Research software as scientific evidence: clarifying missing specifications

Developing research software is a technical task – how it is appropriately disclosed is a scientific matter.

Research software is pivotal in scientific research, generating, processing, and analyzing results intended for publications (*1*). Its increasing integration into academic activities is a testament to its significance across the scientific community, prompting a greater focus on its development practices. Many efforts have emerged to strengthen transparency, accessibility, and reusability in research software by establishing guidelines, recommending good practices, developing verification platforms, and offering educational resources (*2*–*5*). Yet these efforts tend to emphasize the *software* aspect, without clarifying what additional role is implied by the *research* aspect.

We refer to this additional role as the evidentiary role – the idea that research software should serve as part of the evidence that supports scientific claims, beyond its technical function. This evidentiary role shifts the focus away from how to develop research software toward the question of how it should be disclosed and under what specifications, specifically what constitutes evidentiary sufficiency in the context of a given scientific claim. However, no shared specifications currently exist within the stakeholder community (including authors, editors, and reviewers) on how research software should be disclosed to support the evidentiary role.

This lack of consensus has already had real-world consequences. One notable example is Nature's decision to publish the AlphaFold3 paper without its code (*6*). Specifically, *Nature Portfolio Editorial Policies* require that code supporting central claims be made available to reviewers upon request. Despite the stated policies, *Science* reported that one reviewer had only temporary access to an early web server and described repeated, unanswered requests for code. *Nature* later cited biosecurity risks and the inclusion of pseudocode to justify the editorial decision, while *DeepMind* attributed the restriction to commercial considerations. This example illustrates how each stakeholder operates from a distinct position and how the absence of shared expectations can lead to tensions in disclosure decisions.

Beyond tensions, the more serious concern is the gray zone created by this lack of consensus. In this zone, disclosure practices are inconsistently interpreted and selectively applied. Researchers acting in good faith may believe that, as long as they have subjectively avoided data fabrication and result concealment, there is no need to consider what constitutes sufficient disclosure. Meanwhile, mounting evidence shows how others have exploited this lack of consensus to justify withholding critical software components or data elements. In both situations, evidentiary insufficiency can persist without shared disclosure specifications. This goes beyond what case-by-case discretion can fully resolve.

Why, then, is there still no clear consensus on disclosure specifications? One reason lies in the wide variability of disclosure norms across scientific fields (*7*). In some fields, community standards guide software sharing and verification, while disclosure remains informal or absent in others. A second complicating factor is the influence of external disclosure barriers such as proprietary licensing, commercial interests, and security concerns, all of which introduce competing priorities that shape what can or should be disclosed. Together, these factors constrain the broader establishment of shared disclosure specifications, making it increasingly difficult to keep pace with the growing interdisciplinarity and complexity of contemporary research practice.

Herein, we report the outcome of preliminary discussions involving representatives from the stakeholder community that helped initiate the preliminary development of shared disclosure specifications. Building on insights from these discussions, we propose a new conceptual model called Evidence-Oriented Programming. Rather than focusing on how research software is technically implemented, this model reorients software development and evaluation around the relationship between published scientific claims and computational artifacts (i.e., software components and data elements) that substantiate them. To operationalize this model, we introduce the Evidence Chain Method – a strategy for identifying and linking the evidentiary computational artifacts. We discuss its applicability across different fields and propose how it may help resolve trust gaps when external disclosure barriers arise. Overall, these efforts seek to support a shift from case-by-case disclosure decisions toward evidence-oriented specifications across the scientific ecosystem.

From Replicability to Evidentiary Sufficiency

Replication has long been central to how scientists establish confidence in the scientific merit of results, supported by a shared body of community positions and guidance frameworks, such as the FAIR4RS Principles (*2*), the TOP Guidelines (*8*), the OpenAIRE Guidelines, and the COAR Framework (*9*). Therefore, it is worth asking why evidentiary sufficiency, rather than replicability, is our primary concern.

- Replicability is not universally attainable. It is substantively undermined by external disclosure barriers and reliance on proprietary software or hardware environments, which are common realities in contemporary research ecosystems.
- The mere success or failure of replication is an inadequate indicator of the health of science (*10*). A successful replication may simply reproduce unrecognized systematic errors (*11*). Conversely, a failed replication does not necessarily refute the original claims, as such outcomes might arise from undocumented parameters, uncontrolled variables, or even stochastic effects.
- Providing research software does not guarantee reproducibility. A recent reproducibility analysis of over 2,000 R code repositories found that only 26% of scripts could be successfully executed on the first attempt, and even after code cleaning, the success rate reached only 44% (*12*).

