager, Hub de Bioinformatique et Biostatistique—Département Biologie Computationnelle, Institut Pasteur, ELIXIR bio.tools
Jessica Hausman, Jet Propulsion Laboratory, PO.DAAC
Jurriaan H. Spaaks, Netherlands eScience Center, Research Software Directory
Katrina Fenlon, University of Maryland, iSchool
Kristin Vanderbilt, Environmental Data Initiative, IMCR
Lorraine Hwang, University California Davis, Computational Infrastructure for Geodynamics
Lynn Davis, US Department of Energy, Office of Scientific and Technical Information, DOE CODE
Martin Fenner, Front Matter (formerly at DataCite), FORCE11 Software Citation Implementation Working Group, co-chair
Michael R. Crusoe, CWL, Debian-Med
Michael Hucka, California Institute of Technology, SBML; COMBINE
Mingfang Wu, Australian Research Data Commons, Australian Research Data Commons
Morane Gruenpeter, Inria, Software Heritage
Moritz Schubotz, FIZ Karlsruhe - Leibniz-Institute for Information Infrastructure, swMATH
Neil Chue Hong, Software Sustainability Institute/University of Edinburgh, Software Sustainability Institute; FORCE11 Software Citation Implementation Working Group, co-chair
Pete Meyer, Harvard Medical School, SBGrid; BioGrids
Peter Teuben, University of Maryland, College Park, Astrophysics Source Code Library
Piotr Sliz, Harvard Medical School, SBGrid; BioGrids
Sara Studwell, US Department of Energy, Office of Scientific and Technical Information, DOE CODE
Shelley Stall, American Geophysical Union, AGU Data Services
Stephan Druskat, German Aerospace Center (DLR)/University Jena/Humboldt-Universität zu Berlin, Citation File Format

Ted Carnevale, Neuroscience Department, Yale University, ModelDB
Tom Morrell, Caltech Library, CaltechDATA
Tom Pollard, MIT/PhysioNet, PhysioNet

APPENDIX B: POLICY EXAMPLES

Scope statement

- Astrophysics Source Code Library. (n.d.). *Editorial policy*.
https://ascl.net/wordpress/submissions/editorial-policy/
- bio.tools. (n.d.). *Curators Guide*.
https://biotools.readthedocs.io/en/latest/curators_guide.html
- Caltech Library. (2017). *Terms of Deposit*.
https://data.caltech.edu/terms
- Caltech Library. (2019). *CaltechDATA FAQ*.
https://www.library.caltech.edu/caltechdata/faq
- Computational Infrastructure for Geodynamics. (n.d.). *Code Donation*.
https://geodynamics.org/cig/dev/code-donation/
- CoMSES Net Computational Model Library. (n.d.). *Frequently Asked Questions*.
https://www.comses.net/about/faq/#model-library
- ORNL DAAC for Biogeochemical Dynamics. (n.d.). *Data Scope and Acceptance Policy*.
https://daac.ornl.gov/submit/
- RDA Registry and Research Data Australia. (2018). *Collection*. ARDC Intranet.
https://intranet.ands.org.au/display/DOC/Collection
- Remote Sensing Code Library. (n.d.). *Submit*.
https://rscl-grss.org/submit.php
- SciCrunch. (n.d.). *Curation Guide for SciCrunch Registry*.
https://scicrunch.org/page/Curation%20Guidelines
- U.S. Department of Energy: Office of Scientific and Technical Information. (n.d.-a). *DOE CODE: Software Policy*. https://www.osti.gov/doecode/policy
- U.S. Department of Energy: Office of Scientific and Technical Information. (n.d.-b). *FAQs*. OSTI.GOV.
https://www.osti.gov/faqs

Guidance for users

- Astrophysics Source Code Library. (2021). *Q & A*
https://ascl.net/home/getwp/898

- bio.tools. (2021). *API Reference*
https://biotools.readthedocs.io/en/latest/api_reference.html
- Caltech Library. (2019). *CaltechDATA FAQ*.
https://www.library.caltech.edu/caltechdata/faq
- Harvard Dataverse. (n.d.). *Curation and Data Management Services*
https://support.dataverse.harvard.edu/curation-services
- OntoSoft. (n.d.). *An Intelligent Assistant for Software Publication*
https://ontosoft.org/users.html
- ORNL DAAC for Biogeochemical Dynamics. (n.d.). *Learning*
https://daac.ornl.gov/resources/learning/
- U.S. Department of Energy: Office of Scientific and Technical Information. (n.d.). *FAQs*. OSTI.GOV.
https://www.osti.gov/doecode/faq

Guidance for software contributors

- Astrophysics Source Code Library. (n.d.) *Submit a code*.
https://ascl.net/code/submit
- bio.tools. (n.d.) *Quick Start Guide*
https://biotools.readthedocs.io/en/latest/quickstart_guide.html
- Computational Infrastructure for Geodynamics. *Contributing Software*
https://geodynamics.org/cig/dev/code-donation/checklist/
- CoMSES Net Computational Model Library (2019) *Archiving your model: 1. Getting Started*
https://forum.comses.net/t/archiving-your-model-1-getting-started/7377
- Harvard Dataverse. (n.d.) *For Journals*.
https://support.dataverse.harvard.edu/journals

Authorship

- Committee on Publication Ethics: COPE. (2020a). *Authorship and contributorship*.
https://publicationethics.org/authorship
- Committee on Publication Ethics: COPE. (2020b). *Core practices*.
https://publicationethics.org/core-practices
- Dagstuhl EAS Specification Draft. (2016). *The Software Credit Ontology*.
https://dagstuhleas.github.io/SoftwareCreditRoles/doc/index-en.html#
- Journal of Open Source Software. (n.d.). *Ethics Guidelines*.
https://joss.theoj.org/about#ethics

- ORNL DAAC (n.d) *Authorship Policy*.
https://daac.ornl.gov/submit/
- PeerJ Journals. (n.d.-a). *Author Policies*.
https://peerj.com/about/policies-and-procedures/#author-policies
- PeerJ Journals. (n.d.-b). *Publication Ethics*.
https://peerj.com/about/policies-and-procedures/#publication-ethics
- PLOS ONE. (n.d.). *Authorship*.
https://journals.plos.org/plosone/s/authorship
- National Center for Data to Health. (2019). The Contributor Role Ontology.
https://github.com/data2health/contributor-role-ontology

Metadata schema

- ANDS: Australian National Data Service. (n.d.). *Metadata*. ANDS.
https://www.ands.org.au/working-with-data/metadata
- ANDS: Australian National Data Service. (2016). *ANDS Guide: Metadata*.
https://www.ands.org.au/data/assets/pdf_file/0004/728041/Metadata-Workinglevel.pdf
- Bernal, I. (2019). *Metadata for Data Repositories*.
https://doi.org/10.5281/zenodo.3233486
- bio.tools. (2020). *Bio-tools/biotoolsSchema* [HTML].
https://github.com/bio-tools/biotoolsSchema (Original work published 2015)
- bio.tools. (2019). *BiotoolsSchema documentation*.
https://biotoolsschema.readthedocs.io/en/latest/
- The CodeMeta crosswalks. (n.d.)
https://codemeta.github.io/crosswalk/
- Citation File Format (CFF). (n.d.)
https://doi.org/10.5281/zenodo.1003149
- The DataVerse Project. (2020). DataVerse 4.0+ Metadata Crosswalk.
https://docs.google.com/spreadsheets/d/10Luzti7svVTVKTA-px27oq3RxCUM-QbiTkm8iMd5C54
- OntoSoft. (2015). *OntoSoft Ontology*.
https://ontosoft.org/ontology/software/
- Zenodo. (n.d.-a). *Schema for Depositing*.
https://zenodo.org/schemas/records/record-v1.0.0.json
- Zenodo. (n.d.-b). *Schema for Published Record*.
https://zenodo.org/schemas/deposits/records/legacyrecord.json

Conditions of use policy

- Allen Institute. (n.d.). *Terms of Use*.
https://alleninstitute.org/legal/terms-use/
- Europeana. (n.d.). *Usage Guidelines for Metadata*. Europeana Collections.
https://www.europeana.eu/portal/en/rights/metadata.html
- U.S. Department of Energy: Office of Scientific and Technical Information. (n.d.). *DOE CODE FAQ: Are there restrictions on the use of the material in DOE CODE?*
https://www.osti.gov/doecode/faq#are-there-restrictions
- Zenodo. (n.d.). *Terms of Use*.
https://about.zenodo.org/terms/

Privacy policy

- Allen Institute. (n.d.). *Privacy Policy*.
https://alleninstitute.org/legal/privacy-policy/
- CoMSES Net. (n.d.). *Data Privacy Policy*.
https://www.comses.net/about/data-privacy/
- Nature. (2020). *Privacy Policy*.
https://www.nature.com/info/privacy
- Research Data Australia. (n.d.). *Privacy Policy*.
https://researchdata.ands.org.au/page/privacy
- SciCrunch. (2018). *Privacy Policy*. SciCrunch.
https://scicrunch.org/page/privacy
- Science Repository. (n.d.). *Privacy Policies*.
https://www.sciencerepository.org/privacy
- Zenodo. (n.d.). *Privacy policy*.
https://about.zenodo.org/privacy-policy/

Retention policy

- Bioconductor. (2020). *Package End of Life Policy*.
https://bioconductor.org/developers/package-end-of-life/
- Caltech Library. (n.d.). *CaltechDATA FAQ*.
https://www.library.caltech.edu/caltechdata/faq
- CoMSES Net Computational Model Library. (n.d.). *How long will models be stored in the Computational Model Library?*
https://www.comses.net/about/faq/

- Dryad. (2020). *Dryad FAQ - Publish and Preserve your Data*.
https://datadryad.org/stash/faq#preserved
- Software Heritage. (n.d.). *Content policy*.
https://www.softwareheritage.org/legal/content-policy/
- Zenodo. (n.d.). *General Policies v1.0*.
https://about.zenodo.org/policies/

End-of-life policy

- Figshare. (n.d.). *Preservation and Continuity of Access Policy*.
https://knowledge.figshare.com/articles/item/preservation-and-continuity-of-access-policy
- Open Science Framework. (2019). *FAQs*. OSF Guides.
http://help.osf.io/hc/en-us/articles/360019737894-FAQs
- NASA Earth Science Data Preservation Content Specification (n.d.)
https://earthdata.nasa.gov/esdis/eso/standards-and-references/preservation-content-spec
- Zenodo. (n.d.). *Frequently Asked Questions*.
https://help.zenodo.org/

APPENDIX C: RESOURCE INFORMATION

Since the first Task Force meeting was held in 2019, we have asked new resource representatives joining our community to provide the information shown in Table C.1. Thanks to this effort, the group has been able to learn about each resource, identify similarities and differences, and thus better inform our meeting discussions.

Tables C.2–C.4 provide an updated overview of the main features of all resources currently involved in the discussion and implementation of the best practices (30 resources in total as of December, 2021). Participating resources are diverse, and belong to a variety of discipline-specific (*e.g.*, neurosciences, biology, geosciences, *etc.*) and domain generic repositories. Curated resources tend to have a lower number of software entries. Most resources have been created in the last 20 years, with the oldest resource dating from 1991. Most resources accept a software deposit, support DOIs to identify their entries, are actively curated, and can be used to cite software.

Table C.1 Questions asked to resource representatives.

Question	Answer type
Repository name and abbreviation	Text
Repository home page	URL
Representative name and email address	Text
Is the repository discipline-specific?	Yes/No
Is the repository for discipline software only?	Yes/No
Is a software deposit accepted?	Yes/No
Is a software deposit required?	Yes/No
Does your resource have a public scope/editorial policy?	URL
Supported unique identifier(s) type(s)	Text
Can the repository mint DOIs?	Yes/No
Is the repository actively curated?	Yes/No/Other
How are entries added?	Text
Is your resource currently used to cite software?	Yes/No/Other
When did your resource start operating?	Year started
What is the number of records (as of filling date)?	Integer
Notes/comments/additional information	Text

Table C.2 Information shared by 30 resources participating in the SciCodes consortium, as of December 2021.

Question	#Yes	#No	#Other
Is the resource discipline-specific?	18	12	0
Does the resource accept software only?	17	13	0
Does the resource require a software deposit?	5	25	0
Does the resource accept a software deposit	22	8	0
Can the resource mint DOIs?	16	14	0
Is the resource actively curated?	21	3	6
Can the resource be used to cite software?	21	6	3

Table C.3 Number of entries described in the resources of the SciCodes consortium, by December 2021.

