Scientific Open-Source Software Is Less Likely to Become Abandoned Than One Might Think! Lessons from Curating a Catalog of Maintained Scientific Software

ADDI MALVIYA THAKUR, University of Tennessee at Knoxville, USA and Oak Ridge National Lab, USA
REED MILEWICZ, Sandia National Laboratories, USA
MAHMOUD JAHANSHAHI, University of Tennessee at Knoxville, USA
LAVÍNIA PAGANINI, Eindhoven University of Technology, Netherlands
BOGDAN VASILESCU, Carnegie Mellon University, USA
AUDRIS MOCKUS, University of Tennessee at Knoxville, USA

Scientific software is essential to scientific innovation and in many ways it is distinct from other types of software. Abandoned (or unmaintained), buggy, and hard to use software, a perception often associated with scientific software can hinder scientific progress, yet, in contrast to other types of software, its longevity is poorly understood. Existing data curation efforts are fragmented by science domain and/or are small in scale and lack key attributes. We use large language models to classify public software repositories in World of Code into distinct scientific domains and layers of the software stack, curating a large and diverse collection of over 18,000 scientific software projects. Using this data, we estimate survival models to understand how the domain, infrastructural layer, and other attributes of scientific software affect its longevity. We further obtain a matched sample of non-scientific software repositories and investigate the differences. We find that infrastructural layers, downstream dependencies, mentions of publications, and participants from government are associated with a longer lifespan, while newer projects with participants from academia had shorter lifespan. Against common expectations, scientific projects have a longer lifetime than matched non-scientific open-source software projects. We expect our curated attribute-rich collection to support future research on scientific software and provide insights that may help extend longevity of both scientific and other projects.

CCS Concepts: • **Software and its engineering → Software creation and management**; **Software organization and properties**; • **Human-centered computing → Collaborative and social computing**.

Additional Key Words and Phrases: Open Source Software, Scientific Software, Software Ecosystems, Software Sustainability, Software Longevity, Research Software Engineering, Large Language Models

ACM Reference Format:
Addi Malviya Thakur, Reed Milewicz, Mahmoud Jahanshahi, Lavínia Paganini, Bogdan Vasilescu, and Audris Mockus. 2025. Scientific Open-Source Software Is Less Likely to Become Abandoned Than One Might Think! Lessons from Curating a Catalog of Maintained Scientific Software. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE099 (July 2025), 24 pages. https://doi.org/10.1145/3729369

Authors' Contact Information: Addi Malviya Thakur, University of Tennessee at Knoxville, Knoxville, USA and Oak Ridge National Lab, Oak Ridge, USA; Reed Milewicz, Sandia National Laboratories, Albuquerque, USA, rmilewi@sandia.gov; Mahmoud Jahanshahi, University of Tennessee at Knoxville, Knoxville, USA, mjahansh@vols.utk.edu; Lavínia Paganini, Eindhoven University of Technology, Eindhoven, Netherlands, l.f.paganini@tue.nl; Bogdan Vasilescu, Carnegie Mellon University, Pittsburgh, USA, vasilescu@cmu.edu; Audris Mockus, University of Tennessee at Knoxville, Knoxville, USA, audris@utk.edu.

ACM 2994-970X/2025/7-ARTFSE099
https://doi.org/10.1145/3729369

1 Introduction

Computing is central to science, enabling next-generation simulations and data analyses that drive innovations in fields like medicine, energy, climate science, and engineering [Cerf 2024]. These advances all depend on an ever-growing ecosystem of open-source and community-driven scientific software. In the past two decades, the scientific software community has largely moved away from privately-developed software used by small teams, and towards open science and open data as well as FAIRness (findability, accessibility, interoperability, and reusability) [Fouilloux et al. 2023; Lamprecht et al. 2020; Ramachandran et al. 2021].

As this ecosystem has grown, however, so too have concerns about its sustainability, understood broadly as its *ability to endure* [Becker et al. 2015], or to be maintained in a state where it continues to provide value to the scientists using it (e.g., [Morris 2021; Trainer et al. 2014]). There are many challenges to sustaining open-source software (OSS) [Chengalur-Smith et al. 2010; Coelho et al. 2017; Valiev et al. 2018], to which scientific OSS adds a unique set. These include insufficient training in software development on the part of researcher-developers [Howison and Herbsleb 2011], interdisciplinary collaboration challenges [Sun et al. 2024], mismatched priorities of science funders that tend not to value software work highly [Johanson et al. 2018], and a need for highly specialized skills for ongoing maintenance to not only maintain compatibility, portability, etc., but also scientific relevance [Rechert et al. 2021]. In short, progress has been promising but scaling sustainable and trustworthy OSS software still hinders scientific advances.

However, while there is increasing recognition of the importance of scientific OSS and of the challenges to sustaining this digital infrastructure, we lack a centralized view of this infrastructure and its state of maintenance. We also lack empirical evidence on the key factors that influence the health and sustainability of scientific OSS. Both are critically needed, e.g., to draw attention to critical scientific OSS projects that are undermaintained and at risk of becoming abandoned, and to help guide resource allocation.

To this end, in this paper, we report on a two-part empirical study (Figure 1). First, we present a novel approach to identify scientific OSS repositories and classify them in terms of scientific domain and software stack layer (from small-scale applications to infrastructure), resulting in a dataset of 18,247 repositories. Second, we conduct survival analyses of these repositories, modeling the factors associated with a higher risk of becoming abandoned and estimating how the risk of becoming abandoned compares to a matched set of non-scientific OSS projects.

Our analysis reveals several key insights: Many scientific software repositories lack mentions of publications or funding, and thus risk being overlooked. Mathematics-related projects exhibit the longest average lifespan, whereas projects in computer science have the shortest. Scientific infrastructure-layer projects demonstrate the greatest longevity, supported by community involvement, more downstream users, and government participation. Projects with more upstream dependencies and academic involvement tend to have shorter lifespans. And finally, scientific OSS tends to survive longer than non-scientific counterparts.

In short, the contributions of our work are: (1) a validated AI-assisted methodology for detecting scientific software repositories at scale, (2) a cross-cutting dataset of 18,247 scientific software repositories spanning different scientific domains and technology stack layers, (3) quantitative findings on scientific software longevity including survival curves and proportional hazards models, and (4) a comparison between scientific OSS and generic OSS repositories.

2 Background

Before we present our research questions, we start with some background on the scientific software ecosystem and the challenges to its sustainability, to set the stage.

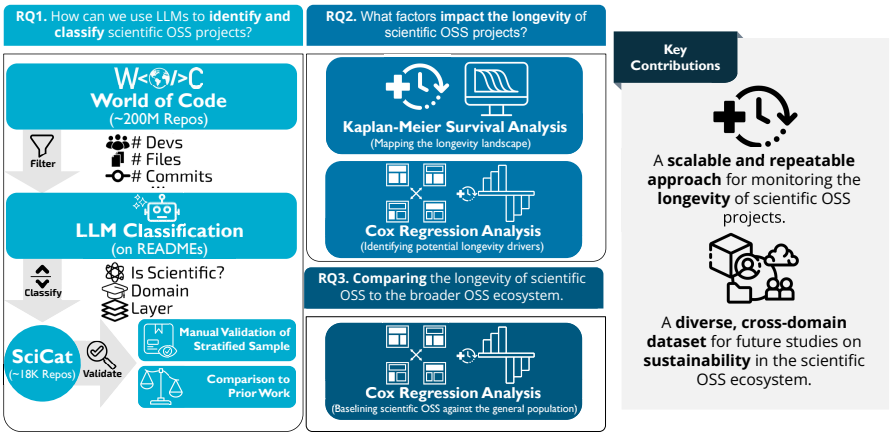

Fig. 1. A visual walkthrough of the methodology used in our paper.

2.1 The Scientific Software Ecosystem

Scientific software broadly refers to software that is used for scientific purposes (see [Kanewala et al. 2014]), e.g., domain-specific simulations, numerical libraries, visualization tools, and workflow managers [Sochat et al. 2022]. It plays a central role in the scientific enterprise and drives discoveries that advance our well-being, prosperity, security, and understanding of the universe.

Scientific software development has had a long and storied history [Dyson 2012], but largely independent from conventional software development [Faulk et al. 2009]. Early-days scientific software was, as a rule, developed in private by small independent teams of domain scientists and mathematicians without formal training in software engineering [Johanson et al. 2018]. In the past decade and a half, a convergence of different factors has forced the scientific community to revisit how it develops software; these include a crisis of lost productivity and credibility of results due to poor software quality [Johanson et al. 2018], scientific challenges that cannot be solved without integrated multidisciplinary solutions [Ober et al. 2017], and a shift toward greater openness in science through the sharing of code and data [Ramachandran et al. 2021]. This has led to scientific software becoming increasingly (1) community-driven (*c.f.,* [Bonomi 2019; Charles et al. 2020]), (2) open source [Hasselbring et al. 2020; Katz et al. 2018], and (3) produced both by researchers who have a growing interest in software engineering best practices [Adorf et al. 2018; Dubey et al. 2020; Haider et al. 2021; Lee 2018; Queiroz et al. 2017] and a growing number of research software engineering (RSE) professionals [Baxter et al. 2012; Brett et al. 2017; Cohen et al. 2020].

Today, scientific software is part of a predominately open-source **software ecosystem** (or SECO) — also referred to by [Jansen et al. 2022] as the Worldwide Research Software Ecosystem (WRSE). Here, we use the definition by [Manikas et al. 2013], which is that an ecosystem encompasses "the interaction of a set of actors on top of a common technological platform that results in a number of software solutions". That is, we have a diverse range of **actors** (individual scientific software developers, teams, and institutions) coordinating their work on common **platforms** (*e.g.,* GitHub, Gitlab, and Bitbucket) to develop software **solutions** for scientific applications (*e.g.,* multi-physics simulation codes, high-performance computing libraries, and visualization tools) which are assembled into complex software stacks by users (who are frequently developers themselves).

According to Hinsen's conceptual model of scientific software stacks [Hinsen 2019], scientific software developers may contribute packages at different **layers**: the topmost layer consists of **project-specific code**, more likely written by individual scientists, and in an ad-hoc manner for a

particular research project; one layer below are **domain-specific tools**, which can vary in their level of maturity and user base; finally, below there is **infrastructure** created specifically for scientific computing, which is expected to be mature and to serve multiple fields of research.

2.2 (Un)sustainability in the Scientific Software Ecosystem

As the scientific software community has matured, the shift towards a complex, interdependent, and open-source SECO has also introduced new challenges, especially concerning software **sustainability**. By sustainability, we follow the the Karlskrona Manifesto to mean the ability of a socio-technical system (such as a software project) to endure, a concept which spans individual, social, technical, economic, and environmental dimensions [Becker et al. 2015]. However, we focus on the **technical** and **economic** dimensions of sustainability,[1] as these align most directly with another common view of sustainability in the software literature, i.e., "the ability to maintain the software in a state where scientists can understand, replicate, and extend previously reported results that depend on that software" [Trainer et al. 2014]. In this sense, the **longevity** of information, systems, and infrastructure is a prerequisite for their adequate evolution with changing surrounding conditions (technical) and maintaining capital and added value (economic), respectively.