These realities highlight the need to assess scientific validity in the context of the full evidentiary structure, rather than focusing on isolated acts of replication, and they motivate our exploration of how research software can help construct and support such a context.

Evidence-Oriented Programming

While evidentiary sufficiency can, in principle, be assessed at any stage of research, the peer review stage represents the most critical juncture. Only during peer review, when publication decisions are still pending, does sufficient leverage exist to address potential evidentiary insufficiency. Once publication has occurred, it becomes significantly more challenging to request additional disclosure, and any errors that surface may be considerably harder to correct in a timely manner, potentially allowing flawed results to propagate more widely before correction can occur (*11*).

For the specific context of the peer review stage, we have developed a conceptual model, called *Evidence-Oriented Programming* (EOP), to examine the tensions between stakeholder priorities and the possibilities for negotiated consensus. Any attempt to assess evidentiary sufficiency during peer review must begin with stakeholder consensus on the scope, timing, and form of disclosure.

- **Scope:** Full openness remains a desirable ideal, but it is not always feasible or appropriate, as different stakeholder groups – authors, editors, reviewers, and external actors (such as regulatory agencies, funding bodies, and other entities that exert oversight or guidance over disclosure practices) – bring distinct and competing priorities. As illustrated in Figure 1A, the final disclosure state of research software often reflects the interplay of these priorities rather than adherence to a single normative standard. EOP does not seek to persuade all stakeholders to adopt full openness. Instead, it accepts the reality of competing priorities and aims to balance legitimate constraints with the evidentiary integrity needed to support scientific claims and reduce the risk of serious downstream consequences (Figure 1B). While the minimum bounds of acceptable disclosure remain unclear, it should exclude practices collectively regarded as unacceptable by the stakeholder community. Potentially, retraction cases can serve as concrete reference points for identifying unacceptable disclosure practices.

- **Timing:** In most cases, authors can make research software fully accessible at the time of submission, and we encourage them to do so. However, potential conflicts of interest between authors and reviewers must not be overlooked, particularly when reviewers might improperly use the research software to publish overlapping work ahead of the authors (*13*). In such cases, EOP needs to support authors in providing a mandatory disclosure schedule, explicitly indicating which components remain withheld, including associated hash values to enable verification when these components are eventually disclosed, along with a justification of the disclosure risks and their relevance to the scientific claims under review.

- **Form:** Evaluating research software poses challenges distinct from general-purpose software review (*14*). Firstly, research software often includes novel data types, modeling assumptions, or uncommon dependencies. While they also occur in general-purpose software, they become especially consequential in research settings, where their coupling to the evidentiary evaluation of scientific claims can render evaluation during peer review difficult or even infeasible. In addition, external disclosure barriers may prevent full source code availability, with critical components accessible only through compiled binaries or restricted endpoints, sometimes under time-limited conditions. Furthermore, reviewers may lack the domain expertise required to evaluate complex or poorly documented research software. Finally, single- and double-blind review formats restrict author-reviewer interaction, reducing opportunities to resolve ambiguities or negotiate appropriate disclosure standards. Accordingly, EOP must encompass a structured approach to representing research software, helping stakeholders align on how it should be disclosed, interpreted, and evaluated.