#Entries	#Resources
Unknown	2
Less than 1K	15
1K–100K	10
More than 100K	3

Table C.4 Date of creation of the resources in the SciCodes consortium, by December 2021.

Creation year	#Resources
Before 2000	6
2000–2010	9
After 2010	15

ACKNOWLEDGEMENTS

The best practices presented here were proposed and developed by a Task Force of the FORCE11 Software Citation Implementation Working Group. The following authors, randomly ordered, contributed equally to discussion, conceptualization, writing, reviewing, and editing this article: Daniel Garijo, Lorraine Hwang, Hervé Ménager, Alice Allen, Michael Hucka, Thomas Morrell, and Ana Trisovic.

Task Force on Best Practices for Software Registries participants: Alain Monteil, Alejandra Gonzalez-Beltran, Alexandros Ioannidis, Alice Allen, Allen Lee, Andre Jackson, Bryce Mecum,Caifan Du, Carly Robinson, Daniel Garijo, Daniel Katz, Genevieve Milliken, Hervé Ménager, Jurriaan Spaaks, Katrina Fenlon, Kristin Vanderbilt, Lorraine Hwang, Michael Hucka, Neil Chue Hong, P. Wesley Ryan, Peter Teuben, Shelley Stall, Stephan Druskat, Ted Carnevale, Thomas Morrell.

SciCodes Consortium participants: Alain Monteil, Alejandra Gonzalez-Beltran, Alexandros Ioannidis, Alice Allen, Allen Lee, Ana Trisovic, Anita Bandrowski, Bruce Wilson, Bryce Mecum, Carly Robinson, Celine Sarr, Colin Smith, Daniel Garijo, David Long, Harry Bhadeshia, Hervé Mé nager, Jeanette M. Sperhac, Joy Ku, Jurriaan Spaaks, Kristin Vanderbilt, Lorraine Hwang, Matt Jones, Mercé Crosas, Michael R. Crusoe, Mike Hucka, Ming Fang Wu, Morane Gruenpeter, Moritz Schubotz, Olaf Teschke, Pete Meyer, Peter Teuben, Piotr Sliz, Sara Studwell, Shelley Stall, Ted Carnevale, Tom Morrell, Tom Pollard, Wolfram Sperber.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was supported by the Alfred P. Sloan Foundation (Grant Number G-2019-12446), and the Heidelberg Institute of Theoretical Studies. Ana Trisovic is funded by the Alfred P. Sloan Foundation (Grant Number P-2020-13988). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Alfred P. Sloan Foundation: G-2019-12446 and P-2020-13988.
Heidelberg Institute of Theoretical Studies.

Competing Interests

The authors declare that they have no competing interests.

Author Contributions

- Daniel Garijo conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Hervé Ménager conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Lorraine Hwang conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Ana Trisovic conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Michael Hucka conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Thomas Morrell conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Alice Allen conceived and designed the experiments, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

There is no data or code associated with this publication.

REFERENCES

Allen A, DuPrie K, Berriman B, Hanisch RJ, Mink J, Teuben PJ. 2013. Astrophysics source code library. *Astronomical Data Analysis Software and Systems XXII* **475**:387.

Allen L, O'Connell A, Kiermer V. 2019. How can we ensure visibility and diversity in research contributions? How the contributor role taxonomy (credit) is helping the shift from authorship to contributorship. *Learned Publishing* **32(1)**:71–74 DOI 10.1002/leap.1210.

Allen A, Schmidt J. 2015. Looking before leaping: creating a software registry. *Journal of Open Research Software* **3(1)**:e15 DOI 10.5334/jors.bv.

Alliez P, Di Cosmo R, Guedj B, Girault A, Hacid M-S, Legrand A, Rougier NP. 2019. Attributing and referencing (research) software: best practices and outlook from inria. *Computing in Science and Engineering* **22(1)**:1–14 DOI 10.1109/MCSE.2019.2949413.

Australian Research Council. 2018. ARC open access policy. *Available at https://www.arc.gov.au/policies-strategies/policy/arc-open-access-policy*.

Baker M. 2016. 1,500 scientists lift the lid on reproducibility. *Nature News* **533(7604)**:452–454 DOI 10.1038/533452a.

Barnes N. 2010. Publish your computer code: it is good enough. *Nature* **467(7317)**:753 DOI 10.1038/467753a.

Baruch P. 2007. Open access developments in France: the HAL open archives system. *Learned Publishing* **20(4)**:267–282 DOI 10.1087/095315107X239636.

Berman F, Crosas M. 2020. The research data alliance: benefits and challenges of building a community organization. *Harvard Data Science Review* **2(1)** DOI 10.1162/99608f92.5e126552.

Bourne PE, Clark TW, Dale R, de Waard A, Herman I, Hovy EH, Shotton D. 2012. Improving the future of research communications and e-scholarship (Dagstuhl Perspectives Workshop 11331). *Dagstuhl Manifestos* **1(1)**:41–60 DOI 10.4230/DagMan.1.1.41.

Brinckman A, Chard K, Gaffney N, Hategan M, Jones MB, Kowalik K, Kulasekaran S, Ludäscher B, Mecum BD, Nabrzyski J, Stodden V, Taylor IJ, Turk MJ, Turner K. 2019. Computing environments for reproducibility: capturing the "Whole Tale". *Future Generation Computer Systems* **94**:854–867 DOI 10.1016/j.future.2017.12.029.

CERN and OpenAIRE. 2013. Zenodo. *Available at https://doi.org/10.25495/7gxk-rd71*.

Chen X, Dallmeier-Tiessen S, Dasler R, Feger S, Fokianos P, Gonzalez JB, Hirvonsalo H, Kousidis D, Lavasa A, Mele S, Rodriguez DR, Šimko T, Smith T, Trisovic A, Trzcinska A, Tsanaktsidis I, Zimmermann M, Cranmer K, Heinrich L, Watts G, Hildreth M, Lloret Iglesias L, Lassila-Perini K, Neubert S. 2019. Open is not enough. *Nature Physics* **15(2)**:113–119 DOI 10.1038/s41567-018-0342-2.

Chue Hong NP, Katz DS, Barker M, Lamprecht A-L, Martinez C, Psomopoulos FE, Harrow J, Castro LJ, Gruenpeter M, Martinez PA, Honeyman T. 2021. FAIR principles for research software (FAIR4RS principles). *Research Data Alliance* **3(1)**:37–59 DOI 10.3233/DS-190026.

Clyburne-Sherin A, Fei X, Green SA. 2019. Computational reproducibility via containers in psychology. *Meta-Psychology* **3**:892 DOI 10.15626/MP.2018.892.

CoreTrustSeal. 2019. CoreTrustSeal trustworthy data repositories requirements 2020–2022. *Available at https://www.coretrustseal.org/why-certification/requirements/*.

Dashnow H, Lonsdale A, Bourne PE. 2014. Ten simple rules for writing a PLOS ten simple rules article. *PLOS Computational Biology* **10(10)**:1–5 DOI 10.1371/journal.pcbi.1003858.

Di Cosmo R, Zacchiroli S. 2017. Software heritage: why and how to preserve software source code. In: *iPRES 2017 - 14th International Conference on Digital Preservation*. Kyoto, Japan, 1–10.

Directorate-General for Research and Innovation (European Commission). 2018. *Turning FAIR into reality: final report and action plan from the European Commission expert group on FAIR data*. Luxembourg: Publications Office of the European Union.

Du C, Cohoon J, Priem J, Piwowar H, Meyer C, Howison J. 2021. Citeas: better software through sociotechnical change for better software citation. In: *CSCW '21: Companion Publication of the 2021 Conference on Computer Supported Cooperative Work and Social Computing*.

Editorial Staff. 2019. Giving software its due. *Nature Methods* **16(3)**:207 DOI 10.1038/s41592-019-0350-x.

Ensor N, Stooksbury S, Smith A, Johnson LA, Vowell L, Martin M, Hensley M, Finkbeiner D, Robinson C, Knight K, Nelson J, Davis L, Lee I, Sherline C, Welsch T, Billings JJ, West MB, Sowers T, Watson A. 2017. Doe code. *Available at https://doi.org/10.11578/dc.20171031.3*.

Fox P, Erdmann C, Stall S, Griffies SM, Beal LM, Pinardi N, Hanson B, Friedrichs MAM, Feakins S, Bracco A, Pirenne B, Legg S. 2021. Data and Software Sharing Guidance for Authors Submitting to AGU Journals. DOI 10.5281/zenodo.5124741.

Frank RD, Chen Z, Crawford E, Suzuka K, Yakel E. 2017. Trust in qualitative data repositories. *Proceedings of the Association for Information Science and Technology* **54(1)**:102–111 DOI 10.1002/pra2.2017.14505401012.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang J. 2004. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology* **5(10)**:1–16 DOI 10.1186/gb-2004-5-10-r80.

Gil Y, Garijo D, Mishra S, Ratnakar V. 2016. OntoSoft: a distributed semantic registry for scientific software. In: *2016 IEEE 12th International Conference on e-Science (e-Science)*. Piscataway: IEEE, 331–336.

Gil Y, Ratnakar V, Garijo D. 2015. OntoSoft: capturing scientific software metadata. In: *K-CAP 2015: Proceedings of the 8th International Conference on Knowledge*. ACM Press, 1–4.

Grethe JS, Bandrowski A, Banks DE, Condit C, Gupta A, Larson SD, Li Y, Ozyurt IB, Stagg AM, Whetzel PL, Marenco L, Miller P, Wang R, Shepherd GM, Martone ME. 2014. SciCrunch: a cooperative and collaborative data and resource discovery platform for scientific communities. *Neuroinformatics* **8**:e00069 DOI 10.3389/conf.fninf.2014.18.00069.

Greuel G-M, Sperber W. 2014. swMATH—an information service for mathematical software. In: Hong H, Yap C, eds. *Mathematical Software—ICMS 2014*. Berlin, Heidelberg: Springer, 691–701.

Grosbol P, Tody D. 2010. Making access to astronomical software more efficient. *ArXiv preprint*. DOI 10.48550/arXiv.1004.4430.

Guha RV, Brickley D, Macbeth S. 2016. Schema.org: evolution of structured data on the web. *Communications of the ACM* **59(2)**:44–51 DOI 10.1145/2844544.

Hettrick S. 2018. Software in research survey. Zenodo. *Available at https://doi.org/10.5281/zenodo.1183562*.

Howison J, Herbsleb JD. 2011. Scientific software production: incentives and collaboration. In: *CSCW '11: Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*. New York, NY, USA: Association for Computing Machinery, 513–522.

Hwang L, Kellogg LH. 2017. CIG community standards and best practices for scientific software (Invited). SSA 2017 annual meeting announcement and program. *Seismological Research Letters* **88(2B)**:463–723 DOI 10.1785/0220170035.

Ince DC, Hatton L, Graham-Cumming J. 2012. The case for open computer programs. *Nature* **482(7386)**:485–488 DOI 10.1038/nature10836.

Ison J, Ienasescu H, Chmura P, Rydza E, Ménager H, Kalaš M, Schwämmle V, Grüning B, Beard N, Lopez R, Duvaud S, Stockinger H, Persson B, Vařeková RS, Raček T, Vondrášek J, Peterson H, Salumets A, Jonassen I, Hooft R, Nyrönen T, Valencia A, Capella S, Gelpí J, Zambelli F, Savakis B, Leskošek B, Rapacki K, Blanchet C, Jimenez R, Oliveira A, Vriend G, Collin O, van Helden J, Løngreen P, Brunak S. 2019. The bio.tools registry of software tools and data resources for the life sciences. *Genome Biology* **20(1)**:1–4 DOI 10.1186/s13059-019-1772-6.

Ison J, Ienasescu H, Rydza E, Chmura P, Rapacki K, Gaignard A, Schwämmle V, van Helden J, Kalaš M, Ménager H. 2021. biotoolsSchema: a formalized schema for bioinformatics software description. *GigaScience* **10(1)**:giaa157 DOI 10.1093/gigascience/giaa157.

Janssen MA, Alessa LN, Barton M, Bergin S, Lee A. 2008. Towards a community framework for agent-based modelling. *Journal of Artificial Societies and Social Simulation* **11(2)**:6.