As discussed above, the scientific software community has moved towards closer coordination and greater reuse of each other's open-source software. That shift exposes projects to the usual OSS sustainability problems plus science-specific pressures. In conventional OSS SECOs, packages can become obsolete or inactive [Cogo et al. 2021; Miller, Jahanshahi, et al. 2025], breaking changes can cause cascading effects on projects [Bogart et al. 2021], communities around software projects can rise and fall [Valiev et al. 2018], projects can die due to lack of financial viability [Xavier et al. 2020], and poor governance can cause an ecosystem to splinter and fall apart [Gamalielsson et al. 2014]; the scientific software ecosystem is much the same in these respects. However, unlike the creation or maintenance of OSS in and of itself, many scientific applications have to be regularly ported to new and cutting-edge hardware (necessitating costly rework) [Rechert et al. 2021], when packages are revealed to be untrustworthy and irreproducible it can jeopardize any research based on them (possibly harming the careers of the users!) [Soergel 2014], and much of the scientific software in existence was created by people whose job is to do science, *not* to write software [Johanson et al. 2018]. These dynamics concern the public and funders who support and nurture this ecosystem [Barker et al. 2022; Strasser et al. 2022]; without a holistic perspective, they run the risk of funding duplicative projects, failing to grow communities around software to carry it forward, or propping up poor quality software that collapses under its own technical debt. At present, however, there is relatively little data on scientific software sustainability at an ecosystem level, which impairs effective decision-making.

In recent years there have been a range of works on scientific software sustainability, including experience reports and case studies [Sun et al. 2024], reports on developers' practices and perceptions of sustainability [Feitosa et al. 2023], and community-building strategies [Ram et al. 2018]. On the repository mining front, studies have investigated different proxies for sustainability, such as cross-project references [Sun et al. 2024] and complexity metrics [Willenbring et al. 2021]. These studies, however, are relatively small scale and domain specific.

In contrast, we provide a baseline of data to support understanding and awareness of the broader scientific OSS ecosystem. For that reason we focus our attention on **longevity** of projects and correlates (*e.g.,* community size, number of dependencies, etc.), as these are well-attested in the SECO literature and allow us to compare scientific OSS to other kinds of OSS documented in prior work. Given that the transferability of findings from the conventional OSS SECO literature to the scientific software domain is uncertain, having this data allows us to make well-grounded comparisons.

[1]This is not to suggest that environmental, individual, and social dimensions of sustainability are of lesser concern.

While this study focuses on STEM disciplines, scientific software is also essential in domain such as social sciences, arts, and humanities. For example, fields such as linguistics, archaeology, history, and political science rely on computational tools for text analysis, geospatial mapping, and data mining [Arnold et al. 2024; Stoltz et al. 2024]. However, these disciplines often have different funding models, collaboration structures, and sustainability challenges [Newfield 2025].

3 Research Questions

Our study consists of two parts. First, we take a representative cross-section of the scientific OSS ecosystem, which requires a new technique to identify and categorize the software projects, as no such dataset exists. Second, we study the factors associated with increased longevity (or, conversely, increased risk of abandonment) within our sample and across scientific and non-scientific OSS. Next we formulate and motivate each of our research questions.

RQ1. How can we use LLMs to identify and classify scientific OSS projects?

What We Know. Ours is not the first study to assemble a corpus of scientific software projects. Several efforts have been made to build small datasets of projects to study particular phenomena such as security [Murphy et al. 2020], contributor roles [Milewicz et al. 2019], sustainability-related practices and incentives [Howison, Deelman, et al. 2015], and tools for metadata extraction [Mao et al. 2019]. Other studies have sought to build larger datasets by using publication metadata (e.g., citations; and metadata entries in publications, grant-funding-related archives, and institutional catalogs) [Garijo et al. 2024; Kelley et al. 2021; Wattanakriengkrai et al. 2022]. The closest work to ours in terms of scale and methodology is by Wattanakriengkrai et al. [2022], who scanned GitHub for repositories with links to academic papers, turning up 20,278 links. Finally, there have been several efforts in recent years to make scientific software more discoverable such as Papers with Code which catalog and promote code repositories associated with publications.

What We Do Not Know. Prior works, while all valuable, (1) have focused on populations that are very small, (2) target a specific domain, (3) do not break down projects according to layers, or (4) rely on some kind of self-reporting such as registering with an online catalog or GitHub links in publications. We hypothesize, however, that both publication and citation practices as well as software development practices may differ across scientific domains and layers of the software stack. This led us down a different path, which is to detect scientific software repositories online based on their content. In particular, we investigate whether large language models (LLMs) could assist with detecting and classifying these repositories.

RQ2. What factors impact the longevity of scientific OSS projects?

What We Know. It is argued that "the lifespan of scientific software tends to be either very long or very short" [Sanders et al. 2008]. Exact numbers, however, are harder to come by; as Hinsen [2019] has noted, the "uncertain survival probability" of any given project complicates the decision-making for all actors in the ecosystem. The works most closely related to ours are a pilot study by Hasselbring et al. [2020] and follow-up work by Eitzen [2020], who measured longevity of scientific software projects found by searching for mentions of DOIs/citations in repositories and by URLs in papers. Among their findings, [Hasselbring et al. 2020] report significant differences between domains in terms of longevity; in particular, they report that the median lifespan for computational science software repositories in their dataset is 15 days vs. 5 years for computer science repositories.

What We Do Not Know. As both Hasselbring et al. [2020] and Eitzen [2020] include one-off repositories for scripts associated with papers in their samples, it is unclear how previous findings would generalize to scientific software projects that are intended to be actively developed and

maintained, and that may have a large dependent user base. While we recognize that there are sustainability concerns involved in those one-off paper repositories (e.g., for reproducibility of scientific results), arguably becoming under- or unmaintained does not pose as much of a threat to the stability of the ecosystem as would, say, a core math library shutting down. Thus, we focus on scientific OSS projects that are more likely intended to endure, and ask the following subquestions:

RQ2a. How long do scientific OSS projects survive, and how does this vary across different domains and layers of the software stack?

RQ2b. What factors predict the longevity of scientific OSS projects?

Longevity is a prerequisite for the much more complex notion of sustainability. With occasional exceptions of "feature-complete" projects [Valiev et al. 2018], software systems need continuous maintenance to remain compatible with changing environments, fix bugs and security vulnerabilities, add needed features, etc. In OSS, such maintenance cannot be taken for granted, as contributors and maintainers are often volunteers, and typically free to disengage at any time. All these maintenance updates, at the very least, require a project to be active ("alive"), as opposed to dormant or abandoned.[2] Thus, longevity is a critical indicator of health and success in both traditional [Valiev et al. 2018; M. Zhou, Mockus, et al. 2016] and scientific open-source software [Jørgensen 2021].

RQ3. How does the longevity of scientific OSS compare to the broader OSS ecosystem?

What We Know. As discussed in Section 2, there are (1) clear differences that distinguish scientific software development from its conventional counterpart, but also that (2) as the scientific software ecosystem has grown, it has had to contend with problems shared by many other OSS ecosystems.

What We Do Not Know. The challenges that are unique to scientific OSS may change the dynamics of the scientific software ecosystem in unpredictable ways, possibly threatening the sustainability of scientific OSS even further. Given the importance of scientific open-source software for science, it is important to better understand how it compares to other types of OSS to have a better chance of identifying undermaintained projects in need of additional support. To our knowledge, there are no studies to date that have empirically compared the OSS scientific software ecosystem to any non-scientific counterparts. From a software engineering research perspective, being able to compare and contrast scientific and non-scientific software at an ecosystem level would be very helpful, as it builds on the wealth of OSS ecosystem research that is already available.

4 RQ1: Identifying and Classifying Scientific OSS Software

We start by presenting our novel approach to identify scientific OSS repositories based on their contents and classify them in terms of domain and Hinsen's software stack layers [Hinsen 2019].

4.1 Methods

The main goal of our approach is to build a large, multi-domain, multi-layer dataset of scientific software with public code repositories that we will use to study scientific software longevity in different contexts at scale. To this end, we developed and validated an automatable, content-centric approach to identify and categorize scientific software among OSS repositories indexed by the World of Code (WoC) research infrastructure [Ma et al. 2021]. Our method uses a large language model (LLM) prompted with the contents of the README files of the repositories. READMEs typically contain information on what a project does and how it works [Prana et al. 2019], and we expect this information can identify scientific software even when no explicit links to publications or acknowledgments of grants are present, which is an identification approach used in prior work.

[2]Although being "alive" is not sufficient for a project to be well-maintained, as the available maintenance effort may still fall short of the demand for updates [Champion et al. 2021].

At a high level, our process consists of two parts: (a) sampling OSS repositories using the WoC infrastructure and (b) using an LLM to categorize them.

Sampling and Pre-processing: WoC contains metadata for nearly all public software projects, including GitHub, GitLab, and BitBucket, as well as numerous smaller platforms and individual forges. Among others, the WoC data are curated to identify and cross-link forks [Mockus, Spinellis, et al. 2020] and different aliases corresponding to unique author IDs [Fry et al. 2020], and include information on all versions of the code (including READMEs), commit activity timelines, and time-stamped package dependencies, which are needed for our analysis. When we collected our data, WoC was at version *V*, which was updated with new repositories found between March 1–30, 2023 and git objects retrieved by mid-May 2023; it included over 209 million repositories (including forks), "deforked" to 131 million unique projects.

Sampling was necessary for two reasons. First, conceptually, it is well known that many public repositories are not meant for software development, but rather contain code dumps, student homework assignments, and other types of artifacts for which questions of longevity and sustainability are less relevant [Kalliamvakou et al. 2016; Munaiah et al. 2017]. Second, pragmatically, at the time of our study it was simply infeasible to process all repository READMEs in WoC using an LLM.

Since there are no universally accepted criteria for identifying "real" software projects among those with public repositories, we applied a set of commonly used size and activity-based heuristics grounded in the literature [Carruthers et al. 2022]. For example, previous research found that project maturity correlates with repository size [Liao et al. 2019] (larger repositories are more likely to correspond to actual software development projects) and having some minimum amount of activity [Ait et al. 2022; He et al. 2024] (code dumps tend to involve only a few commits over a short period); there is also evidence that scientific software is often developed in collaborative settings by multiple authors [Koehler Leman et al. 2020]. In the same spirit, our sampling strategy considers the number of files in the repository (at least 10), the number of commits (more than 300), the number of commit authors (at least 3), the number of months with activity in the project (more than 6 consecutive active months), the last commit date (after November 2018, to avoid less relevant repositories that may have been abandoned long ago), and the programming language(s) identified in the repository (to remove repositories without identifiable code).