Jiménez RC, Kuzak M, Alhamdoosh M, Barker M, Batut B, Borg M, Capella-Gutierrez S, Chue Hong N, Cook M, Corpas M, Flannery M, Garcia L, Gelpí JL, Gladman S, Goble C, González Ferreiro M, Gonzalez-Beltran A, Griffin PC, Grüning B, Hagberg J, Holub P, Hooft R, Ison J, Katz DS, Leskošek B, López Gómez F, Oliveira LJ, Mellor D, Mosbergen R, Mulder N, Perez-Riverol Y, Pergl R, Pichler H, Pope B, Sanz F, Schneider MV, Stodden V, Suchecki R, Svobodová Vařeková R, Talvik H-A, Todorov I, Treloar A, Tyagi S, van Gompel M, Vaughan D, Via A, Wang X, Watson-Haigh NS, Crouch S. 2017. Four simple recommendations to encourage best practices in research software. *F1000Research* **6**:876 DOI 10.12688/f1000research.

Jones MB, Boettiger C, Mayes AC, Slaughter P, Gil Y, Chue Hong N, Goble C. 2017. CodeMeta. *Available at https://github.com/codemeta/codemeta*.

Katz DS, Gruenpeter M, Honeyman T. 2021. Taking a fresh look at FAIR for research software. *F1000Research* **2(3)**:100222 DOI 10.1016/j.patter.2021.100222.

Katz DS, Niemeyer KE, Smith AM. 2018. Publish your software: introducing the journal of open source software (JOSS). *Computing in Science Engineering* **20(3)**:84–88 DOI 10.1109/MCSE.2018.03221930.

Katz DS, Niemeyer KE, Smith AM, Anderson WL, Boettiger C, Hinsen K, Hooft R, Hucka M, Lee A, Löffler F, Pollard T, Rios F. 2016. Software vs. data in the context of citation. *PeerJ Preprints* **4**:e2630v1 DOI 10.7287/peerj.preprints.2630v1.

Kern F, Fehlmann T, Keller A. 2020. On the lifetime of bioinformatics web services. *Nucleic Acids Research* **48(22)**:12523–12533 DOI 10.1093/nar/gkaa1125.

Lamprecht A-L, Garcia L, Kuzak M, Martinez C, Arcila R, Martin Del Pico E, Dominguez Del Angel V, van de Sandt S, Ison J, Martinez PA, McQuilton P, Valencia A, Harrow J, Psomopoulos F, Gelpi JL, Chue Hong N, Goble C, Capella-Gutierrez S. 2020. Towards FAIR principles for research software. *Data Science* **3(1)**:37–59 DOI 10.3233/DS-190026.

Lin D, Crabtree J, Dillo I, Downs RR, Edmunds R, Giaretta D, De Giusti M, L'Hours H, Hugo W, Jenkyns R, Khodiyar V, Martone ME, Mokrane M, Navale V, Petters J, Sierman B, Sokolova DV, Stockhause M, Westbrook J. 2020. The TRUST principles for digital repositories. *Scientific Data* **7(1)**:144 DOI 10.1038/s41597-020-0486-7.

Merali Z. 2010. Computational science: …error. *Nature* **467(7317)**:775–777 DOI 10.1038/467775a.

Ministère de l'Enseignement supérieur, de la Recherche et de l'Innovation. 2021. Second national plan for open science. *Available at https://www.ouvrirlascience.fr/second-national-plan-for-open-science/*.

Momcheva I, Tollerud E. 2015. Software use in astronomy: an informal survey. *ArXiv preprint*. DOI 10.48550/arXiv.1507.03989.

Morin A, Urban J, Adams PD, Foster I, Sali A, Baker D, Sliz P. 2012. Shining light into black boxes. *Science* **336(6078)**:159–160 DOI 10.1126/science.1218263.

Office of Science and Technology Policy. 2016. Principles for promoting access to federal government-supported scientific data and research findings through international scientific cooperation. *Available at https://www.nesdisia.noaa.gov/docs/iwgodsp_principles_0.pdf*.

ORNL. 2013. Oak ridge national laboratory distributed active archive center. *Available at https://daac.ornl.gov/*.

Peckham SD, Hutton EWH, Norris B. 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. *Computers and Geosciences* **53(December (4))**:3–12 DOI 10.1016/j.cageo.2012.04.002.

Peng RD. 2011. Reproducible research in computational science. *Science* **334(6060)**:1226–1227 DOI 10.1126/science.1213847.

Serban A, van der Blom K, Hoos H, Visser J. 2020. Adoption and effects of software engineering best practices in machine learning. In: *ESEM '20: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* New York: Association for Computing Machinery.

Smith AM, Katz DS, Niemeyer KE. 2016. Software citation principles. *PeerJ Computer Science* **2(2)**:e86 DOI 10.7717/peerj-cs.86.

Soito L, Hwang LJ. 2017. Citations for software: providing identification, access and recognition for research software. *International Journal of Digital Curation* **11(2)**:48–63 DOI 10.2218/ijdc.v11i2.390.

Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, Heroux MA, Ioannidis JPA, Taufer M. 2016. Enhancing reproducibility for computational methods. *Science* **354(6317)**:1240–1241 DOI 10.1126/science.aah6168.

Task Force on Best Practices for Software Registries, Monteil A, Gonzalez-Beltran A, Ioannidis A, Allen A, Lee A, Bandrowski A, Wilson BE, Mecum B, Du CF, Robinson C, Garijo D, Katz DS, Long D, Milliken G, Ménager H, Hausman J, Spaaks JH, Fenlon K, Vanderbilt K, Hwang L, Davis L, Fenner M, Crusoe MR, Hucka M, Wu M, Chue Hong N, Teuben P, Stall S, Druskat S, Carnevale T, Morrell T. 2020. Nine best practices for research software registries and repositories: a concise guide. *ArXiv preprint*. DOI 10.48550/arXiv.2012.13117.

Thelwall M, Kousha K. 2016. Figshare: a universal repository for academic resource sharing? *Online Information Review* **40(3)**:333–346 DOI 10.1108/OIR-06-2015-0190.

Trisovic A, Durbin P, Schlatter T, Durand G, Barbosa S, Brooke D, Crosas M. 2020. Advancing computational reproducibility in the dataverse data repository platform. In: *P-RECS '20: Proceedings of the 3rd International Workshop on Practical Reproducible Evaluation of Computer Systems*. 15–20.

Trisovic A, Lau MK, Pasquier T, Crosas M. 2022. A large-scale study on research code quality and execution. *Scientific Data* **9(1)**:60 DOI 10.1038/s41597-022-01143-6.

Veretnik S, Fink JL, Bourne PE. 2008. Computational biology resources lack persistence and usability. *PLOS Computational Biology* **4(7)**:e1000136 DOI 10.1371/journal.pcbi.1000136.

Weiner B, Blanton MR, Coil AL, Cooper MC, Davé R, Hogg DW, Holden BP, Jonsson P, Kassin SA, Lotz JM, Moustakas J, Newman JA, Prochaska JX, Teuben PJ, Tremonti CA, Willmer CNA. 2009. Astronomical software wants to be free: a manifesto. In: *Astro2010: The Astronomy and Astrophysics Decadal Survey*. Vol. 2010. P61.

Wilkinson MD, Dumontier M, Aalbersberg IJJ, Appleton G, Axton M, Baak A, Blomberg N, Boiten J-W, da Silva SLB, Bourne PE, Bouwman J, Brookes AJ, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo CT, Finkers R, Gonzalez-Beltran A, Gray AJG, Groth P, Goble C, Grethe JS, Heringa J, 't Hoen PAC, Hooft R, Kuhn T, Kok R, Kok J, Lusher SJ, Martone ME, Mons A, Packer AL, Persson B, Rocca-Serra P, Roos M, van Schaik R, Sansone S-A, Schultes E, Sengstag T, Slater T, Strawn G, Swertz MA, Thompson M, van der LJ, van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft K, Zhao J, Mons B. 2016. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data* **3(1)**:160018 DOI 10.1038/sdata.2016.18.

Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, Haddock SHD, Huff KD, Mitchell IM, Plumbley MD, Waugh B, White EP, Wilson P. 2014. Best practices for scientific computing. *PLOS Biology* **12(1)**:e1001745 DOI 10.1371/journal.pbio.1001745.

Yakel E, Faniel IM, Kriesberg A, Yoon A. 2013. Trust in digital repositories. *International Journal of Digital Curation* **8(1)**:143–156 DOI 10.2218/ijdc.v8i1.251.

A survey of researchers' code sharing and code reuse practices, and assessment of interactive notebook prototypes

Lauren Cadwallader and Iain Hrynaszkiewicz

Public Library of Science, Cambridge, United Kingdom

ABSTRACT

This research aimed to understand the needs and habits of researchers in relation to code sharing and reuse; gather feedback on prototype code notebooks created by NeuroLibre; and help determine strategies that publishers could use to increase code sharing. We surveyed 188 researchers in computational biology. Respondents were asked about how often and why they look at code, which methods of accessing code they find useful and why, what aspects of code sharing are important to them, and how satisfied they are with their ability to complete these tasks. Respondents were asked to look at a prototype code notebook and give feedback on its features. Respondents were also asked how much time they spent preparing code and if they would be willing to increase this to use a code sharing tool, such as a notebook. As a reader of research articles the most common reason (70%) for looking at code was to gain a better understanding of the article. The most commonly encountered method for code sharing–linking articles to a code repository–was also the most useful method of accessing code from the reader's perspective. As authors, the respondents were largely satisfied with their ability to carry out tasks related to code sharing. The most important of these tasks were ensuring that the code was running in the correct environment, and sharing code with good documentation. The average researcher, according to our results, is unwilling to incur additional costs (in time, effort or expenditure) that are currently needed to use code sharing tools alongside a publication. We infer this means we need different models for funding and producing interactive or executable research outputs if they are to reach a large number of researchers. For the purpose of increasing the amount of code shared by authors, *PLOS Computational Biology* is, as a result, focusing on policy rather than tools.

Subjects Bioinformatics, Computational Biology, Science Policy, Computational Science

Keywords Open science, Publishing practices, Research code dissemination, Research code reuse, Research code sharing, Survey results

Submitted 7 June 2022
Accepted 1 August 2022
Published 22 August 2022

Corresponding author
Lauren Cadwallader,
lcadwallader@plos.org

Academic editor
Björn Brembs

Additional Information and Declarations can be found on page 18

DOI **10.7717/peerj.13933**

INTRODUCTION

Code sharing requirements of journals and funders are increasing but are not as prevalent as requirements for sharing other research outputs, such as research data. Software tools, such as code notebooks, can facilitate code sharing in a way that reduces barriers to computational reproducibility but are not necessarily cost (*e.g.*, time) free to authors. Some publishers have experimented with executable code and interactive features in their articles. Policies can also be employed to increase the amount of code shared alongside

published articles. Researchers working in fields such as computational biology generate code for a large proportion of their studies (*Hrynaszkiewicz, Harney & Cadwallader, 2021a*; *Hrynaszkiewicz, Harney & Cadwallader, 2021b*). Sharing code improves reproducibility, especially when made available before publication (*Fernández-Juricic, 2021*). Lack of source code—along with raw data, and protocols—has been described as the main barrier to computational reproducibility of published research (*Seibold et al., 2021*). However, technical and cultural barriers to computational reproducibility have been identified in the literature (*Samota & Davey, 2021*; *Hrynaszkiewicz, Harney & Cadwallader, 2021a*; *Van den Eynden et al., 2016*). These barriers include insufficient time, funds and skills to prepare code for sharing. A desire to protect intellectual property (IP) is also reported as a common or important barrier to code sharing.

Journals and publishers must understand and respond to these challenges in the research communities they serve if they wish to support open, reproducible research, and test and implement solutions. Introducing policies is an important way for journals to increase awareness and adoption of research practices that are important to a particular community, as demonstrated by the increase in research data sharing policies and practices in the last decade (*Hrynaszkiewicz, 2020*). In 2021, *PLOS Computational Biology* introduced a strengthened, mandatory code sharing policy in response to a desire of this community to support reproducibility by increasing the availability of code associated with articles published in the journal (*Cadwallader et al., 2021*). The introduction of this policy was supported by the results of a survey of the computational biology community, which demonstrated their support for a mandatory code sharing policy in *PLOS Computational Biology* (*Hrynaszkiewicz, Harney & Cadwallader, 2021a*). The survey results also found that code sharing and access are important to researchers, and that they are satisfied with their ability to share their own code, but they are not satisfied with their ability to access other researchers' code. Following the Jobs To Be Done theory (*Ulwick & Osterwalder, 2016*), this finding implies that there may be opportunities for new solutions (which could be products, policies, services or features) that support researchers in accessing other researchers' code.