Through this process, we filtered the 131 million deforked projects in WoC down to 430,469 projects that met our criteria. We then filtered projects with top-level file names that matched the string "readme" (case insensitive), which captures the modern README.md convention commonly used by GitHub, as well as many other older variations. This resulted in our filtered dataset of 350,308 README files (one per repository) used for LLM classification.

Note that it is possible that some scientific OSS projects did not meet our sampling criteria, and thus did not get a chance to be considered. Similarly, it is possible that some repositories not intended for active development and long-term maintenance remain in our sample despite our relatively strict filters. Our goals here are not to be all encompassing (i.e., identify all scientific OSS) nor perfectly precise (indeed, we expect both goals are futile), but rather to compile a sufficiently large and diverse sample for our subsequent analyses. Importantly, our sampling criteria should not affect the soundness of our subsequent analyses, since we use the same filtering criteria to identify the matching non-scientific OSS repositories we compare against.

Identifying and Classifying The Scientific Software: We used prompt engineering on OpenAI's LLMs to determine whether the repositories in our sample can be considered scientific software, to identify the likely layer they occupy in Hinsen's [2019] scientific software stack (discussed above), and the most likely scientific domain they are intended for. At the time of our data collection, GPT 4 was the top performing model available. However, the number of README files for classification

(350,308) was still larger than we could feasibly process using GPT 4 (in terms of both speed and cost per query). Therefore, we first performed a less precise but more efficient first pass through GPT 3.5 to further narrow down our sample of candidate READMEs, and then made a second pass over this smaller sample using GPT 4 for final classification with expectedly higher accuracy.

Concretely, for our first pass, we instructed GPT 3.5 to determine the repositories' relevance to scientific research based on the contents of their README. After five rounds of prompt engineering, we arrived at a prompt (see Supplementary Materials) asking the LLM to answer six questions about each README, including whether it describes *scientific application software* (defined as "domain-specific science and engineering software" in our prompt), *science support software* (i.e., "software used to support scientific applications or research"), or *research software* (i.e., "software used to generate, process or analyze results that you intend to appear in a publication"); and whether it mentions a publication or research funding related to the software. All three types describe software used for scientific purposes, i.e., can be considered **scientific software** [Kanewala et al. 2014]. Including the latter questions about mentions of papers or funding in the prompt allows us to compare the content-based classification (the most novel component of our approach) with the traditional classification based on explicit links to publications or funding, which has been used in prior work (albeit not with LLMs).

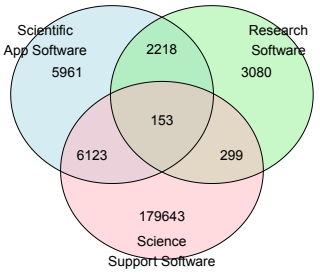

Fig. 2. Overlap between the three scientific-software-related labels assigned by GPT 3.5.

As the definitions of the three types of software have high theoretical overlap, this step unsurprisingly yielded many READMEs tagged with multiple type labels (Figure 2), e.g., 43% of the 14,455 repositories labeled as *scientific application software* were also tagged as *science support software*. Interestingly, while there is also considerable overlap between the *scientific application software* group and READMEs labeled as mentioning a publication or research funding (Figure 3), our approach also identified many candidate scientific repositories without such explicit traces, that may not be discoverable using traditional approaches (we discuss the accuracy of our approach in the Evaluation section below). Since the *scientific application software* group had the highest overlap with the other two types and was of a more manageable size, we used these repositories for our second pass. In addition, we also included the remaining non-overlapping repositories with *mentions of publications or research funding*, for a total of 24,767 repositories (READMEs) out of 342,656 successfully classified files.[3]

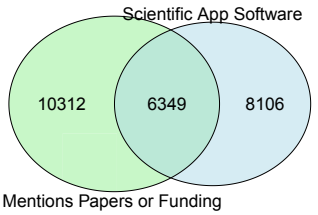

Fig. 3. Overlap between the scientific application software and paper/funding mention labels assigned by GPT 3.5.

In the second step, we prompted GPT 4 (see Supplementary Materials) to classify each of the 24,767 repositories into one of Hinsen's [2019] seven **scientific software stack layers**, ranging from non-scientific infrastructure (layer 1) to non-research software (layer 7). We expected most software would fit into one of three science-related categories (i.e., layer 2: "scientific infrastructure," layer 3: "scientific domain-specific code," and layer 4: "publication-specific code") because of our first filtering pass. However, we kept all seven layers in the prompt as we expected some level of inaccuracy from the first pass. Additionally, as part of the same prompt, we asked the model to associate the repositories with one of 13 predefined **STEM fields**: Astronomy, Biology, Chemistry, Computer Science, Data Science, Earth Sciences, Engineering, Mathematics, Medicine,

[3] 7,654 prompts failed for various reasons, including server error, entity mismatch, rate limit failures, etc.

Neuroscience, Physics, Quantum Computing, and Statistics. While software is also vital beyond traditional scientific disciplines, e.g., in the social sciences and humanities, sustainability challenges in those contexts differ [Tucker 2022], so we restrict the scope of our work to STEM fields to keep the analysis tractable. As above, we provided the contents of each README (truncated if needed to fit the LLM's context window) as part of the pro

Following this final labeling step, we discarded repositories not classified into one of the 13 STEM fields and three stack layers, arriving at our final **SciCat** dataset of 18,247 repositories (Figure 4). As can be observed from the figure, the dataset spans all stack layers and STEM fields, with repositories categorized as Computer Science being the most numerous. For example, these include security analysis tools for smart contracts like MYTHRIL, rendering tools like TUNGSTEN used by graphics researchers, and programming languages for high-performance computing like TAICHI; see dataset in replication package for more examples. Note, only one of the named examples mentions a publication in the README.

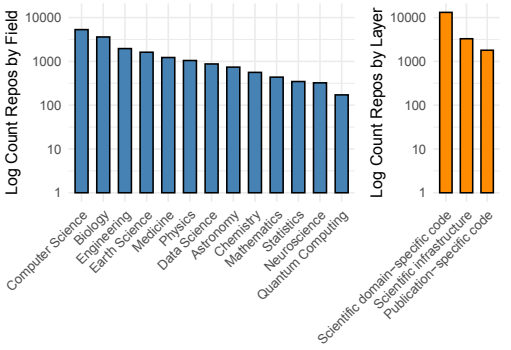

Fig. 4. Distribution of the scientific repositories in our sample, by domain and scientific software stack layer.

4.2 Evaluation

To assess the accuracy of our classification, we used two strategies: (1) manual validation by trained human raters on a sample of the data and (2) cross-validation against existing datasets.

Stratified Sample Validation: We started with a pilot evaluation of the three LLM-generated labels (field, layer, and paper/funding mentions) for a stratified sample of 60 projects. Two human raters classified the same 60 projects on all three dimensions, and engaged in discussions to reconcile differences in their assessments. After resolving disagreements and re-labeling the same projects, the inter-rater reliability (IRR) for the categorization of STEM fields reached 0.733 and for software layers it reached 0.745, indicating "substantial agreement" [Fleiss et al. 1981; Landis et al. 1977]. Most disagreements arose from projects in the machine learning domain, which can be applied across various fields. We also computed the IRR between human raters (after consensus) and the results generated by the LLM: these were 0.714 for field and 0.720 for layer, respectively.

Following the pilot, we expanded the manual validation process to a stratified sample of 468 projects in total, i.e., 12 repositories per field (across 13 fields) and per layer (across 3 layers), randomly sampled within each stratum. The three raters divided the remaining unlabeled projects between them and completed this categorization separately. In the end the IRR between human raters and the LLM was 0.650 for field and 0.624 for layer, both reflecting substantial agreement. For the classification of whether a project mentioned papers or funding, the IRR between the raters and the LLM was 0.467, indicating moderate agreement.

To further illustrate the agreement metrics, Table 1, Table 2 and Table 3 present the F1 score, precision, and recall for classification by field, layer, and mentions of papers or funding, respectively. For mentions of papers or funding, when the response is "Yes", the model achieved a high recall of 0.94 and an F1 score of 0.86, demonstrating strong performance in identifying true positives. However, for the "No" response, the recall and F1 decreased, reflecting a tendency to miss instances where papers or funding were not mentioned. Upon manual review of these instances, we found that the classifier tended to assign credit for the presence of any paper, whereas human raters evaluated whether the papers or citations were directly relevant to the code repository. When they

Table 1. Classification metrics by field

Category	Precision	Recall	F1 Score
Astronomy	0.80	0.90	0.85
Biology	0.79	0.79	0.79
Chemistry	0.77	0.96	0.86
Computer Science	0.86	0.40	0.55
Data Science	0.50	0.53	0.52
Earth Science	0.76	0.87	0.81
Engineering	0.50	1.00	0.67
Mathematics	0.86	0.81	0.83
Medicine	0.68	0.82	0.74
Neuroscience	0.78	0.88	0.82
Physics	0.72	0.76	0.74
Quantum Computing	0.83	0.97	0.90
Statistics	0.76	0.81	0.79
Macro Avg	0.64	0.70	0.66
Weighted Avg	0.74	0.74	0.72

Table 2. Classification metrics by layer

Category	Precision	Recall	F1 Score
Publication-specific code	0.81	0.92	0.86
Scientific domain-specific code	0.83	0.62	0.71
Scientific infrastructure	0.60	0.82	0.69
Macro Avg	0.56	0.59	0.57
Weighted Avg	0.76	0.75	0.74

Table 3. Classification metrics for paper/funding mentions

Response	Precision	Recall	F1 Score
Yes	0.79	0.94	0.86
No	0.79	0.48	0.60

were not, human raters correctly selected "No" as the answer. Overall, these findings highlight the reliability of the classification system but indicate the need for improved prompt engineering and contextual understanding in LLMs to better align with human judgment.

Cross Validation with External Datasets: We also cross-referenced the repositories in SciCat against several existing scientific software datasets from the literature to calculate recall. Note that because SciCat is not intended to be complete and due to the different sampling criteria and different goals of the various existing datasets, recall is not directly computable. For example, Papers with Code contains many replication packages, thus we expect that SciCat's recall values with respect to it would be low even if SciCat was complete, since our sampling criteria are designed to exclude code dumps as much as possible, which many of the replication packages are. Still, we attempt these comparisons to better contextualize our work, by first applying the same filtering criteria we used to identify SciCat candidate repositories before calculating recall values:

- *Journal of Open Source Software (JOSS):* JOSS is a peer-reviewed open access scientific journal that focuses on the publication and dissemination of open-source software from any research discipline [Katz et al. 2018]. After scraping 2,200 repositories from the JOSS website [Katz et al. 2018] and applying our size- and activity-based filters, we were left with 446 unique projects. Of these, 314 are part of our SciCat dataset (70%). A closer examination of the missing ones revealed that many (95) were filtered out during the first classification phase (which involved sampling), with the remaining not classified into one of our predefined fields and stack layers during the second classification phase.
- *National Laboratory Repositories:* Analysis of open repositories from national laboratory GitHub organizations [Nangia et al. 2017] resulted in 155 projects meeting our initial filtering criteria (out of 1,910), 80 of which are in SciCat (47%). As with JOSS, most (71) of the 75 missing national lab projects were filtered out during the first classification stage. Of these 71, further inspection revealed that 59 were categorized as scientific support software or research software only (we did not sample from these; recall Figure 2), and 12 were not marked with any of our three possible flags or with paper/funding mentions (likewise, we did not sample from these). Separately, we inspected a random sample of the national-lab repositories that did not meet our initial sampling criteria, and found many examples we would consider scientific software according to our definition, which suggests that our filters could be further relaxed.
- *Research Software Directory:* The Research Software Directory [Spaaks et al. 2018] is a content management system for research software containing 411 projects, 184 of which are linked

to repositories. After applying our filtering criteria, only 19 repositories matched. Of these, 14 are in SciCat (74%), while 4 were filtered out during the first classification step.