Numerous technical solutions (tools) exist that could play a role in improving code availability, and reuse. Scholarly publishers and tool providers have experimented with interactive and reproducible articles for years (*Akhlaghi et al., 2021*). Such tools inherently require availability of code and data to enable interactivity with and reuse of results. An example of this is the journal, eLife, and reproducible document platform, Stencila, who have collaborated to experiment with publication of Executable Research Articles (ERA; *Tsang & Maciocci, 2020*). Other tools that support code sharing and reuse alongside scholarly articles include commercial platforms such as Code Ocean, which provides executable code capsules; Gigantum, and NextJournal (*Perkel, 2019*) and collaborative, interactive code notebooks such as Observable (*Perkel, 2021*). For a review of infrastructures that support computational reproducibility see *Konkol, Nüst & Goulier (2020)*. Many code notebook tools are built on open source technology, such as Jupyter and MyBinder, and researcher-led efforts to produce code notebook type outputs often use these (*Lasser, 2020*). One relatively new code notebook initiative, NeuroLibre, supported by the Canadian Open Neuroscience Platform, is an open access platform hosting notebooks derived from

published or preprinted research articles that can be freely modified and re-executed (*Boudreau et al., 2021*).

The potential benefits of these tools—for researchers as readers and authors, for publishers, and the accessibility of science—are numerous. Our focus was on how these tools meet researcher needs for code sharing and reuse, as these needs align with PLOS' goals to increase the adoption, and benefits, of open science. But the extent to which these tools do meet these needs is unclear from the available literature. Furthermore, the adoption of new tools or workflows for preparing and sharing code would incur costs, in terms of time and effort, for researchers (as authors, readers, editors and peer reviewers) and publishers. For new tools to be widely adopted it is important to understand if additional effort required to adopt new tools is acceptable to their users. As a publisher PLOS experiments with solutions that support open science in different communities, and partners with community resources, such as data repositories and preprint servers, to achieve this. To this end, rather than creating new solutions, PLOS partnered with NeuroLibre to learn more about the value of their interactive code notebooks and research publications to readers and authors. The results were anticipated to:

- Provide a deeper understanding of how researchers share and interact with code
- Inform *PLOS Computational Biology*'s plans for further supporting code sharing and reuse, beyond its mandatory code sharing policy
- Inform development of NeuroLibre with quantifiable feedback from potential users of the tool on the tool itself and researcher needs that are related to the features of the tool
- Provide PLOS, and other publishers, with quantitative insights on researchers' attitudes and experience with interactive article features, to inform future publishing innovation approaches.

METHODS

We created a survey in English in Alchemer and distributed it in February and March 2021. The survey had three main purposes:

(1) Understand how researchers interact with code as readers of articles
(2) Gather feedback on the prototype NeuroLibre notebook version of *PLOS Computational Biology* articles
(3) Gain a more detailed understanding of researchers' abilities to carry out code sharing tasks, how they rate the importance of these tasks and how satisfied they are with their ability to complete the tasks

The survey was promoted with an accompanying blog (*Cadwallader, 2021*) and email campaign, which was sent to previous PLOS authors and other PLOS registered users in computational biology related disciplines ($n = 23,272$). The survey (*Cadwallader, Hrynaszkiewicz & Harney, 2022*) was launched with the blog on the 11th February 2021 and the email campaign followed on the 19th February. The results were exported from Alchemer on 25th March 2021.

The survey methodology was adapted from our group's previous recent work (described in *Hrynaszkiewicz, Harney & Cadwallader, 2021b*). Briefly, respondents were asked to

answer a series of questions from the perspective of both readers and authors of articles with associated code. To identify if there were opportunities to support researchers with sharing code using new solutions, we asked respondents to rate various code sharing and reuse factors in terms of how important they were to them and how satisfied they were with their ability to complete them. These responses were converted to numerical scores and used to calculate opportunity scores for each factor using the following equation:

$$\text{Opportunity score} = \text{Mean importance} * (1 - \text{mean satisfaction}/100).$$

Opportunity scores above 25 indicate "better than neutral" or marginal opportunities and scores above 36 we regard as good opportunities. This approach is more nuanced than simply using quadrants and looking for high importance/low satisfaction scores.

In addition, NeuroLibre created two prototype interactive notebook versions (*Larremore, 2019*; *Tampuu et al., 2019*) of articles published in *PLOS Computational Biology* (*Larremore, 2019*; *Tampuu et al., 2019*), so they could be shared with the community and their feedback sought on the value and features of the interactive format. Survey respondents were asked to give feedback on one of these prototypes.

Ethical considerations

Approval from a research ethics committee was not sought as we considered the research to be low risk. Sensitive information about the participants was not collected and all data were collected anonymously. Participants were informed that participation was voluntary, and that they were free to withdraw at any time until they submitted their response. The results were only analyzed in aggregate and answers were never associated with individual participants. The data collection procedures and survey tool are compliant with the General Data Protection Regulation 2016/679.

RESULTS

Respondent demographics

The survey received a total of 188 complete responses, with an additional 39 partial responses (some but not all questions answered) and 175 incomplete responses (some but not all demographic questions answered only). 79% of the respondents clicked through from the email campaign link ($n = 316$), which had a 1.4% engagement (click) rate. This analysis will focus on the 188 complete responses.

A range of disciplines are represented by the respondents, with a third of respondents being from the computational biology field (Table 1). For those who chose 'Other', 13 out of 14 respondents were in STEM fields, with math-related fields being most commonly specified ($n = 6$). One individual was from a social sciences discipline.

Responses are skewed more towards researchers with fewer publications, (Fig. 1). Respondents were overwhelmingly from Europe (46%) or North America (40%), with very few respondents indicating their location in other geographic regions (Table 2). 54% of respondents had previously published in *PLOS Computational Biology*.

Table 1 Disciplinary distribution of respondents who completed the survey ($n = 188$).

Research field	*n*	% of total	Research field	*n*	% of total
Computational Biology	63	34%	Engineering and Technology	9	5%
Biology and Life Sciences	34	18%	Physical sciences	10	5%
Bioinformatics	32	17%	Ecology and Environmental Sciences	5	3%
Medicine and Health Sciences	18	10%	Social sciences	3	2%
Other (please specify)	14	7%			

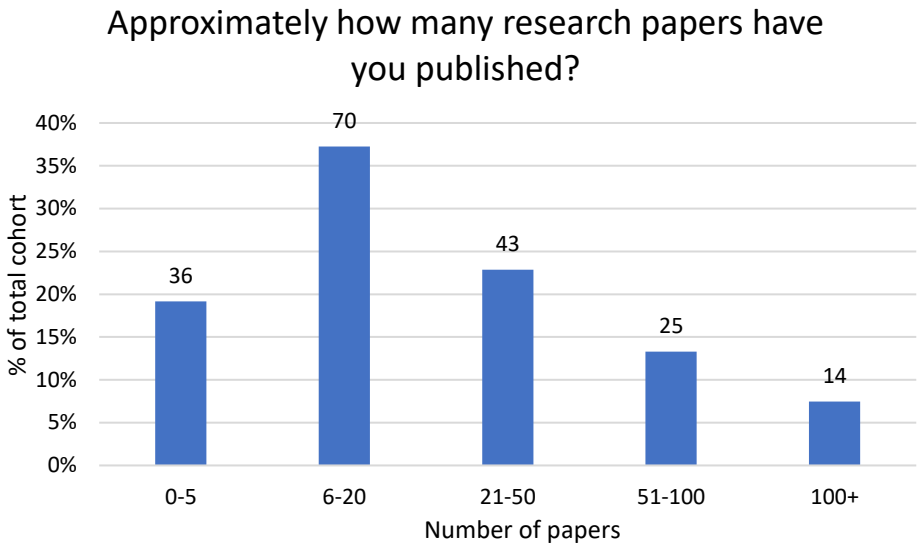


Figure 1 **Distribution of respondents ($n = 188$) according to the number of previously published papers.** The number of respondents in each category is given above the bar.
Full-size DOI: 10.7717/peerj.13933/fig-1

Table 2 Geographical distribution of respondents ($n = 188$).

Region	*n*	% of total cohort
Europe	87	46%
North America	75	40%
Asia	12	6%
South America	7	4%
Australasia	4	2%
Africa	2	1%
Middle East	1	1%

When and why researchers access or read code

Respondents were asked to answer a set of questions from the viewpoint of a reader of research articles that had associated code to understand how they interacted with code in this setting. Three-quarters ($n = 141$) of the respondents look at code associated with a

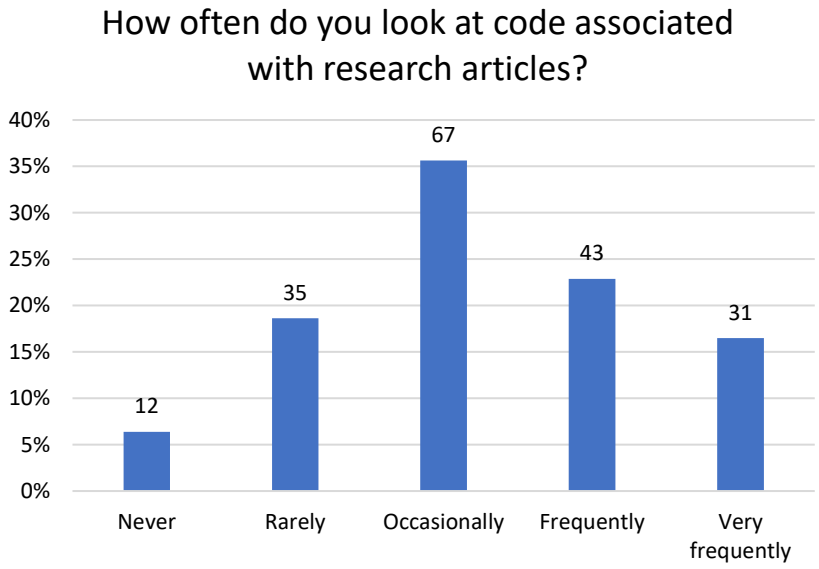


Figure 2 Frequency with which respondents look at code associated with research articles. The number of respondents is given above each bar.

Full-size DOI: 10.7717/peerj.13933/fig-2

research paper at least occasionally, with 39% ($n = 74$) looking at code frequently or very frequently. Only 6% ($n = 12$) said they never looked at the associated code (Fig. 2).

The degree to which readers from different disciplines look at code associated with research articles is variable, although many of the cohorts included in the survey results are small (Fig. 3). Of the largest cohorts surveyed, those in the Biology and Life Sciences look at code associated with articles less frequently than in Computational Biology and Bioinformatics. Lower levels of looking at code are also seen in the Medicine and Health Sciences cohort although this is a smaller group ($n = 18$).

Respondents were asked why they look at code associated with published articles. Free text answers were provided by 178 respondents. Answers were categorised to identify general trends, with the majority of respondents ($n = 100$) giving two or more reasons for looking at the code.

- 125 (70%) respondents look at code to aid their understanding of the article. For example, 113 respondents (63%) specified that they wish to directly verify the code or examine its use in the context of the research presented and 38 respondents (21%) look at the code to better understand the methods described in the article, *e.g.*, what parameters were selected.
- 86 (48%) respondents gave answers that fell into the 'reuse' category, *e.g.*, directly reusing the code (62 responses/35%) and reusing selected parts of the code (27 responses/15%). Other reuse reasons were using the code as an example in teaching (one response), as a comparison to the reader's own code (six response/3%) and to reuse the data (one response).
- Respondents also looked at the code to assess the quality of the research (37 respondents/21%), giving reasons such as to check for minimal standards (eight

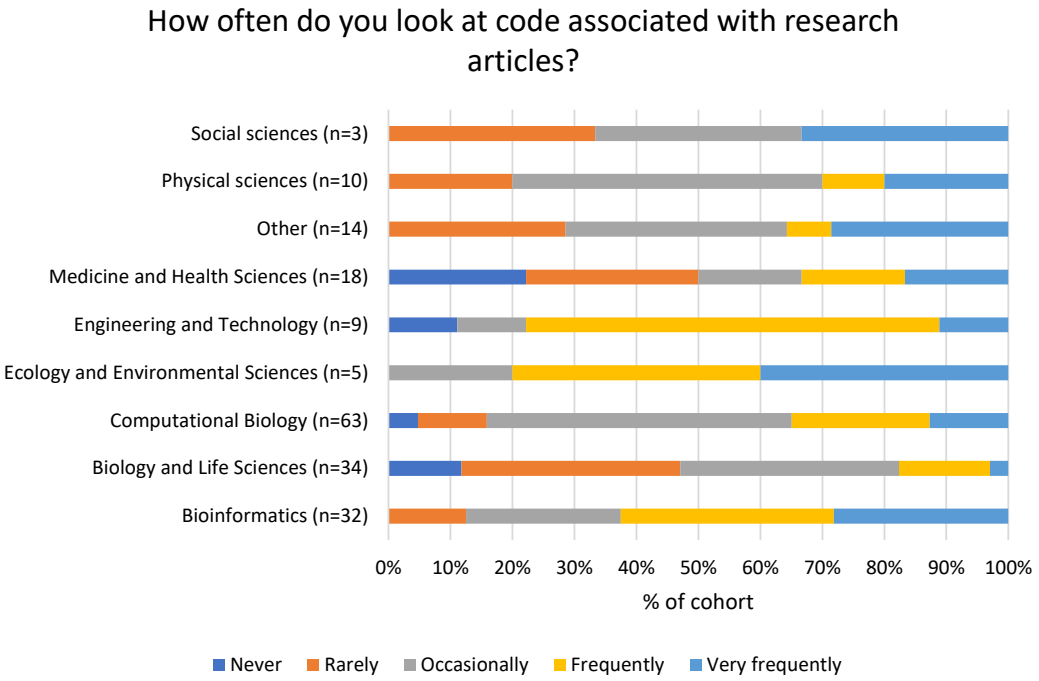


Figure 3 **Frequency at which authors look at code associated with research papers according to discipline.**

Full-size DOI: 10.7717/peerj.13933/fig-3

responses/4%), for trust or transparency reasons (five responses/3%) and replicate the analysis using their own data (21 responses/12%).

- Reasons linked to discovery were also given by five respondents (3%), for example finding new GitHub repositories of interest and looking for novel code.

The usefulness of methods for accessing or reading code

Respondents were asked how useful they found various methods of accessing code associated with a research article, when considering the 6 months before they completed the survey. Not all respondents had encountered the methods specified. Using a 'Link to a code repository' was the most common method (encountered by 98%), followed by 'link to a website' (88%) and 'available on request' (87%) (Fig. 4). A link to archived code, that is, a snapshot of code deposited in a generalist repository was encountered by 72% of respondents. Links to code notebooks were encountered by 66% and executable code capsules by 40%. The methods were not defined for respondents, although they had been asked to look at a prototype notebook before answering the questions.

'Link to code repository' was rated as the most useful method –both in terms of the number of respondents who rated it 'extremely' or 'very useful', and the number who rated it as 'not at all useful' (Fig. 5). Accessing code that is 'available on request' was rated as least useful (based on number of 'not at all useful').

The five-point unipolar scale used in this question can be mapped to a value from 0 to 100, with 0 equalling 'not at all useful' and 100 equalling 'extremely useful'. 'I have not

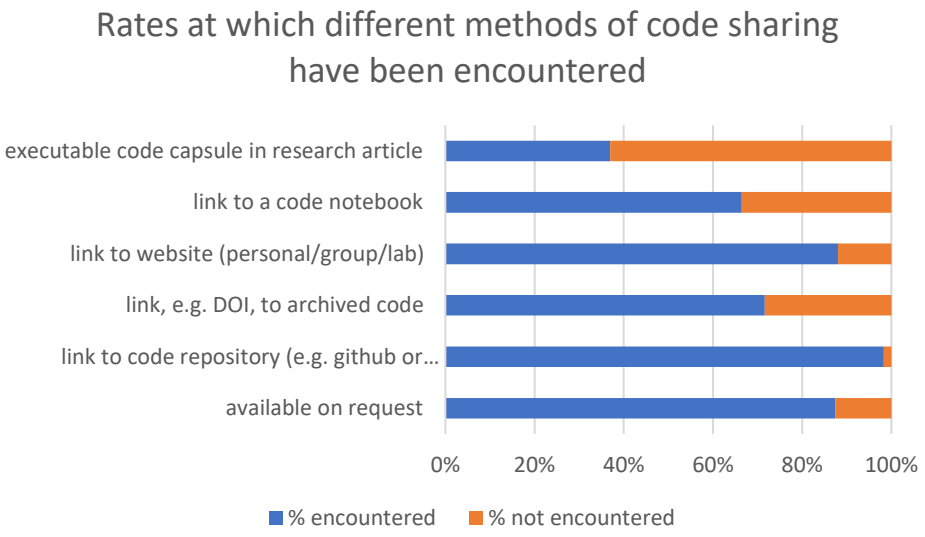


Figure 4 **Rates that the various methods of code sharing have been encountered by the respondents.**
Full-size DOI: 10.7717/peerj.13933/fig-4

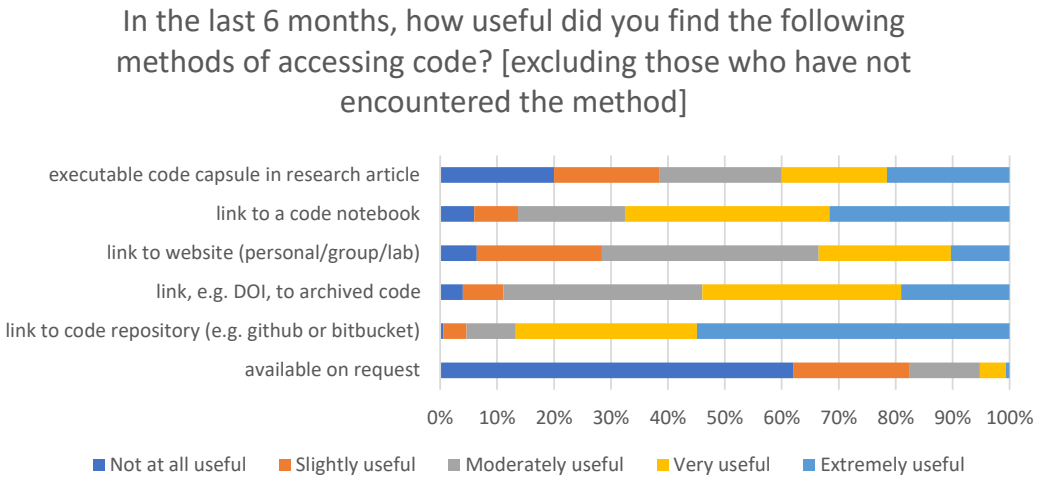


Figure 5 **The usefulness of different code sharing methods as percentages of the respondents who had encountered each method.**
Full-size DOI: 10.7717/peerj.13933/fig-5

encountered this method of sharing' responses were not scored. Taking the mean rating for all the methods (Fig. 6), the most commonly encountered method (link to a code repository), is also the most useful (mean 84.1 ± 3.2 (95% CI)). The mean scores given were: code notebooks (69.9 ± 5.3 (95% CI)); link to archived code (64.5 ± 4.4 (95% CI)); link to website (52.3 ± 4.2 (95% CI)); and executable code capsules (50.8 ± 8.9 (95% CI)). The 95% confidence intervals for code capsules and link to a website (41.9–59.7 and 48.1–56.5 respectively) do not overlap those for code notebooks and archived code (64.6–75.1 and 60.0–68.9 respectively).

[1] GitHub was the most highly named code repository in all areas of the survey. Bitbucket had a small number of mentions by name.

[2] Zenodo was the most highly named archive repository in all areas of the survey. OSF was also mentioned in this context.

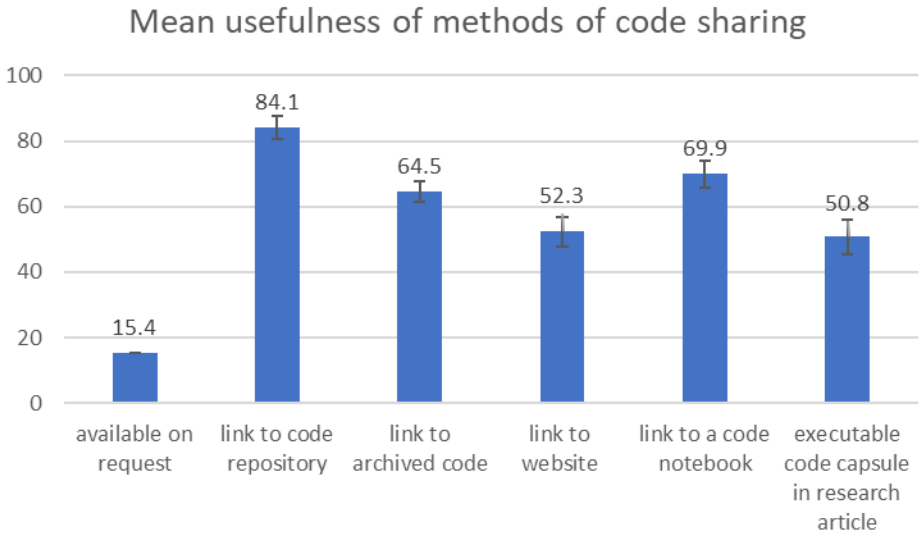


Figure 6 Mean usefulness of different methods of code sharing. A score of zero equates to 'not at all useful' and 100 equals 'extremely useful'. The mean values are given above the bars. Error bars show the 95% confidence interval.

Full-size DOI: 10.7717/peerj.13933/fig-6

The reasons why researchers favoured certain methods of accessing code were gathered via a free text question. The most common reasons, which all received between 18 and 10 mentions, were (in order of number of mentions):

- Ability to see new versions of the code (most associated with code repositories[1])
- Quick to access the code (most associated with code repositories)
- The method allows exploration of the code, which aids understanding (most associated with notebooks)
- The method is associated with good documentation/README files (most associated with code repositories)
- The practicality of the method (most associated with code repositories)
- The method provides long term access to the code (most associated with archived code[2])
- The method allows for reproduction of results (most associated with code repositories and notebooks)
- It is an established method (most associated with code repositories)

Features of code notebooks that are useful when accessing or reading code

All respondents were then asked to rate the importance of various features of the NeuroLibre prototype notebook (*Larremore, 2019*) using a 5-point unipolar scale, or selected that they did not use the feature. Converting these responses to numerical scores on a scale of 0 to 100 and taking the mean (Table 3) gives us a sense of the features readers value the most. The top two features—'having all the code, data and figures in one place' and 'knowing the code is running in the right environment'—are not features unique to code notebooks. Features

Table 3 Mean importance scores given to each feature of the notebook. Table excludes those who answered they did not use the feature or were not aware of its presence/absence. The higher the score, the more important the feature is.

	Mean score	stdev	*n*
Having all the code, data and figures in one place	81.0	22.7	186
Knowing the code is running in the right environment*	73.5	29.3	178
Ability to interact inline with the code in the browser**	66.6	28.5	178
Ability to uncover the data point by hovering over the points on the graph*	65.8	25.3	184
Ability to open up the code as a Jupyter notebook*	64.9	30.5	173
Ability to zoom in/out on the figures*	63.3	26.5	182
Ability to change the parameters of the figure*	62.6	26.7	185
Having extra figures included that were not in the original paper*	53.8	29.0	185

Notes.
*Features marked were included in the NeuroLibre prototype.
**Features marked are present in the NeuroLibre prototype but were not working during the survey period.

related to the interactivity elements of the notebook, *e.g.*, ability to change parameters of the figures, had mean scores in the low to mid 60s. The lowest scoring feature was 'having extra figures included that were not in the original paper'.

Importance and satisfaction of factors associated with sharing code from an author's perspective

Importance and satisfaction responses were converted to numerical scores as described in the Methods section. All factors scored above 50 for mean importance, with standard deviations ranging between 20.6 and 33.3 (Table 4 and Fig. 7). 'Ability to share my code with good accompanying documentation' received the highest mean importance score (82.2, SD: 20.6) and was also fairly well satisfied (72.2 , SD: 23.2). All of the factors have a mean satisfaction score above 50, although the standard deviations all range between 23.2 and 28.8. The lowest scoring factors are 'Readers can easily run the code in the correct environment' (mean satisfaction score 55.4 , SD: 28.0) and 'The data and code are in the same place' (mean satisfaction score 60.4 , SD: 28.8). These are both considered important factors (means scores 76.1 , SD: 23.8 and 73.0 , SD: 28.0 respectively). These are the only two factors that have an opportunity score above 25, although they are not above 36, and therefore present only a marginal opportunity.