- *Papers with Code:* From the dataset of 227,820 machine-learning projects by Papers with Code, only 8,547 met our size and activity criteria. Of these, 4,965 are in SciCat (58%). As before, most of the missing ones (3,286 projects) were filtered out during the first classification and sampling stage. Inspecting these more closely, we discovered that most (about 73%) were marked as scientific support software or research software without other type labels. Among the remaining missing ones, we saw many examples of websites, aggregators, courses, and course projects, which do not fit our definition of scientific software.
- *NSF Soft-Search:* Brown et al. [2023] identified research projects likely to have produced software while funded by federal grants. Out of 1,520 entries in the dataset, 88 met our initial criteria, of which 64 were included in our final dataset (73%). Among the missing projects, 15 were not labeled as scientific application software during the first classification round and 9 were labeled with layers and fields outside of our scope in the second phase.

In summary, cross-validation revealed varying degrees of overlap between SciCat and external datasets, with coverage rates ranging from 47% to 73%. These results demonstrate the diversity of the SciCat dataset, while highlighting the inherent challenges in constructing a comprehensive and fully representative collection of scientific software repositories. The two main opportunities for future work to increase coverage are relaxing the initial sampling criteria and scaling up the LLM-based processing of README files, whereas we had to resort to sampling.

In general, SciCat can accurately capture relevant projects within the scientific software domain. In addition, more than 40% of the projects in our dataset do not mention any publication or funding source in their README (and thus may be hard to identify otherwise).

4.3 Limitations

SciCat is incomplete because our sampling criteria for candidate repositories from World of Code (WoC), while derived from established repository mining practices and literature, are relatively strict. It also likely misses some scientific repositories hosted individually, since WoC primarily aggregates data from platforms like GitHub and GitLab. Both limit the generalizability of our findings to a subset of the broader scientific software landscape. Finally, SciCat is to some extent inaccurate (although our evaluation results above show that accuracy is high) because of the LLM's imperfect ability to label the README files. However, all these limitations can be considered *accidental*, i.e., we expect that they can be reduced substantially as LLMs improve, and with more computational resources to process more repositories in and outside WoC. More generally, the use of LLMs introduces various known and yet unknown risks. We, therefore, heavily rely on manual validation of LLM-based classification via stratified sampling.

At the same time, a noteworthy *essential* limitation of SciCat is that its construction relies on repository README files. While, as we show above, this is a good choice when READMEs are present and well written, across public repositories in WoC README files are also often incomplete, vague, or poorly maintained, which can lead to misclassification or the exclusion of important projects. This design choice may result in underrepresentation, especially for projects that do not follow standard documentation practices or offer insufficient details about their scientific focus in their README. Therefore, we expect that a promising direction for future research is to combine a repository content-based approach like ours with an approach based on mining software mentions in research papers [Howison and Bullard 2016].

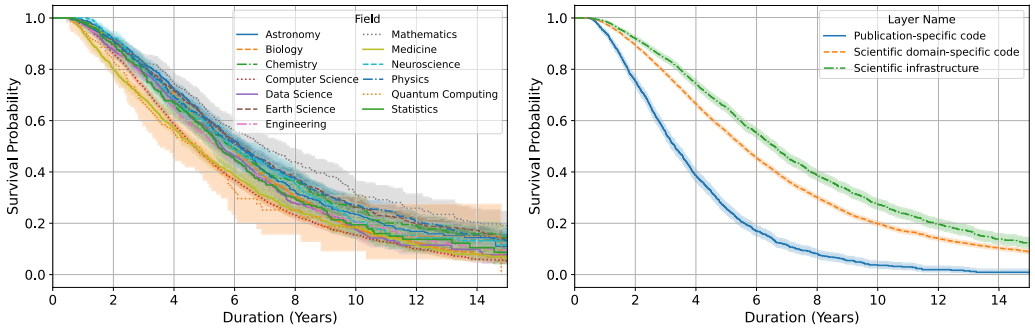

Fig. 5. Kaplan-Meier survival curves by field (left) and layer (right).

5 RQ2: Modeling the Longevity of Scientific OSS Projects

Next we study the longevity of scientific software repositories in SciCat, understood as the time span from the project's earliest to its latest commit. Concretely, we label projects as ***inactive* (abandoned)** if they had **no commits in the last six months** prior to the end of our data collection,[4] and record the date of the last commit as the abandonment date. The commit data are right censored, i.e., projects still active less than six months prior to the end of our observation period may have become abandoned shortly thereafter, or may have remained active to this day (there is more uncertainty about the "abandonment event" at the right end of the observation period). Therefore, we use survival analysis to model the variation in the time to the abandonment event, a standard technique for modeling right-censored time-to-event data [Moore et al. 2016].

To compare typical rates of survival across subpopulations, we compute the restricted mean survival time (RMST), which is the mean of the survival time up to some time horizon, i.e., the area under the curve up to a certain point) [Royston et al. 2013; Uno et al. 2014]. This is considered a useful summary measure for comparing different survival curves over a given time period. Note that having a time horizon limits the impact of long tails (outliers) on the result; for the purpose of our study, we consider a 15 year window.

5.1 RQ2a: Variation Across Fields and Layers

Methods: We start with an exploratory survival analysis of SciCat projects broken down by field and stack layer, in response to RQ2a. We use the non-parametric Kaplan-Meier (KM) estimator, a robust way to estimate survival functions from time-to-event data. We will test hypotheses about which factors are associated with variation in these survival probabilities in RQ2b below.

Results: The Kaplan-Meier survival plots in Figure 5 offer insights into the longevity of scientific software projects across different domains and layers within the software stack. The plot on the right shows survival by layer and compares publication-specific code, domain-specific code, and infrastructure layers. The sharp decline in survival probability for publication-specific code in the early years highlights its vulnerability to obsolescence, likely due to its niche use and limited applicability. Scientific domain-specific code fares slightly better but still follows a rapid decline, reflecting that while it may have broader usage than publication-specific code, it is still prone to technological shifts or evolving standards. In contrast, scientific infrastructure, likely meant to serve more general and long-term needs, does indeed demonstrate a slower, more gradual decline,

[4]We include robustness checks for different operationalizations in our Supplementary Materials. The coefficient estimates vary slightly, as expected, but their direction is consistent.

indicating better longevity. This suggests that broad-purpose infrastructure projects tend to persist longer as a result of their foundational role in supporting multiple scientific domains.

The left plot highlights a wide range of survival probabilities depending on the field. Fields such as Astronomy, Biology, and Chemistry exhibit relatively steady declines over time, perhaps reflecting their established nature and slower pace of technological change. Fields like Computer Science and Quantum Computing experience sharper drops in survival probability, likely due to the rapid advancements and high turnover. Medicine and Neuroscience show better longevity, maintaining higher survival probabilities for longer periods, possibly due to the ongoing need for research and development there. These survival patterns highlight the variable lifespan of projects in different domains and layers, indicating the complexity of factors that contribute to project longevity.

Over a 15 year period, the average restricted survival time of scientific OSS projects in our study is 6.44 years. Survival rates in our data varies by field, with mathematics representing the longest average survival (7.91 years) and computer science the shortest (5.74 years). The application layer has significantly shorter survival times than domain-specific (3.94 years vs. 6.52 years), with infrastructure having the greatest average longevity (7.44 years).

5.2 RQ2b: Longevity or Survival Analysis:

Methods: We run a Cox proportional-hazards multiple regression analysis [Kleinbaum et al. 2012] of SciCat projects, which allows us to simultaneously model the relative importance of different factors in explaining the variation in the risk of projects becoming abandoned, according to our six-month inactivity definition. The semiparametric nature of the Cox model [1972] also allows us to explore the relative impact of various factors on project longevity without making strict assumptions about the time-to-event distribution. This type of regression for survival analysis is standard in numerous domains, including software engineering [Valiev et al. 2018; M. Zhou and Mockus 2011, 2012; M. Zhou, Mockus, et al. 2016].

In our model we include a number of control variables corresponding to known factors associated with project longevity in the traditional OSS literature, plus several independent variables following our discussion in Section 2.2 of factors that may impact the longevity of scientific software differently compared to traditional OSS. Table 4 lists all the control and independent variables we measured, together with the rationale for their inclusion and our hypotheses, where possible, for their effects. All these variables were calculated using WoC data.

The controls include measures of **software size**, **project team size**, and **community size**, as well as measures indicative of the project's position in the global software supply chain based on its **upstream and downstream dependencies**. The independent variables include the three we inferred in Section 4, **field**, **layer**, and having **mentions of publications or research funding**, which allows us to formalize the exploratory analysis in RQ1 (field and layer) and contextualize our results in the scarce existing literature (mentions paper or funding). We also test for differences between **programming languages**, as their usage tends to differ substantially between scientific fields. Moreover, we test for effects associated with **academic and government participation**. Collaboration between academic and non-academic participants can enrich an open source project, fostering continuous innovation and the transfer of cutting-edge knowledge [Colazo et al. 2010; Fitzgerald 2006; R. A. Ghosh 2005; Lakhani et al. 2003; Mockus, Fielding, et al. 2002]; at the same time, it can be challenging for scientists and engineers to collaborate [Sun et al. 2024], and there may be greater-than-average turnover induced by the academic involvement because of rotating graduate students and postdocs [Valiev et al. 2018]. Government participation may come with increased stability and access to resources [Reddy et al. 2002], which could translate to improved software longevity and impact [Ribeiro et al. 2020; Schweik et al. 2012; Sharma et al. 2007; Weiss

Table 4. Control (top half) and independent (bottom half) variables measured for our Cox regression.