Time spent on preparing code as authors

The survey also asked questions about the amount of time authors spent preparing to share their code. The majority of respondents spend more than one day preparing code and this observation holds true when it is separated into cohorts based on the number of papers published (Fig. 8). The researchers with the most papers (>50) are most likely to take more than one week to prepare their code for sharing, whereas the most common response for researchers with fewer papers (<50) was more than one day but less than one

week. This may be a reflection on the number of additional constraints on time felt by more established, *i.e.*, published, researchers, such as teaching or supervision of students.

Time authors are willing to spend improving their methods of sharing code

Respondents were also asked how much extra time they would be willing to spend on using a new tool to make the code easier to read and run. This question was chosen as our preliminary interviews with researchers suggested that making code easier to run and read for others was important for authors, which is supported by the satisfaction and importance scores seen in this survey (Table 4 and Fig. 7). Answers were varied, with the top three responses being 'more than one day' (36%), 'a day' (21%) and 'a couple of hours' (20%). There does not appear to be a trend if the respondents are split into cohorts based on the number of previous publications (Fig. 9). However, those who already spend more than a day preparing their code are more likely to spend extra time on a new tool to improve their code.

DISCUSSION

What do readers value and why?

The findings from this survey show the most prevalent reason for readers looking at code was for verification or examination purposes, with 70% of respondents looking at the code to aid their understanding of the article. In journals where word limits apply, the reproducibility of the research can be compromised if methodological details—in this case computational methods—are not fully detailed (*Samota & Davey, 2021*; *Haddaway & Verhoeven, 2015*) and it is unsurprising, therefore, that researchers commonly look at code to aid their understanding of the work. The number of respondents who wished to rerun (rather than examine) the code for reproducibility reasons was lower (~16%), which has also been observed in other studies (*Peterson & Panofsky, 2021*).

The desire to look at the code rather than run it aligns well with the ranking of a code repository, such as Github, as the most useful method for accessing code by readers (only 1% ranked it as not at all useful), as the presentation of code in these repositories lends itself to exploration or examination but not to immediately rerunning or interacting with code. This survey did not map participants' workflows so they could be downloading and running code locally, although this is not always easy or possible (*Samota & Davey, 2021*). 98% of respondents had encountered code shared *via* code repositories and this prevalence is perhaps a factor in its high usefulness scores as it is widely used by researchers in computational disciplines. The high encounter rate combined with the high usefulness scores indicates that generally readers are satisfied with the most common methods of code sharing.

The survey results also show best practice for code sharing (depositing code in an archive repository) has been encountered by 72% of our respondents. This is a higher percentage than seen in our previous research on data sharing practices where 56% deposit data in a repository. With both code and data, often researchers aren't following what is considered

Table 4 Mean satisfaction and importance scores for code sharing factors. Respondents were asked to rate the factors using a 7 and 5 point Likert scale respectively, which have been converted into scores out of 100. The higher the score, the more important or satisfied the respondent is. The final factor in the table was only asked in relation to importance.

	Mean Satisfaction	Standard deviation	*n*	Mean importance	Standard deviation	*n*	Opportunity score
Ability to share the code in my preferred form, e.g., as zip file or executable code capsule	68.6	26.3	165	60.7	28.3	182	19.1
Ability to share the code in my preferred location/platform	74.0	23.3	176	68.7	29.1	187	17.9
Ability to share my code with good accompanying documentation	72.2	23.2	178	82.2	20.6	187	22.8
Spend less time on the preparing my code for sharing, e.g., cleaning code, writing documentation	67.1	24.3	177	62.5	31.6	188	20.6
Spend less time on uploading the code and documentation to a repository	75.9	23.7	179	56.5	33.3	188	13.6
Depositing my code in a permanent archive	71.1	26.2	169	74.5	26.9	187	21.5
Readers can easily run the code in the correct environment	55.4	28.0	171	76.1	23.8	186	33.9
The data and code are in the same place	60.4	28.8	172	73.0	28.0	183	28.9
Curation checks are run on my code by a third party				58.9	29.4	176	

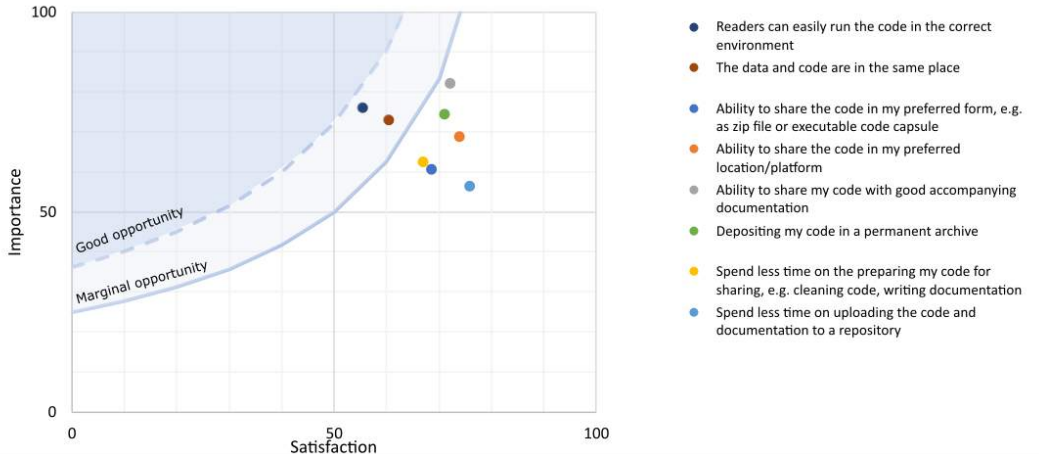


Figure 7 Plot of mean satisfaction and importance scores for each factor related to being an author of code. The blue shaded areas denote where factors have to plot in order to be considered marginal or good opportunities using the Opportunity Score as outlined in the Methods section. Data for the figure are given in Table 4.

Full-size DOI: 10.7717/peerj.13933/fig-7

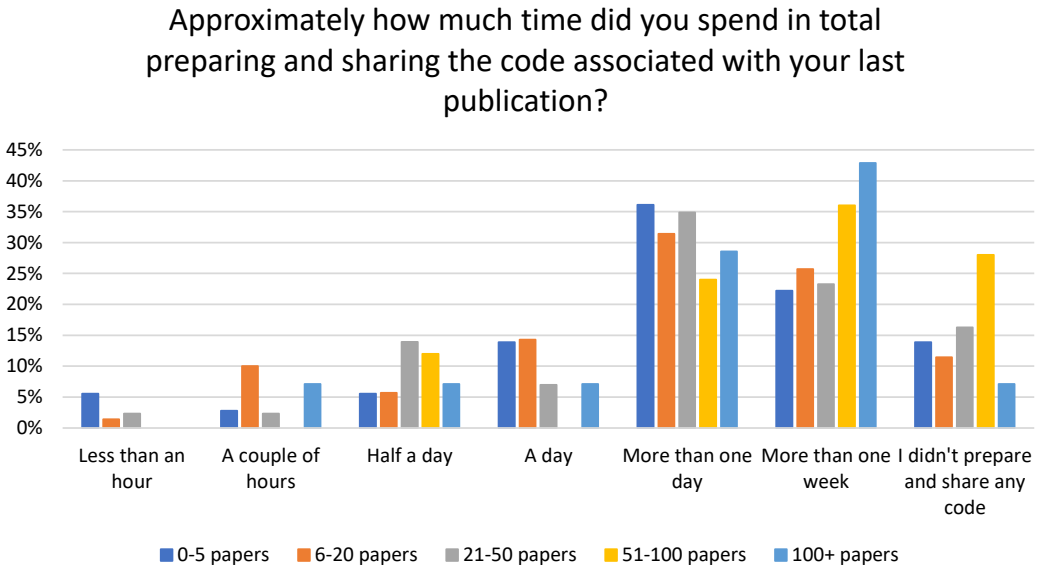


Figure 8 Amount of time spent preparing code for publication by number of published papers.

Full-size DOI: 10.7717/peerj.13933/fig-8

to be best practice (using repositories) but are satisfied with their ability to share data, from their perspective (*Hrynaszkiewicz, Harney & Cadwallader, 2021b*).

At the other end of the scale (discounting the "available on request" option which was viewed very negatively), executable code capsules had the lowest mean usefulness score of all the methods presented (50.8) whereas code notebooks scored higher (69.9). This is interesting given that they have similar features and aims and raises the question: what are

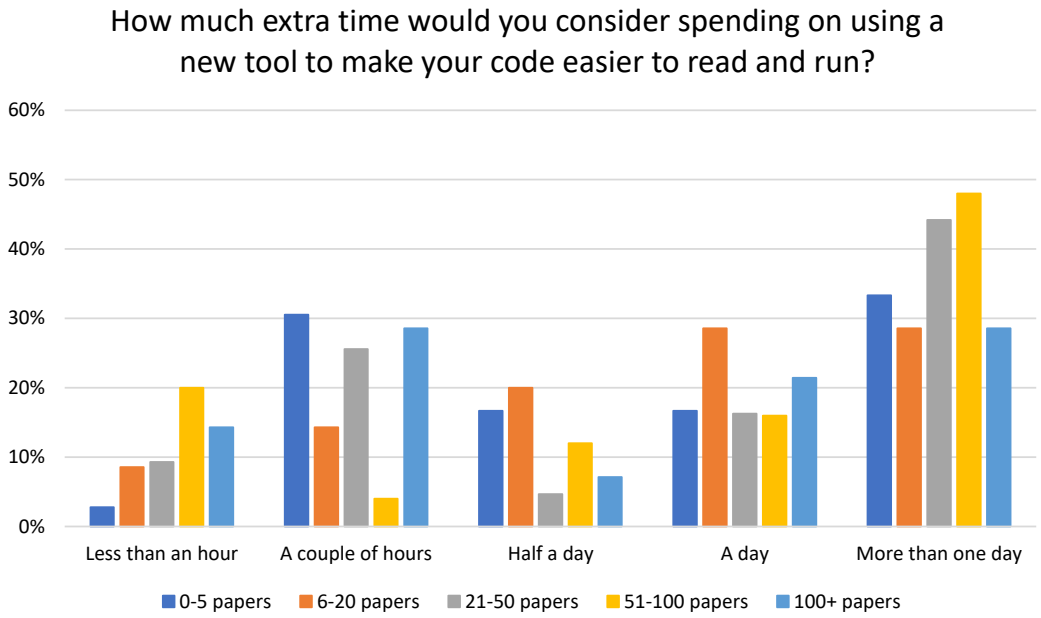


Figure 9 **Amount of time researchers are willing to spend using a new tool to make their code easier to read and run, by number of published papers.**

Full-size DOI: 10.7717/peerj.13933/fig-9

notebooks doing better than code capsules, or what needs are they meeting that capsules aren't? Unfortunately, we cannot answer that question directly with our survey data.

The survey question on why readers favoured certain methods of access give some insight into user needs when it comes to accessing code. Versioning, good documentation and long term access are elements considered best practice for code sharing (*Lamprecht et al., 2020*) and were all amongst the most common reasons given for preferred methods. The other reasons relate to *what* readers wish to do with the code—explore the code and/or reproduce the results in a quick and accessible manner—and are what these methods of code sharing are good at facilitating.

Prototype notebook features

Respondents were asked to rank the importance of a range of features they may have encountered in the prototype notebook, however, many of these features are not exclusive to this notebook and can be found in other code sharing tools. Presenting the prototype notebooks may have affected the respondents' answers to the usefulness of the features, however, given that a third of respondents had not encountered a notebook associated with a research article in the last 6 months the prototype did offer some useful context to those participants and gave all respondents a similar experience to guide their answers. Readers scored 'having all the code, data and figures in one place'—a feature also present in tools such as code capsules—as the most important (mean score 81.0/100; see Table 3). The usefulness of having code, data and figures in one place aligns with how information is often presented in a published article: figures are together with the text, and the data and code are shared (if they are shared) on a different, or multiple different, platforms

making the research outputs dispersed. This issue could be solved in a number of different ways, either through technological solutions (such as notebooks, executable code capsules or imbedded repository widgets on article pages), publishing practices (such as requiring authors to share outputs in a certain way) or through changing researcher behaviour so they share their research as a single package of text, figures, data and code regardless of any mandates or policies they have to comply with or solutions offered by publishers.