Variable	Description	Rationale / Hypotheses
Num. Core Authors	Count of authors with commit counts above the 80th percentile	While projects may have many *peripheral* contributors, the *core* team, which tends to have the most influence over a project's future, can be very small [Coelho et al. 2017; Joblin et al. 2017].
Num. Commits	Count of commits	Captures project size and historical activity. Larger projects with more development momentum tend to survive longer [Alves et al. 2018].
Community Size	Count of repository forks	Similarly to Num. Authors, indicates attention to the project and maintenance effort available [S. Zhou et al. 2019].
Earliest Commit Year	Year of earliest commit	Controls for possible environmental changes in the OSS ecosystem over time [Alves et al. 2018].
Num. Defined Packages	Count of packages defined (e.g., `package` statements in Java) in the repository	Software intended for reuse is more likely to be packaged for distribution [Decan et al. 2019] and intended reusability should increase longevity.
Upstream Package Ratio	Fraction of packages used (e.g., as `import` statements in Java) that are defined in "upstream" repositories	This is a proxy measure for the amount of external upstream dependencies.[a] Projects with more upstream dependencies face more breaking changes [Bogart et al. 2021], package abandonment [Miller, Kästner, et al. 2023], and other risks. They also tend to accumulate more technical debt, leading to maintenance burdens that increase the risk of abandonment [Cogo et al. 2021; Dey et al. 2019].
Num. Downstream Projects	Count of "downstream" projects that import packages defined in the repository	More downstream dependents indicate a larger user base and possible lock-in effects, which may motivate maintainers to keep the project active [Dey et al. 2019; Valiev et al. 2018].
Language	Dominant language in the repository, based on filename extensions	Much scientific software is developed in languages like Python and R. Different language ecosystems tend to have different culture and norms [Bogart et al. 2021].
Layer	Inferred software stack layer (Section 4)	Software that plays an infrastructural role for science is expected to be longer-lived [Hinsen 2019].
Field	Inferred STEM field (Section 4)	[Hasselbring et al. 2020] report significant differences in longevity between research software in different fields, among software with explicit links to publications.
Mentions Paper or Funding	True if the README was labeled as mentioning papers/funding (Section 4)	The combination of financial support and academic recognition should help attract new contributors and funding opportunities [Howison, Deelman, et al. 2015; Koehler Leman et al. 2020; Strasser et al. 2022].
Has Academic Participants	True if any core authors had a .edu-domain email	Captures obvious academic involvement, which can impact software longevity in many ways [Koehler Leman et al. 2020; Valiev et al. 2018].
Has Government Participants	True if any core authors had a .gov-domain email	Captures obvious government involvement, such as US National Labs. Such involvement may indicate professional (research) software engineers [Koehler Leman et al. 2020; Schwartz et al. 2024].

[a]Given the multilingual nature of WoC, these cannot be counted directly.

2005]; there are many examples of long-lasting government-funded projects lasting years or even decades [Bozeman 2000; Nelson 1993].

As with all statistical models, we want to establish a relationship between the response and predictors, as statistical models do not demonstrate causal relationships. One of the first steps in statistical modeling is to avoid highly correlated predictors, as they increase the errors of the coefficient estimates and often lead to hard-to-interpret results. To this end, we computed pairwise rank correlation coefficients between all our variables and eliminated number of authors, where correlation with core authors was above 0.65. As multiple comparisons inflate the risk of false discovery, we applied the conservative Bonferroni correction [Sedgwick 2012].

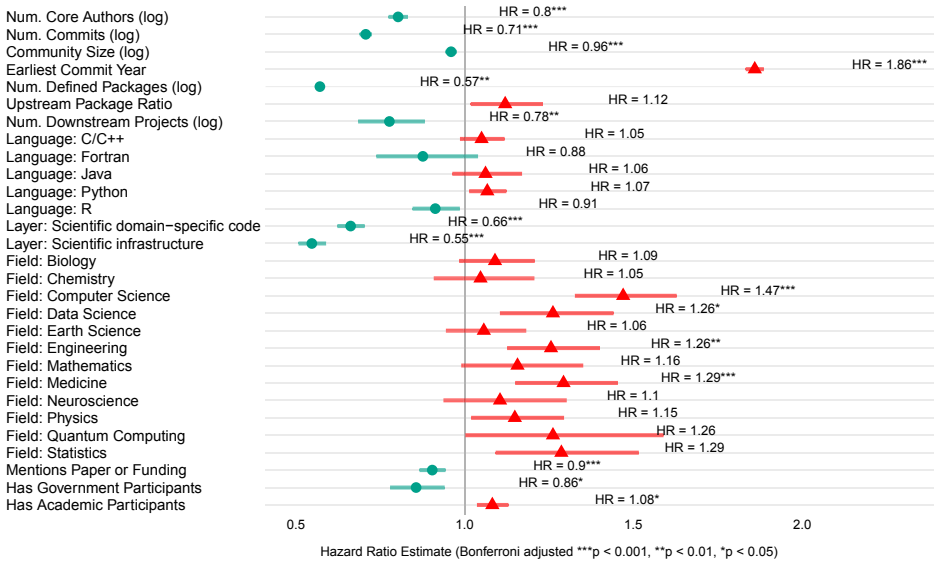

Fig. 6. The Cox proportional hazard regression model for scientific software. Out of 18,244 projects, 11,631 had their last commit more than six months before the data was collected.

Results: Next, we present the findings of our Cox proportional-hazards regression analysis exploring the relationship between the survival times of scientific software repositories in our SciCat dataset and the predictor variables we discussed above. Figure 6 visualizes the estimated *hazard rates* (HR) (exponentiated Cox model coefficients). It is crucial to note that in the Cox proportional-hazards model, *HR above one indicates an increased likelihood of project cessation, not survival.*

Looking first at our control variables, we observe effects largely consistent with prior work. Larger team size (**number of core authors**), project size (**number of commits**), and **community size** all indicate lower risk of project dormancy ($HR = 0.8, 0.7, 0.96$, respectively; for example, increasing the log of the core author count by 1 decreases the hazard rate by 20%), consistent with the understanding that strong and active community participation enhances OSS projects' longevity. The **number of packages defined** in the project and the **number of downstream dependents** also increase longevity ($HR = 0.57$ and 0.78), indicating that more reusable projects serving critical roles in the broader ecosystem tend to be maintained for longer periods.

Note that the percentage of projects with at least one downstream dependent varies with layer in the expected fashion: 11%, 26%, and 37% for publication-specific, domain-specific, and scientific infrastructure, respectively. That is, as we transition from specific publication code to more foundational scientific infrastructure, projects are more likely to have downstream dependencies, as expected for infrastructure. Also note that not all infrastructure components have downstream dependencies. For example, runtime environments, workflows, and other tools categorized as infrastructure may not always provide clear indications of their use in downstream projects. It is notable, that the infrastructure projects are longer-lived even after adjusting for downstream dependents.

While the 95% confidence interval for the **upstream project ratio** does not overlap with 1, the effect (HR of 1.12) is no longer statistically significant after the Bonferroni correction. If, however, we consider that the correction is very conservative and the previous consistent findings in the literature, an increase in **upstream project ratio** is likely associated with an increase in abandonment risk. Thus, the network of upstream dependencies, when large relative to the focal project's own modules, is associated with shorter project lifespan.

Surprisingly, we note a strong effect associated with the **earliest commit** year: the more recently the project started, the higher the risk of abandonment ($HR = 1.86$), indicating perhaps a drastic change in open source culture over the years.

Moving on to our main predictors, the model is consistent with the observation from Section 4 that scientific infrastructure projects have the lowest abandonment hazard among the three **layers** ($HR = 0.55$), followed by domain-specific code ($HR = 0.66$). The coefficient for **scientific domains** represent that field's contrast with Astronomy (the baseline level, chosen alphabetically). Scientific fields such as Computer Science, Data Science, Engineering, and Medicine have HR above one, indicating shorter lifespans than comparable projects in Astronomy. While Mathematics had the largest marginal survival rate in the exploratory analysis above in Section 4, when adjusting for other covariates it is trending worse than Astronomy (HR of 1.16 or about 16% more likely to be abandoned than a comparable Astronomy project), albeit not statistically significantly after the Bonferroni correction. Our analysis, post adjustment, also did not find statistically significant differences between Astronomy and Biology, Chemistry, Earth Science, Neuroscience, Quantum Computing, and Statistics. No differences among **programming languages** could be observed except for R (which had HR of 0.91) that was no longer significant after adjustment.

Projects **mentioning funding or scientific publications** show longer lifespans ($HR = 0.9$) than comparable projects without such mentions. **Government participation** reduces the hazard rate ($HR = 0.86$), which implies that government involvement brings stability and long-term viability to a project. Interestingly, **academic participation** is linked to a slight increase in hazard ($HR = 1.08$), possibly due to the often short-term or cyclical nature of academic commitments.

Like other OSS, scientific software with fewer upstream and more downstream dependencies, larger communities, and more core contributors tend to live longer. Government participation and absence of academic participation, indications of funding, and mentions of scientific publications also tend to increase longevity. The lower levels of the stack, as expected, survive longer.

6 RQ3: Comparing Longevity of Scientific Software with the Broader OSS Ecosystem

For our final analysis we compare the longevity of the scientific software repositories in SciCat to a matched group of non-scientific OSS repositories.

Methods: To estimate if the non-scientific software has different survival times, we need to conduct a "natural experiment". If we select a random sample of non-scientific software, we will be comparing not longevity but precursors of longevity, such as activity. The goal for the sample selection process is, therefore, to be noninformative or ignorable [Little et al. 1987] by which we mean that the resulting sample and population are compositionally similar on the set of covariates that explain treatment effect (survival time) variability [Stuart et al. 2011; Tipton 2013].

For scientific software, we select all projects and we match the distribution of scientific projects to that of non-scientific using the following stratified sampling procedure: We divided our scientific sample based on three variables, namely number of commits, number of authors, and earliest commit date. These three give us a gauge for activity, community size, and time, respectively. We divided the number of commits into four bins, the first bin being projects with less than 750 commits, the second bin with commits between 750 and 1,800, the third bin between 1,800 and 5,000, and the last bin projects with more than 5,000 commits. We did the same for number of authors with first bin less than 10 authors, second bin 10 to 25 authors, third bin 25 to 60 and last bin more than 60 authors. The logic for choosing these thresholds was based on our data distribution, so that each bin has roughly half the count of the previous bin. That is, the first bin in each category corresponds roughly to 54% of the data, the second bin to 27%, third to 13% and fourth to 6%. Finally, we divided

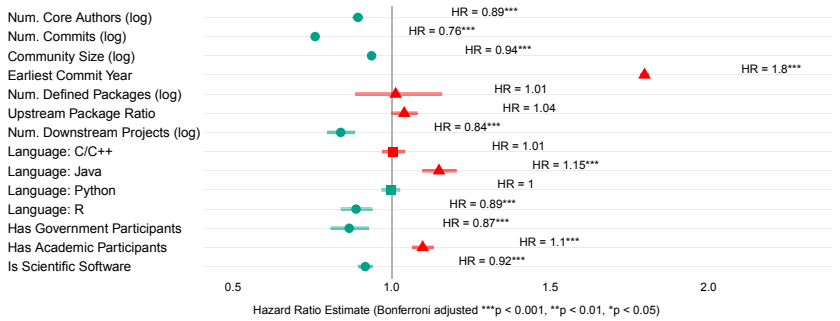

Fig. 7. The proportional hazard regression model for scientific and matched non-scientific software. Out of 54,710 projects, 41,069 had last committed more than six months before the data was collected.

our data based on their earliest commit date into three bins: projects before 2016, between 2016 and 2018, and the third bin with anything after 2019. These thresholds were chosen so that the bins have roughly the same number of projects. With the explained criteria, we created 48 bins (4*4*3) and for each bin, we randomly sampled twice the number of scientific projects from non-scientific projects in OSS using WoC's MongoDB projects database. That is, if, for example, there were 500 projects in a bin with 750-1800 commits, 25-60 authors and after 2019, we randomly sampled 1,000 nonscientific projects with these exact criteria for number of commits, authors, and earliest commit time. Having 18,247 scientific projects, we sampled 36,494 non-scientific projects and created a total sample of 54,741 scientific projects and comparable non-scientific projects.

We then estimated a survival model with the specification from RQ2b, excluding the domain and layer variables which are not available for non-scientific repositories, and adding a categorical indicator for scientific software to distinguish between the two groups.

Results: Figure 7 visualizes the estimated hazard rates for various factors and their statistical significance. Given the larger sample and a different set of predictors, the estimated coefficients, as expected, differ slightly from the model used in RQ2. Specifically, the model can better discriminate among **programming languages** with Java projects having shorter and R projects having longer lifetimes. Eliminating science-specific factors from the model makes the number of packages defined in the project no longer statistically significant. The direction of the effects stays the same and their interpretation was provided previously. The key result of this analysis, though, is given by the population comparison indicator (**Is Scientific Software**), which is has negative $HR = 0.92$ (or about a 8% lower abandonment risk), indicating longer survival times.

Scientific OSS projects survive longer than counterparts of the same size from the same era.

7 Threats to Validity

We discussed limitations of our dataset above in Section 4.3. Here we discuss additional threats to the validity of our analyses in Sections 5 and 6.

Construct Validity: The most notable threat comes from our operationalization of project abandonment which which may overlook "feature-complete" projects that no longer need frequent updates but remain valuable and functional. It is also possible that OSS maintainers return to projects after long breaks [Calefato et al. 2022], which implies that some of the projects we label as abandoned may resume activity eventually. To assess this threat, we experimented with different thresholds of abandonment, up to 24 consecutive months of inactivity (see Supplementary Materials), with no qualitative change in our findings; that is, breaks are unlikely to invalidate our results.

Internal Validity: First, to ensure the soundness of our regression analysis, we started by modeling a wide range of variables to minimize potential biases from confounding factors, and then reduced that number by excluding highly correlated predictors to improve accuracy of the estimates and to simplify interpretation. We also corrected data errors, such as times in the future, and checked for outliers or unusual patterns of residuals to minimize undue influence of single observations on the overall model. In addition, we used transformations where appropriate to reduce the influence of outliers or to improve interpretability of the results.

Second, since the substantially higher longevity of scientific projects in our sample compared to non-scientific ones is a surprising result, we conducted an additional robustness check, analyzing GitHub star counts as a measure of popularity and attractiveness to contributors [Borges et al. 2018; Fang et al. 2022]; one can expect that more popular projects are less likely to become abandoned. Since we did not explicitly match projects on stars (RQ3), there is a risk that our estimated longevity differences between scientific and non-scientific projects can be attributed to differences in star counts. Therefore, we attempted to control for stars in our regression, but our community size predictor (number of forks) had almost 90% correlation with star count. In conclusion, we exclude this possible alternative explanation for our result.

However, other alternative explanations may still exist as our study is observational. For example, even though when sampling to select comparable non-scientific projects, projects were matched based on size, activity, and contributors, there may be other predictors of longevity we did not model or match between samples, such as project goals and maintenance practices, the lifecycle of research projects, and changing research priorities. More research is needed to investigate these.

External Validity: While this study spans multiple scientific domains, it may not fully represent all areas of research, particularly interdisciplinary fields that use non-standard terminologies. Niche scientific communities with distinct documentation practices may also be underrepresented, as would fields outside of the 13 we considered, affecting the generalizability of our conclusions. Thus, we advise caution when extending them to other software ecosystems. However, this limitation does not diminish the relevance of the findings within the intended scope.

Finally, our study primarily focuses on open-source scientific software with a substantial history of activity, which may overlook important sustainability patterns in less active repositories, such as feature-complete or archival projects. We also cannot make claims about proprietary scientific software, which operates under different incentive structures, emphasizing profitability and internal funding, rather than community-driven sustainability. As a result, the longevity patterns observed in this study may not fully translate to proprietary or industry-developed software. While findings should not be extrapolated to proprietary or archival software without caution, they provide valuable insights into longevity within scientific software ecosystems with community participation.

8 Discussion and Implications

Here we highlight implications of our work for research, scientific software, and policymakers.

Enhanced Empirical Access To Scientific OSS: As a major contribution of our study, we showed that a carefully-prompted LLM can identify scientific open-source software based on the contents of the repository README file even when no explicit mentions of research publications or funding are present. Our approach can both identify many known examples of scientific software, and also discover many new ones. This adds a new tool to a space where better traceability has long been a concern [Howison and Bullard 2016], where contemporary efforts to improve traceability exist (using explicit signals such as mentions of the software in publications and of publications in software) [Istrate et al. 2022; Wattanakriengkrai et al. 2022], and where there are still many open questions of interest to practitioners, funders, policy makers, and researchers. We view our

LLM-based approach as complementary. We recommend more future work evaluations of when and how an LLM-based approach like ours, which may be less precise but have higher recall than ones based on explicit links, could be combined with the latter to maximize accuracy.

There are many open questions about scientific software that our publicly-available SciCat dataset could help answer at scale. Researchers could use our dataset to investigate unexplored dimensions of scientific software sustainability, such as the impact of governance models, contributor networks, the evolution of funding sources, and the relationship between sustainability and scientific impact[Barabasi 2024]. Meanwhile, we believe our quantiative study lays the groundwork for future qualitative ones that could be based on a theory-relevant sampling of the curated list of projects and/or to further develop the generalizability and applicability of these findings across fields. Of particular interest could be the social sciences and humanities, where software sustainability challenges differ due to funding constraints, project-based development, and the dependence on individual researchers for maintenance [Tucker 2022].

In the process of validating and cross-referencing our collection with multiple external sources we found a number of drawbacks to both manually and automatically curated resources. While extracting references to publications can be easily automated, a large portion of scientific software does not include such references. Furthermore, a reference to a publication, even if it is provided, often may not be related to the software itself and may not be an indication that a repository contains scientific software. Manually curated scientific software collections include projects that are of educational nature, like tutorials or documentation. Furthermore, we found most of the projects in such manually curated collections to be inactive.

Supporting Scientific OSS Development: Our analysis of SciCat projects revealed substantial differences in longevity between scientific OSS at different layers of the software stack, with projects labeled *scientific infrastructure* (i.e., those expected to be better maintained) being indeed the longest lasting. It also revealed a positive association between traditional OSS success factors (such as community involvement and having a large and active user base) and longevity. We interpret this as good news, in the sense that the same mechanisms that contribute to the success of non-scientific open-source software also seem to impact scientific OSS in predictable ways, despite the different challenges that scientific OSS are subject to. This adds some evidence to suggest that investing in scientific software can yield long-term benefits for funding agencies, and that programs such as the US National Science Foundation's Pathways to Enable Open-Source Ecosystems (POSE) are worthwhile, as they support often-overlooked but critical factors such as community participation.

On a practical note, our survival model confirmed that reusability of scientific software is associated with greater longevity – both the software labeled infrastructure and that with many downstream dependent projects tend to be maintained longer. This suggests that scientific software developers should aim to design and package their tools in a way that facilitates reuse by other projects, increasing their utility to broader audiences and, in turn, prolonging their lifespan.

The longevity was strongly associated with downstream dependencies, much more so than with upstream projects. It demonstrates that carefully measuring usage in downstream projects is important, yet nontrivial task (in contrast to the identification of upstream packages). We relied on unique capabilities of the WoC research infrastructure [Ma et al. 2021] to obtain these measures.

Our analysis revealed two surprising findings. First, government participation was linked to increased longevity, while academic involvement was linked to reduced longevity, highlighting the need for further research on sustaining scientific software in these different contexts. Second, projects that mentioned publications or funding in their README tended to last longer.

One possible explanation is visibility: software clearly identifiable as scientific may attract more contributors on platforms like GitHub, where purpose influences engagement [Huang et al. 2021].

Such projects may also be more likely to draw attention from funders and other scientists, increasing their chances of being cited or mentioned in scholarly work.

We recommend future research compare the success of highly visible scientific software (e.g., those in known datasets or frequently cited) with less prominent but equally important infrastructure projects [Nesbitt et al. 2024].

Scientific OSS Is Less Likely To Become Abandoned Than One Might Think: Our most unexpected finding is that scientific software in SciCat tends to be longer-lived than a matched sample of non-scientific OSS projects. This suggests that while concerns about scientific software abandonment are valid in some cases, they may be overstated when applied broadly.

Although this result holds across different definitions of abandonment and robustness checks (see above), we urge caution in generalizing it beyond our sample, which leans toward larger, collaboratively developed projects rather than smaller or individually developed ones. More research is needed to understand whether scientific software outside SciCat behaves differently.

Still, our findings raise the question of whether scientific software benefits from unique external factors, like academic incentives and funding, that are short-term and unpredictable but largely absent in non-scientific OSS. Ironically, this suggests non-scientific projects might benefit from adopting certain practices common in the scientific realm: tying development to research or business outcomes, promoting use in publications or white papers, and seeking external funding may all help improve sustainability.

9 Conclusion

In our study, we produced and analyzed a large ~18K) and diverse (13 scientific domains) dataset of collaboratively developed OSS scientific software projects as a vehicle for studying scientific software sustainability; we used LLMs with curated prompts to automatically classify project READMEs, checking our results through both manual validation and comparisons with prior datasets and online collections. We analyzed scientific software longevity across this population and compared scientific projects with general open-source ones. We find that scientific software tends to be longer-lived, and factors like fewer upstream and more downstream dependencies, larger communities, government involvement, and the absence of academic participation were linked to increased longevity.

This curated collection helps to further research scientific software and results in findings that can inform science policy related to software development. Although our study offers some insight into the longevity of open scientific software, more work is needed. Future enhancements to our collection could enable researchers to assess the role of funding and institutional support, to include the humanities and social sciences, and support the study of less active projects.