The second highest scoring feature (mean score 73.5/100) was 'knowing the code is running in the right environment'. *Samota & Davey (2021)* found that even researchers trained in computational methods had regularly encountered technical barriers to computational reproducibility. Containerisation—packaging the code and all the components needed to run it correctly—is one solution to this problem. It is interesting that this factor scores so high, yet so few respondents wish to run the code, or rated solutions, such as notebooks and executable code capsules, highly for usefulness. Authors scored their satisfaction with their ability to ensure readers are running their code in the correct environment the lowest out of all factors we surveyed (mean 55.4, SD: 28.0). Although this is the lowest score, it is still above 50 and so there is little opportunity to better support this activity. It is not clear from our survey findings that offering a tool to assist with readers running their code in the correct environment would meaningfully change the way readers interact with code although perhaps the possibility of verifying reproducibility will increase confidence in the results (*Nosek et al., 2015*).

The ability to interact with the code inline was ranked as the third most important feature of the prototype code notebook, which supports readers' desire to run, and possibly modify, the code in the correct environment. Conversely, *Samota & Davey (2021)* found a "link to the source code of interactive figures" the least valued feature out of the list in the survey. While this may suggest that readers don't wish to run the code, it may also be an indication that readers don't like having to access links to code (contrary to our findings that researchers like accessing code via repositories). The interactive features, such as zooming in on data points or changing parameters, had lower importance scores, in the low to mid 60s, falling between the moderately important (50/100) and very important (75/100) rating. No one feature of the notebook stands out as being the main reason why respondents would look at a notebook like the one tested—those who scored the likelihood of looking at the notebook highly, generally scored each of the features highly as well.

Other opportunities to support authors

Authors' ability to share the code with good documentation had the highest mean importance score (82.2, SD: 20.6) and a high satisfaction score (mean 72.2, SD:23.2) and good documentation was commonly given as a reason by readers for their preferred method of accessing data. In another survey of computational biology authors (*Hrynaszkiewicz, Harney & Cadwallader, 2021a*), we found that there was a disconnect between how satisfied researchers are with their ability to share code well and the ability of others to share code. That data suggest authors regard themselves as competent at this task but view the competence of others less favourably. This is an area of interest that is worth future exploration to understand if this perceived gap in skills is genuine.

Comparing policy to technology as solutions for increasing code sharing

There is evidence from our survey and others (*e.g.*, *Perkel, 2017*; *Samota & Davey, 2021*) that researchers regard the ability to interact with code published in its complete software environment as beneficial. Using containerisation tools, such as Docker, have been recommended for increasing the reproducibility of research (*Burton et al., 2020*) but it has also been acknowledged that this requires skills that not many researchers in this field have (*Kim, Poline & Dumas, 2018*). Platforms that utilise this technology have been adopted or trialled by several publishers, for example Code Ocean has been deployed by some Springer Nature journals, and some Taylor & Francis journals.

However, it has been acknowledged that authors already using GitHub and Zenodo may feel that the creation of a code capsule is redundant (*Cheifet, 2021*). The trial of code capsules at several Nature journals demonstrated that peer reviewers were verifying the code and reproducing the results of the manuscripts they were assessing (*Cheifet, 2021*) but it is unclear to what extent this was above the level of reviewer engagement seen before the trial or what proportion of reviewers were engaging in this type of activity. Our survey was focused on the needs of readers and authors rather than peer reviewers, but showed that readers have mixed feelings about the usefulness of executable code capsules.

Samota & Davey (2021) state that top-down requirements from journals to release reproducible data and code will in part rely on the availability of technical solutions that are accessible and useful to most scientists. In one sense, these solutions are already available in the form of code repositories, although we acknowledge this doesn't enforce reproducible code and data sharing because the code is not curated or reviewed. However, technology is only one barrier and the journals that have implemented enhanced solutions are, to our knowledge, yet to show that these are making a significant difference to the quality or amount of code that is shared. Additionally, the added benefit, as opposed to the perceived benefit, that they bring to authors and readers versus the use of other methods of sharing, has not been demonstrated. On the other hand, simply sharing the code underlying a publication in a repository has been shown to bring benefits to authors, such as acting as a signal of credibility (*McKiernan et al., 2016*) and increased citations of the article (*Vandewalle, 2012*), which has similarly been shown for data sharing (*Piwowar, Day & Fridsma, 2007*; *Colavizza et al., 2020*).

Whilst quality and reusability of code is very important for increasing the reproducibility, trust and transparency of research; the lack of shared code is still a huge issue that needs to be overcome. *Serghiou et al. (2021)* found that 70% of publishers have never published an article with shared code when analysing over 2.7 million articles in PubMed Central (PMC), and only 2.5% of published articles share code. PLOS journals have higher code sharing rates, with 41% of *PLOS Computational Biology* article sharing code in 2019 (*Serghiou, 2021*).

Additional time to prepare code for sharing

Additional effort is required to produce interactive and executable versions of published research but our survey showed that even for those researchers already engaged in code

sharing, the majority (64%) would not be willing to spend more than a day using a tool that makes code easier to read and run. This suggests that the average researcher may be unwilling to incur additional costs (in time, effort or expenditure) themselves to achieve these outputs, supporting a need for different models for funding and producing these outputs—at least until such time as they can be produced more efficiently. Asking people to predict their future behaviour can lead to overestimation of positive effects (*Wood et al., 2016*) and therefore it is possible that the number of researchers unwilling to spend more than a day on a new tool is actually higher than 64%. During the pilot at Nature journals, the creation of a code capsule took a median time of nine days (*Nature Biotechnology, 2019*). Time has been found to be a barrier to sharing other research outputs, such as data, in other studies as well (see, amongst others, *Perrier, Blondal & MacDonald, 2020*; *Tenopir et al., 2020*; *Digital Science et al., 2021*)

Given the mixed feelings of researchers regarding features of interactive notebooks that are not related to code access, and the lack of desire to invest the required effort to produce them, *PLOS Computational Biology* has opted for the time being to focus on policy and guidance rather than *technological* solutions to improve code sharing. The importance of these *cultural* solutions is often underestimated in relation to reproducible code (*Samota & Davey, 2021*). At *PLOS Computational Biology*, we observed a high degree of voluntary code sharing (*Cadwallader et al., 2021*) before implementation of a mandatory policy, and preliminary results of the impact of the policy on the amount of code shared look positive in line with what has been learnt from implementing mandatory versus optional but encouraged data sharing policy, with the latter causing little change to the status quo (*Christensen et al., 2019*; *Colavizza et al., 2020*; *Statham et al., 2020*). We are focusing on supporting good foundational behaviours by authors that we know are important, such as sharing code with good documentation and metadata (*Kim, Poline & Dumas, 2018*; *Stodden et al., 2016*). As more code associated with publications is made available as a result of these activities, we anticipate there will be more opportunities to understand how the quality, reusability, and interactivity of shared code affect reproducibility—and the role of technological solutions.

Limitations

One possible limitation of this study is non-response bias. As no incentive was offered to complete the survey, respondents who are already motivated to engage with code sharing may have been more likely to participate. The survey was also directed at computational biologists and related disciplines therefore may not be applicable to all disciplines. It is also worth noting that only 34% of respondents identified as working specifically in the computational biology field. Also, there is an uneven distribution in terms of the number of published papers, with most respondents having published fewer than 20 papers, which may limit the generalisability of the findings to other researchers at other career stages. The geographical spread of our respondents also limits the generalisability of our findings. The survey did not give explanations of the different methods of code sharing and assumed the respondents to be familiar with terms such as ‘‘code capsule’’ and ‘‘archived in an open access repository’’.

CONCLUSIONS

The survey findings have given some valuable insights into researcher behaviour and attitudes towards code sharing and more interactive, executable or reproducible publication formats –which require much effort to create. We have observed a "negative result" with regard to clear opportunities for implementing new features and services in the publishing workflow, but we have a better understanding of why researchers look at code—this predominantly seems to be to better understand the article and code used. This is an issue that could be addressed with multiple potential solutions that we did not evaluate, such as reporting guidelines for methods of relevant studies. Further, the results suggest that researchers are on the whole satisfied with code being shared via a code repository, such as GitHub, because this is a well used tool that gives the user freedom to use the code how they wish (*e.g.*, download, fork, read through). Good accompanying documentation is important to researchers and whilst they think their ability to produce documentation is good, the readers of their code may disagree.

Authors of code have variable practices when it comes to the amount of time they spend preparing code. It is unclear if those spending minimal amounts of time preparing code are doing so because their code is already well prepared for sharing, or because they do not attach much importance to spending time preparing their code as it is not regarded as necessary for career advancement, or because they do not have the time to spend on preparation. The NeuroLibre interactive code notebook demonstrated that readers find many of the features valuable and overall they are generally supportive of notebooks but do not see them as revolutionary in the way code is shared. For publishers wishing to experiment with or implement interactive features or versions of articles, it is important to note that researchers (authors) are likely to need additional support or funding to be incentivised to create these outputs. For publishers wishing to increase code sharing, policy may be a more effective solution, in the computational biology community.

ACKNOWLEDGEMENTS

The authors thank James Harney, Gary Beardmore, Helen McDonald and Philip Mills from PLOS for their contributions to the survey work. We also thank James Harney, Marcel LaFlamme and Dan Morgan from PLOS and Professor Jason Papin, University of Virginia and PLOS Computational Biology co-Editor-in-Chief, for comments on an earlier version of this manuscript. We would also like to thank NeuroLibre for the creation of the prototype notebooks and engaging in experimentation with us.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

The authors received no funding for this work.

Competing Interests

Both authors are employees of Public Library of Science (PLOS).

Author Contributions

- Lauren Cadwallader conceived and designed the experiments, performed the experiments, analyzed the data, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Iain Hrynaszkiewicz conceived and designed the experiments, authored or reviewed drafts of the article, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

The survey instrument used and anonymised survey data is available at Figshare: Cadwallader, Lauren; Hrynaszkiewicz, Iain; Harney, James (2022): Data from: A survey of researchers' code sharing and reuse practices, and assessment of interactive notebook prototypes. figshare. Dataset. https://doi.org/10.6084/m9.figshare.19122611.v1.

REFERENCES

Akhlaghi M, Infante-Sainz R, Roukema BF, Khellat M, Valls-Gabaud D, Baena-Gallé R. 2021. Toward long-term and archivable reproducibility. *Computing in Science & Engineering* **23**:82–91 DOI 10.1109/MCSE.2021.3072860.

Boudreau M, Poline J-B, Bellec P, Stikov N. 2021. On the open-source landscape of PLOS Computational Biology. *PLOS Computational Biology* **17**:e1008725 DOI 10.1371/journal.pcbi.1008725.

Burton M, Lavin MJ, Otis J, Weingart SB. 2020. Digits: two reports on new units of scholarly publication. *The Journal of Electronic Publishing* **22**:1 DOI 10.3998/3336451.0022.105.

Cadwallader L. 2021. Exploring code notebooks through community focused collaboration. *Available at https://theplosblog.plos.org/2021/02/exploring-code-notebooks-through-community-focused-collaboration/* (accessed on 14 January 2022).

Cadwallader L, Hrynaszkiewicz I, Harney J. 2022. Data from: a survey of researchers' code sharing and reuse practices and assessment of interactive notebook prototypes. Figshare. Dataset DOI 10.6084/m9.figshare.19122611.

Cadwallader L, Papin JA, Mac Gabhann F, Kirk R. 2021. Collaborating with our community to increase code sharing. *PLOS Computational Biology* **17**:e1008867 DOI 10.1371/journal.pcbi.1008867.

Cheifet B. 2021. Promoting reproducibility with Code Ocean. *Genome Biology* **22**:65 DOI 10.1186/s13059-021-02299-x.

Christensen G, Dafoe A, Miguel E, Moore DA, Rose AK. 2019. A study of the impact of data sharing on article citations using journal policies as a natural experiment. *PLOS ONE* **14**:e0225883 DOI 10.1371/journal.pone.0225883.

Colavizza G, Hrynaszkiewicz I, Staden I, Whitaker K, McGillivray B. 2020. The citation advantage of linking publications to research data. *PLOS ONE* **15**:e0230416 DOI 10.1371/journal.pone.0230416.