10 Data Availability

All data and materials used in this study are publicly available in the replication package.[5]

Acknowledgments

This work was partially supported by the National Science Foundation NSF Awards 2120429, 1901102, and 2317168. This manuscript has been authored by UT-Battelle, LLC, USA under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid up, irrevocable, worldwide license to publish or reproduce the published form of the manuscript, or allow others to do so, for U.S. Government purposes. The DOE will provide public access to these results in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

[5] https://doi.org/10.5281/zenodo.15278783

References

Carl S Adorf, Vyas Ramasubramani, Joshua A Anderson, and Sharon C Glotzer. 2018. "How to professionally develop reusable scientific software—and when not to." *Comput. Sci. Eng.*, 21, 2, 66–79.

Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2022. "An Empirical Study on the Survival Rate of GitHub Projects." In: *Int'l Conf. MSR (MSR)*, 365–375.

Carina Alves, Joyce Oliveira, and Slinger Jansen. 2018. "Understanding governance mechanisms and health in software ecosystems: A systematic literature review." In: *Int'l Conf. Enterprise Inf. Systems (ICEIS)*. Springer, 517–542.

Taylor Arnold and Lauren Tilton. 2024. *Humanities Data in R: Exploring Networks, Geospatial Data, Images, and Text*. (2nd ed.). Springer.

Albert-Laszlo Barabasi. 2024. *NSF Award "SoS: BIO: Evaluating the Impact of Biomedical Tools and Methods"*. https://reporter.nih.gov/project-details/11120817. (2024).

Michelle Barker and Daniel Katz. 2022. "Overview of research software funding landscape." *Zenodo*.

Rob Baxter, N Chue Hong, Dirk Gorissen, James Hetherington, and Ilian Todorov. 2012. "The research software engineer." In: *Digital Research Conf., Oxford*, 1–3.

Christoph Becker, Ruzanna Chitchyan, Leticia Duboc, et al.. 2015. "Sustainability design and software: The Karlskrona manifesto." In: *Int'l Conf. Software Eng.* Vol. 2. IEEE, 467–476.

Chris Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2021. "When and how to make breaking changes: Policies and practices in 18 open source software ecosystems." *ACM Trans. Softw. Eng. Methodol.*, 30, 4, 1–56.

Massimiliano Bonomi. 2019. "Promoting transparency and reproducibility in enhanced molecular simulations." *Nature Methods*, 16, 8, 670–673.

Hudson Borges and Marco Tulio Valente. 2018. "What's in a GitHub star? Understanding repository starring practices in a social coding platform." *J. Syst. Softw.*, 146, 112–129.

Barry Bozeman. 2000. "Technology transfer and public policy: a review of research and theory." *Research Policy*, 29, 4, 627–655.

Alys Brett, Michael Croucher, Robert Haines, et al.. 2017. "Research software engineers: State of the nation report 2017."

Eva Maxfield Brown, Lindsey Schwartz, Richard Lewei Huang, and Nicholas Weber. 2023. "Soft-Search: Two Datasets to Study the Identification and Production of Research Software." In: *Joint Conf. Digital Libraries (JCDL)*. IEEE, 228–231.

Fabio Calefato, Marco Aurelio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. 2022. "Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub." *Empir. Softw. Eng.*, 27, 3, 76.

Juan Andrés Carruthers, Jorge Andrés Diaz-Pace, and Emanuel Agustín Irrazábal. 2022. "How are software datasets constructed in Empirical Software Engineering studies? A systematic mapping study." In: *Euromicro Conf. Software Eng. and Advanced Applications (SEAA)*. IEEE, 442–450.

Vinton G Cerf. 2024. "The Boundary Hunters." *Communications of the ACM*, 67, 8, 5–5.

Kaylea Champion and Benjamin Mako Hill. 2021. "Underproduction: An approach for measuring risk in open source software." In: *Int'l Conf. Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 388–399.

Adam S Charles, Benjamin Falk, Nicholas Turner, et al.. 2020. "Toward Community-Driven big Open brain science: open big data and tools for structure, function, and genetics." *Annu. Rev. Neurosci.*, 43, 441–464.

InduShobha Chengalur-Smith, Anna Sidorova, and Sherae L Daniel. 2010. "Sustainability of free/libre open source projects: A longitudinal study." *J. Assoc. Inf. Syst.*, 11, 11, 5.

Jailton Coelho and Marco Tulio Valente. 2017. "Why modern open source projects fail." In: *Int'l Conf. Foundations of Software Eng. (FSE)*. ACM, 186–196.

Filipe R Cogo, Gustavo A Oliva, and Ahmed E Hassan. 2021. "Deprecation of packages and releases in software ecosystems: A case study on npm." *IEEE Trans. Softw. Eng.*, 48, 7, 2208–2223.

Jeremy Cohen, Daniel S Katz, Michelle Barker, et al.. 2020. "The four pillars of research software engineering." *IEEE Software*, 38, 1, 97–105.

Jorge A. Colazo and Yulin Fang. 2010. "Following the sun: Temporal dispersion and performance in open-source software project teams." *J. Assoc. Inf. Syst.*, 11, 11, 684–707.

D. R. Cox. 1972. "Regression models and life-tables." *J. R. Stat. Soc.*, 34, 2, 187–220.

Alexandre Decan, Tom Mens, and Philippe Grosjean. 2019. "An empirical comparison of dependency network evolution in seven software packaging ecosystems." *Empir. Softw. Eng.*, 24, 1, 381–416.

Tapajit Dey, Yuxing Ma, and Audris Mockus. 2019. "Are Software Dependency Supply Chain Metrics Useful in Predicting Change of Popularity of NPM Packages?" In: *Int'l Conf. Predictive Models and Data Analytics in Software Eng.* ACM, 1–10.

Anshu Dubey, Jared O'Neal, Klaus Weide, and Saurabh Chawdhary. 2020. "Distillation of best practices from refactoring flash for exascale." *SN Computer Science*, 1, 4, 1–9.

George Dyson. 2012. *Turing's cathedral: the origins of the digital universe*. Vintage.

Claudia Eitzen. 2020. "Research Software-Publication and Sustainability." Ph.D. Dissertation. Kiel University.

Hongbo Fang, Hemank Lamba, James Herbsleb, and Bogdan Vasilescu. 2022. ""This is damn slick!" Estimating the impact of tweets on open source project popularity and new contributors." In: *Int'l Conf. Software Engineering (ICSE)*, 2116–2129.

Stuart Faulk, Eugene Loh, Michael L Van De Vanter, Susan Squires, and Lawrence G Votta. 2009. "Scientific computing's productivity gridlock: How software engineering can help." *Comput. Sci. Eng.*, 11, 6, 30–39.

Daniela Feitosa, Christina von Flach, and Joenio Costa. 2023. "Understanding practices and challenges of developing sustainable research software: A pilot interview." In: *OpenScience Worksh.* Universidade Federal da Bahia.

Brian Fitzgerald. 2006. "The transformation of open-source software." *MIS Quarterly*, 30, 3, 587–598.

Joseph L Fleiss, Bruce Levin, Myunghee Cho Paik, et al.. 1981. "The measurement of interrater agreement." *Statistical methods for rates and proportions*, 2, 212-236, 22–23.

Anne Fouilloux, Jean Iaquinta, Alok Kumar Gupta, et al.. 2023. "Building on Communities to Further Software Sustainability." *Comput. Sci. Eng.*, 25, 3, 84–88.

Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2020. "A Dataset and an Approach for Identity Resolution of 38 Million Author IDs extracted from 2B Git Commits." In: *Int'l Conf. MSR (MSR)*.

Jonas Gamalielsson and Björn Lundell. 2014. "Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved?" *J. Syst. Softw.*, 89, 128–145.

Daniel Garijo, Miguel Arroyo, Esteban Gonzalez, Christoph Treude, and Nicola Tarocco. 2024. "Bidirectional paper-repository tracing in software engineering." In: *Int'l Conf. MSR (MSR)*. IEEE, 642–646.

Rishab Aiyer Ghosh. 2005. "Understanding free software developers: Findings from the FLOSS study." In: *Perspectives on Free and Open Source Software*. MIT Press, 23–46.

Michael Haider, Michael Riesch, and Christian Jirauschek. 2021. "Realization of best practices in software engineering and scientific writing through ready-to-use project skeletons." *Optical and Quantum Electronics*, 53, 10, 1–17.

Wilhelm Hasselbring, Leslie Carr, Simon Hettrick, Heather Packer, and Thanassis Tiropanis. 2020. "Open source research software." *IEEE Computer*.

Runzhi He, Hengzhi Ye, and Minghui Zhou. 2024. *Revealing the value of Repository Centrality in lifespan prediction of Open Source Software Projects*. (2024). https://arxiv.org/abs/2405.07508 arXiv: 2405.07508 [cs.SE].

Konrad Hinsen. 2019. "Dealing With Software Collapse." *Comput. Sci. Eng.*, 21, 3, 104–108.

James Howison and Julia Bullard. 2016. "Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature." *J. Assoc. Inf. Sci. Technol.*, 67, 9, 2137–2155.

James Howison, Ewa Deelman, Michael J McLennan, Rafael Ferreira da Silva, and James D Herbsleb. 2015. "Understanding the scientific software ecosystem and its impact: Current and future measures." *Research Evaluation*, 24, 4, 454–470.

James Howison and James D Herbsleb. 2011. "Scientific software production: incentives and collaboration." In: *ACM Conf. Computer Supported Cooperative Work (CSCW)*, 513–522.

Yu Huang, Denae Ford, and Thomas Zimmermann. 2021. "Leaving my fingerprints: Motivations and challenges of contributing to OSS for social good." In: *Int'l Conf. Software Engineering (ICSE)*. IEEE, 1020–1032.

Ana-Maria Istrate, Donghui Li, Dario Taraborelli, et al.. 2022. "A large dataset of software mentions in the biomedical literature." *arXiv preprint arXiv:2209.00693*.

Slinger Jansen, Elena Baninemeh, Siamak Farshidi, et al.. 2022. "FAIRSECO: An infrastructure for measuring impact of research software." In: *CEUR Worksh. Proc.* Vol. 3245. CEUR WS.

Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. "Classifying developers into core and peripheral: An empirical study on count and network metrics." In: *Int'l Conf. Software Eng. (ICSE)*. IEEE, 164–174.

Arne Johanson and Wilhelm Hasselbring. 2018. "Software engineering for computational science: Past, present, future." *Comput. Sci. Eng.*, 20, 2, 90–109.

Niklas Jørgensen. 2021. "Assessing Open Source Software as a Scholarly Contribution." *Commun. ACM*, 64, 6, 62–67.

Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, et al.. 2016. "An in-depth study of the promises and perils of mining GitHub." *Empir. Softw. Eng.*, 21, 2035–2071.

Upulee Kanewala and James M. Bieman. 2014. "Testing scientific software: A systematic literature review." *Inf. Softw. Technol.*, 56, 10, 1219–1232.

Daniel Katz, Kyle Niemeyer, and Arfon Smith. 2018. "Publish your software: Introducing the Journal of Open Source Software (JOSS)." *Comput. Sci. Eng.*, 20, 3, 84–88.

Aidan Kelley and Daniel Garijo. 2021. "A framework for creating knowledge graphs of scientific software metadata." *Quantitative Science Studies*, 2, 4, 1423–1446.

"The Cox Proportional Hazards Model and Its Characteristics." *Survival Analysis: A Self-Learning Text*. Springer, 97–159.

Julia Koehler Leman, Brian D Weitzner, Pamela D Renfrew, et al.. 2020. "Better together: Elements of successful scientific software development in a distributed collaborative community." *PLOS Comput. Biol.*, 16, 5.

Karim R. Lakhani and Eric von Hippel. 2003. "How Open Source Software Works: "Free" User-to-User Assistance." *Research Policy*, 32, 6, 923–943.

Anna-Lena Lamprecht, Leyla Garcia, Mateusz Kuzak, et al.. 2020. "Towards FAIR principles for research software." *Data Science*, 3, 1, 37–59.

J Richard Landis and Gary G Koch. 1977. "An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement among Multiple Observers." *Biometrics*, 33, 2, 363–374.

Benjamin D Lee. 2018. "Ten simple rules for documenting scientific software." *PLOS Comput. Biol.*, 14, 12, e1006561.

Zhifang Liao, Benhong Zhao, Shengzong Liu, et al.. 2019. "A prediction model of the project life-span in open source software ecosystem." *Mobile Networks and Applications*, 24, 1382–1391.

R. J. A. Little and D. B. Rubin. 1987. *Statistical Analysis with Missing Data*. Willey Series in Probability and Mathematical Statistics. John Willey & Sons.

Yuxing Ma, Tapajit Dey, Chris Bogart, et al.. 2021. "World of code: enabling a research workflow for mining and analyzing the universe of open source VCS data." *Empir. Softw. Eng.*, 26, 1–42.

Konstantinos Manikas and Klaus Marius Hansen. 2013. "Software ecosystems–A systematic literature review." *J. Syst. Softw.*, 86, 5, 1294–1306.

Allen Mao, Daniel Garijo, and Shobeir Fakhraei. 2019. "SoMEF: A framework for capturing scientific software metadata from its documentation." In: *2019 IEEE Int'l Conf. on Big Data (Big Data)*. IEEE, 3032–3037.

Reed Milewicz, Gustavo Pinto, and Paige Rodeghero. 2019. "Characterizing the roles of contributors in open-source scientific software projects." In: *Int'l Conf. MSR (MSR)*. IEEE, 421–432.

Courtney Miller, Mahmoud Jahanshahi, Audris Mockus, Bogdan Vasilescu, and Christian Kästner. 2025. "Understanding the Response to Open-Source Dependency Abandonment in the npm Ecosystem." In: *Int'l Conf. Software Eng. (ICSE)*.

Courtney Miller, Christian Kästner, and Bogdan Vasilescu. 2023. ""We Feel Like We're Winging It:" A Study on Navigating Open-Source Dependency Abandonment." In: *Int'l Conf. Foundations of Software Eng. (FSE)*, 1281–1293.

Audris Mockus, Roy T. Fielding, and James Herbsleb. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla." *ACM Trans. Softw. Eng. Methodol.*, 11, 3, 309–346.

Audris Mockus, Diomidis Spinellis, Zoe Kotti, and Gabriel John Dusing. 2020. "A Complete Set of Related Git Repositories Identified via Community Detection Approaches Based on Shared Commits." In: *Int'l Conf. MSR (MSR)*.

Dirk F Moore et al.. 2016. *Applied survival analysis using R*. Vol. 473. Springer.

Lee Morris. 2021. "Understanding Software Sustainability in the field of Research Software Engineering." Ph.D. Dissertation. University of Huddersfield.

Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. "Curating GitHub for engineered software projects." *Empir. Softw. Eng.*, 22, 3219–3253.

Justin Murphy, Elias T Brady, Shazibul Islam Shamim, and Akond Rahman. 2020. "A curated dataset of security defects in scientific software projects." In: *Proc. of the 7th Symp. on Hot Topics in the Science of Security*, 1–2.

Udit Nangia and Daniel Katz. 2017. "Track 1 Paper: Surveying the U.S. National Postdoctoral Assoc. Regarding Software Use and Training in Research."

Richard R. Nelson. 1993. *National Innovation Systems: A Comparative Analysis*. Oxford University Press, New York, NY. ISBN: 978-0195076172.

Andrew Nesbitt, Boris Veytsman, Daniel Mietchen, et al.. 2024. "Biomedical open source software: Crucial packages and hidden heroes." *arXiv preprint arXiv:2404.06672*.

Christopher Newfield. 2025. "Humanities Decline in Darkness: How Humanities Research Funding Works." *Public Humanities*, 1, e31.

Ileana Ober and Iulian Ober. 2017. "On patterns of multi-domain interaction for scientific software development focused on separation of concerns." *Procedia computer science*, 108, 2298–2302.

Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. "Categorizing the content of GitHub README files." *Empir. Softw. Eng.*, 24, 1296–1327.

F Queiroz, R Silva, J Miller, S Brockhauser, and H Fangohr. 2017. "Good Usability Practices in Scientific Software Development." In: *Worksh. on Sustainable Software for Science (WSSSPE5)*. WSSSPE.

Karthik Ram, Carl Boettiger, Scott Chamberlain, et al.. 2018. "A community of practice around peer review for long-term research software sustainability." *Comput. Sci. Eng.*, 21, 2, 59–65.

Rahul Ramachandran, Kaylin Bugbee, and Kevin Murphy. 2021. "From Open Data to Open Science." *Earth and Space Science*, 8, 5, e2020EA001562.

Klaus Rechert, Jurek Oberhauser, and Rafael Gieschke. 2021. "How long can we build it? ensuring usability of a scientific code base." *Int'l Journal of Digital Curation*, 16, 1, 11–11.

Bernard Reddy and David Evans. May 2002. "Government Preferences for Promoting Open-Source Software: A Solution in Search of a Problem." *Michigan Telecommunications and Technology Law Review*, 9, (May 2002).

J. Paulo Ribeiro, E. Lima, and G. Rossi. 2020. "Governments adoption of open-source software: A systematic review of barriers and enablers." *Government Inf. Quarterly*, 37, 3, 101447.

Patrick Royston and Mahesh KB Parmar. 2013. "Restricted mean survival time: an alternative to the hazard ratio for the design and analysis of randomized trials with a time-to-event outcome." *BMC medical research methodology*, 13, 1–15.

Rebecca Sanders, Diane Kelly, and Terry Shepard. 2008. "The development and use of scientific software."

Samuel D Schwartz, Stephen F Fickas, Boyana Norris, and Anshu Dubey. 2024. "A survey of open source software repositories in the us department of energy's national laboratories." *Comput. Sci. Eng.*

Charles M. Schweik and Robert C. English. 2012. *Internet success: A study of open-source software commons*. MIT Press.

Philip Sedgwick. 2012. "Multiple significance tests: the Bonferroni correction." *BMJ*, 344.

Shubh Sharma and Paul Sturges. 2007. "Making use of open-source software in government: The practicalities of policy." *Inf. Development*, 23, 2, 115–125.

Vanessa Sochat, Nicholas May, Ian Cosden, Carlos Martinez-Ortiz, and Sadie Bartholomew. 2022. "The Research Software Encyclopedia: a community framework to define research software." *J. Open Research Software*.

David AW Soergel. 2014. "Rampant software errors may undermine scientific results." *F1000Research*, 3.

Jurriaan H Spaaks, J Maassen, T Klaver, et al.. 2018. "Research Software Directory." *Zenodo*.

Dustin S. Stoltz and Marshall A. Taylor. Mar. 2024. *Mapping Texts: Computational Text Analysis for the Social Sciences*. Oxford University Press, (Mar. 2024).

Carly Strasser, Kate Hertweck, Josh Greenberg, Dario Taraborelli, and Elizabeth Vu. 2022. "Ten simple rules for funding scientific open source software." *PLOS Comput. Biol.*, 18, 11, e1010627.

Elizabeth A Stuart, Stephen R Cole, Catherine P Bradshaw, and Philip J Leaf. 2011. "The use of propensity scores to assess the generalizability of results from randomized trials." *J. R. Stat. Soc.*

Jiayi Sun, Aarya Patil, Youhai Li, Jin LC Guo, and Shurui Zhou. 2024. "How to Sustain a Scientific Open-Source Software Ecosystem: Learning from the Astropy Project." *arXiv preprint arXiv:2402.15081*.

Elizabeth Tipton. 2013. "Improving generalizations from experiments using propensity score subclassification: Assumptions, properties, and contexts." *Journal of Educational and Behavioral Statistics*, 38, 3, 239–266.

Erik H Trainer, Chalalai Chaihirunkarn, Arun Kalyanasundaram, and James D Herbsleb. 2014. "Community code engagements: summer of code & hackathons for community building in scientific software." In: *Int'l Conf. Supporting Group Work (GROUP)*, 111–121.

Joanna Tucker. 2022. "Facing the Challenge of Digital Sustainability as Humanities Researchers." *Journal of the British Academy*, 10, 93–120.

Hajime Uno, Brian Claggett, Lu Tian, et al.. 2014. "Moving beyond the hazard ratio in quantifying the between-group difference in survival analysis." *Journal of clinical Oncology*, 32, 22, 2380–2385.

Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. "Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem." In: *Int'l Conf. Foundations of Software Eng. (FSE)*, 644–655.

Supatsara Wattanakriengkrai, Bodin Chinthanet, Hideaki Hata, et al.. 2022. "GitHub repositories with links to academic papers: Public access, traceability, and evolution." *J. Syst. Softw.*

Michael Weiss. 2005. "The Promise of Government Incentives for Open-Source Projects." *IEEE Software*, 22, 5, 74–77.

James Willenbring and Gursimran Singh Walia. 2021. *Evaluating the Sustainability of Computational Science and Engineering Software: Empirical Observations*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Bruno Lopes Xavier, Rodrigo Pereira dos Santos, and Davi Viana dos Santos. 2020. "Software Ecosystems and Digital Games: Understanding the Financial Sustainability Aspect." In: *ICEIS (2)*, 450–457.

Minghui Zhou and Audris Mockus. 2011. "Does the initial environment impact the future of developers?" In: *Int'l Conf. Software Eng. (ICSE)*, 271–280.

Minghui Zhou and Audris Mockus. 2012. "What make long term contributors: Willingness and opportunity in OSS community." In: *Int'l Conf. Software Eng. (ICSE)*. IEEE, 518–528.

Minghui Zhou, Audris Mockus, Xiujuan Ma, Lu Zhang, and Hong Mei. 2016. "Inflow and retention in oss communities with commercial involvement: A case study of three hybrid projects." *ACM Trans. Softw. Eng. Methodol.*, 25, 2, 1–29.

Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2019. "What the fork: a study of inefficient and efficient forking practices in social coding." In: *Int'l Conf. Foundations of Software Eng. (FSE)*, 350–361.

Received 2025-02-26; accepted 2025-04-01