Digital Science, Simons N, Goodey G, Hardeman M, Clare C, Gonzales S, Strange D, Smith G, Kipnis D, Iida K, Miyairi N, Tshetsha V, Ramokgola R, Makhera P, Barbour G. 2021. The State of Open Data 2021. Figshare DOI 10.6084/m9.figshare.17061347.v1.

Fernández-Juricic E. 2021. Why sharing data and code during peer review can enhance behavioral ecology research. *Behavioral Ecology and Sociobiology* **75(103)**:s00265–021–03036–x DOI 10.1007/s00265-021-03036-x.

Haddaway NR, Verhoeven JTA. 2015. Poor methodological detail precludes experimental repeatability and hampers synthesis in ecology. *Ecology and Evolution* **5**:4451 DOI 10.1002/ece3.1722.

Hrynaszkiewicz I. 2020. Publishers' responsibilities in promoting data quality and reproducibility. In: Bespalov A, Michel MC, Steckler T, eds. *Good research practice in non-clinical pharmacology and biomedicine. handbook of experimental pharmacology*. Cham: Springer International Publishing, 319–348 DOI 10.1007/164_2019_290.

Hrynaszkiewicz I, Harney J, Cadwallader L. 2021a. A survey of code sharing practice and policy in computational biology. *OSF Preprint* DOI 10.31219/osf.io/f73a6.

Hrynaszkiewicz I, Harney J, Cadwallader L. 2021b. A survey of researchers' needs and priorities for data sharing. *Data Science Journal* **20**:31 DOI 10.5334/dsj-2021-031.

Kim Y-M, Poline J-B, Dumas G. 2018. Experimenting with reproducibility: a case study of robustness in bioinformatics. *GigaScience* **7**:giy077 DOI 10.1093/gigascience/giy077.

Konkol M, Nüst D, Goulier L. 2020. Publishing computational research—a review of infrastructures for reproducible and transparent scholarly communication. *Research Integrity and Peer Review* **5**:10 DOI 10.1186/s41073-020-00095-y.

Lamprecht A-L, Garcia L, Kuzak M, Martinez C, Arcila R, Martin Del Pico E, Dominguez Del Angel V, vandeSandt S, Ison J, Martinez PA, McQuilton P, Valencia A, Harrow J, Psomopoulos F, Gelpi JL, Chue Hong N, Goble C, Capella-Gutierrez S. 2020. Towards FAIR principles for research software. *Data Science* **3**:37–59 DOI 10.3233/DS-190026.

Larremore DB. 2019. Bayes-optimal estimation of overlap between populations of fixed size. *PLOS Computational Biology* **15(3)**:e1006898 DOI 10.1371/journal.pcbi.1006898.

Lasser J. 2020. Creating an executable paper is a journey through Open Science. *Communications Physics* **3**:1–5 DOI 10.1038/s42005-020-00403-4.

McKiernan EC, Bourne PE, Brown CT, Buck S, Kenall A, Lin J, McDougall D, Nosek BA, Ram K, Soderberg CK, Spies JR, Thaney K, Updegrove A, Woo KH, Yarkoni T. 2016. How open science helps researchers succeed. *eLife* **5**:e16800 DOI 10.7554/eLife.16800.

Nature Biotechnology. 2019. Changing coding culture. *Nature Biotechnology* **37**:485–485 DOI 10.1038/s41587-019-0136-9.

Nosek BA, Alter G, Banks GC, Borsboom D, Bowman SD, Breckler SJ, Buck S, Chambers CD, Chin G, Christensen G, Contestabile M, Dafoe A, Eich E, Freese J, Glennerster R, Goroff D, Green DP, Hesse B, Humphreys M, Ishiyama J, Karlan D, Kraut A, Lupia A, Mabry P, Madon T, Malhotra N, Mayo-Wilson E, McNutt M, Miguel E, Paluck EL, Simonsohn U, Soderberg C, Spellman BA, Turitto J,

VandenBos G, Vazire S, Wagenmakers EJ, Wilson R, Yarkoni T. 2015. Promoting an open research culture. *Science* **348**:1422–1425 DOI 10.1126/science.aab2374.

Perkel JM. 2017. TechBlog: interactive figures address data reproducibility. *Naturejobs Blog Available at http://blogs.nature.com/naturejobs/2017/10/20/techblog-interactive-figures-address-data-reproducibility/* (accessed on 07 January 2022).

Perkel JM. 2019. Make code accessible with these cloud services. *Nature* **575**:247–248 DOI 10.1038/d41586-019-03366-x.

Perkel JM. 2021. Reactive, reproducible, collaborative: computational notebooks evolve. *Nature* **593**:156–157 DOI 10.1038/d41586-021-01174-w.

Perrier L, Blondal E, MacDonald H. 2020. The views, perspectives, and experiences of academic researchers with data sharing and reuse: a meta-synthesis. *PLOS ONE* **15**:e0229182 DOI 10.1371/journal.pone.0229182.

Peterson D, Panofsky A. 2021. Self-correction in science: the diagnostic and integrative motives for replication. *Social Studies of Science* **51**:583–605 DOI 10.1177/03063127211005551.

Piwowar HA, Day RS, Fridsma DB. 2007. Sharing detailed research data is associated with increased citation rate. *PLOS ONE* **2**:e308 DOI 10.1371/journal.pone.0000308.

Samota EK, Davey RP. 2021. Knowledge and attitudes among life scientists toward reproducibility within journal articles: a research survey. *Frontiers in Research Metrics and Analytics* **6**:35 DOI 10.3389/frma.2021.678554.

Seibold H, Czerny S, Decke S, Dieterle R, Eder T, Fohr S, Hahn N, Hartmann R, Heindl C, Kopper P, Lepke D, Loidl V, Mandl M, Musiol S, Peter J, Piehler A, Rojas E, Schmid S, Schmidt H, Schmoll M, Schneider L, To X-Y, Tran V, Völker A, Wagner M, Wagner J, Waize M, Wecker H, Yang R, Zellner S, Nalenz M. 2021. A computational reproducibility study of PLOS ONE articles featuring longitudinal data analyses. *PLOS ONE* **16**:e0251194 DOI 10.1371/journal.pone.0251194.

Serghiou S. 2021. Assessment of transparency indicators across the biomedical literature: how open is open? OSF Dataset DOI 10.17605/OSF.IO/E58W.

Serghiou S, Contopoulos-Ioannidis DG, Boyack KW, Riedel N, Wallach JD, Ioannidis JPA. 2021. Assessment of transparency indicators across the biomedical literature: How open is open? *PLOS Biology* **19**:e3001107 DOI 10.1371/journal.pbio.3001107.

Statham EE, White SA, Sonwane B, Bierer BE. 2020. Primed to comply: individual participant data sharing statements on ClinicalTrials.gov. *PLOS ONE* **15**:e0226143 DOI 10.1371/journal.pone.0226143.

Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, Heroux MA, Ioannidis JPA, Taufer M. 2016. Enhancing reproducibility for computational methods. *Science* **354**:1240–1241 DOI 10.1126/science.aah6168.

Tampuu A, Matiisen T, Ólafsdóttir HF, Barry C, Vicente R. 2019. Efficient neural decoding of self-location with a deep recurrent network. *PLOS Computational Biology* **15(2)**:e1006822 DOI 10.1371/journal.pcbi.1006822.

Tenopir C, Rice NM, Allard S, Baird L, Borycz J, Christian L, Grant B, Olendorf R, Sandusky RJ. 2020. Data sharing, management, use, and reuse: practices and perceptions of scientists worldwide. *PLOS ONE* **15**:e0229003 DOI 10.1371/journal.pone.0229003.

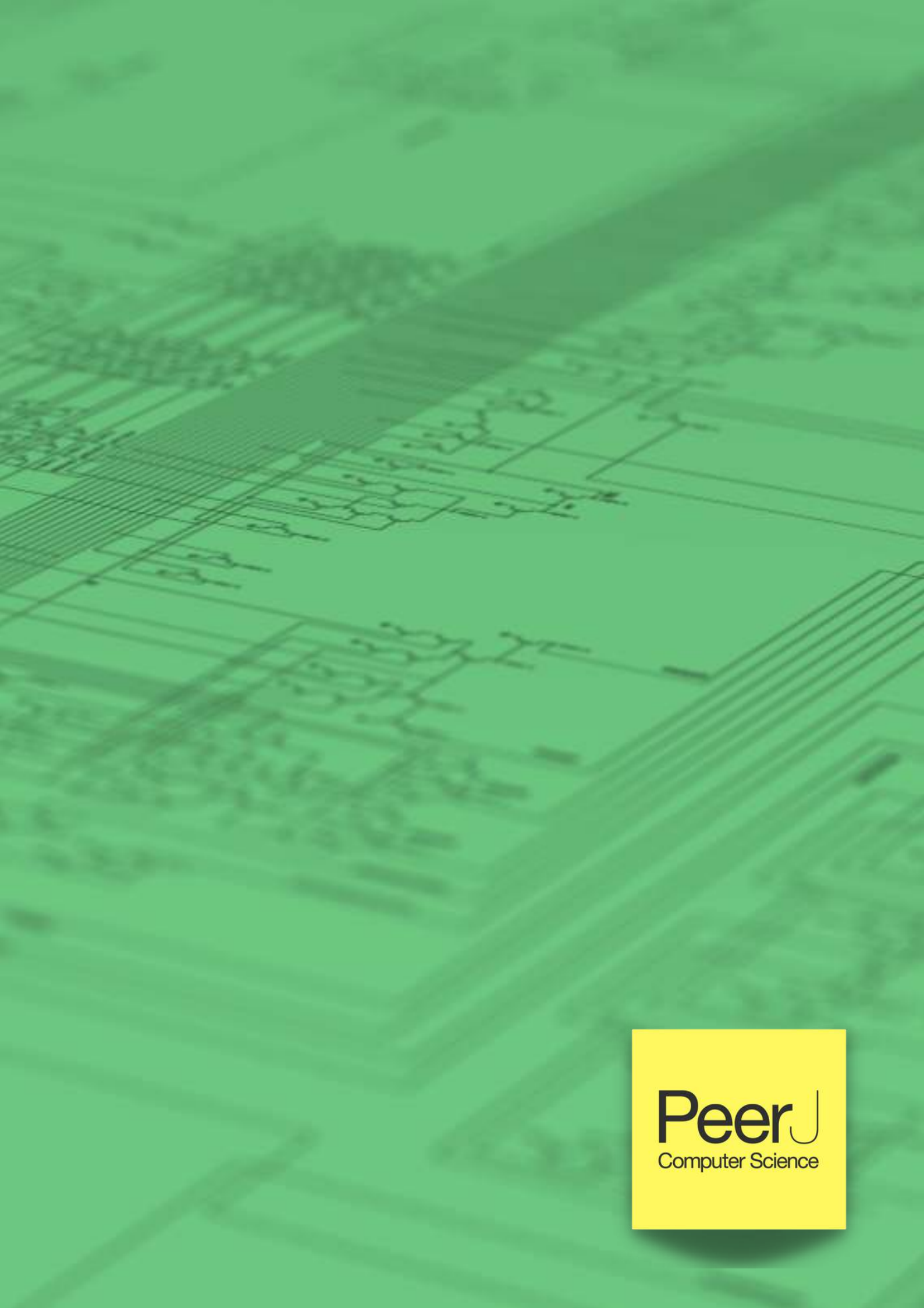
Tsang E, Maciocci G. 2020. Welcome to a new ERA of reproducible publishing. *Available at https://elifesciences.org/labs/dc5acbde/welcome-to-a-new-era-of-reproducible-publishing* (accessed on 14 January 2022).

Ulwick AW, Osterwalder A. 2016. *Jobs to be done: theory to practice*. Houston: Idea Bite Press.

Van den Eynden V, Knight G, Vlad A, Radler B, Tenopir C, Leon D, Manista F, Whitworth J, Corti L. 2016. Survey of Wellcome researchers and their attitudes to open research. *1843500 Bytes* DOI 10.6084/M9.FIGSHARE.4055448.V1.

Vandewalle P. 2012. Code sharing is associated with research impact in image processing. *Computing in Science Engineering* **14**:42–47 DOI 10.1109/MCSE.2012.63.

Wood C, Conner M, Miles E, Sandberg T, Taylor N, Godin G, Sheeran P. 2016. The impact of asking intention or self-prediction questions on subsequent behavior: a meta-analysis. *Personality and Social Psychology Review* **20**:245–268 DOI 10.1177/1088868315592334.



PeerJ
Computer Science