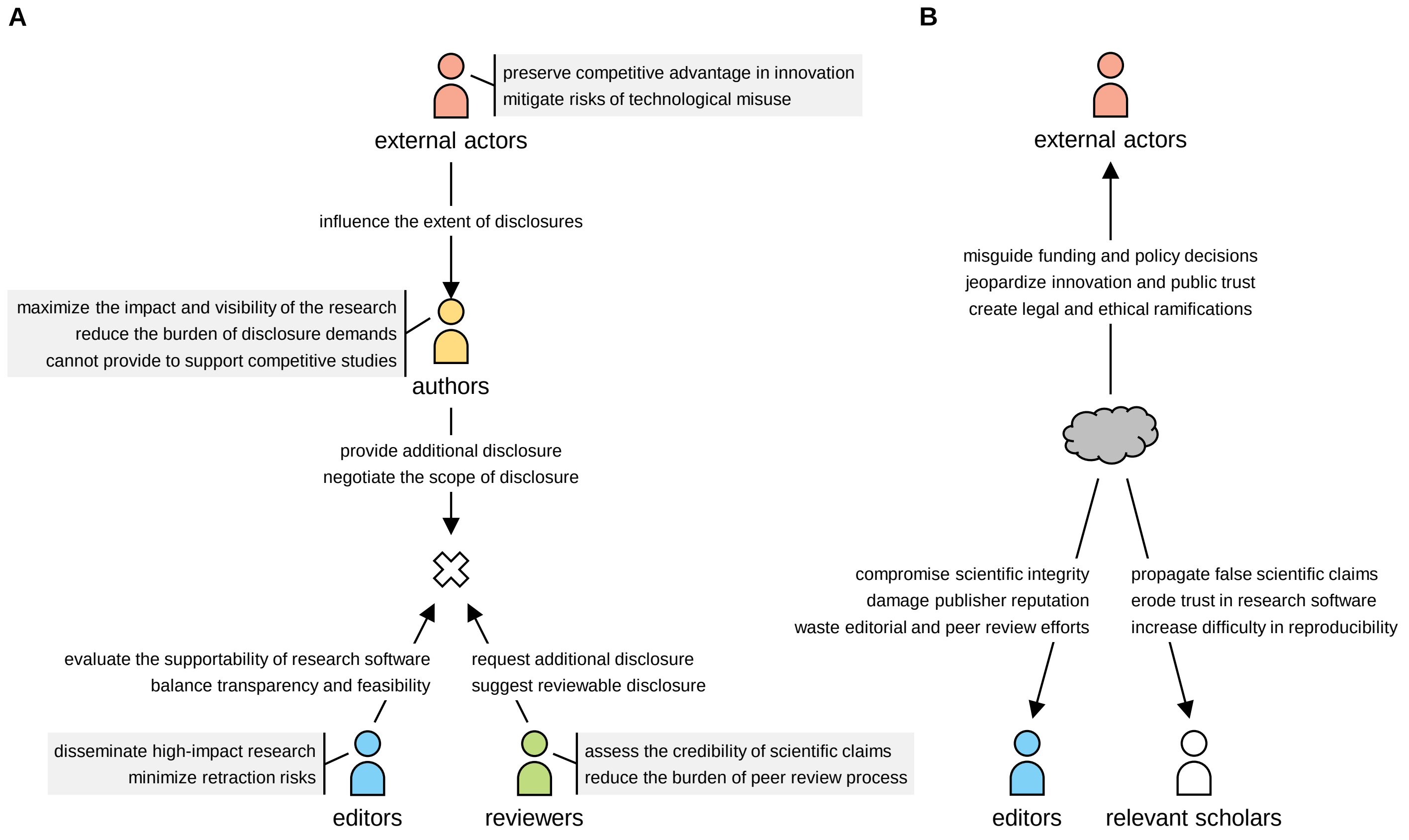
A
preserve competitive advantage in innovation
mitigate risks of technological misuse
external actors
influence the extent of disclosures
maximize the impact and visibility of the research
reduce the burden of disclosure demands
cannot provide to support competitive studies
authors
provide additional disclosure
negotiate the scope of disclosure
evaluate the supportability of research software
balance transparency and feasibility
request additional disclosure
suggest reviewable disclosure
disseminate high-impact research
minimize retraction risks
editors
assess the credibility of scientific claims
reduce the burden of peer review process
reviewers
B
external actors
misguide funding and policy decisions
jeopardize innovation and public trust
create legal and ethical ramifications
compromise scientific integrity
damage publisher reputation
waste editorial and peer review efforts
propagate false scientific claims
erode trust in research software
increase difficulty in reproducibility
editors
relevant scholars

Figure 1 | Interactions within the stakeholder community. **(A)** The priorities of four stakeholder groups in the community. These priorities mediate the behaviors of the corresponding stakeholders, ultimately forming tensions that shape the scope and integrity of research software disclosure. **(B)** Some potential downstream implications from this structural ambiguity. Inconsistencies in disclosure expectations can create uncertainty in assessing scientific claims and may lead to coordination challenges across the broader research ecosystem.

Evidence Chain Method

The Evidence Chain Method (ECM), which we developed by adapting ideas from scientific workflows (*15*), offers a concrete way to put EOP into practice. One of the core difficulties it addresses is the invisibility of evidentiary relationships: individual software modules or datasets often lack the necessary context to clarify their role in supporting a scientific claim. ECM responds to this challenge by mapping computational artifacts to the claim they support, making evidentiary structures explicit, interpretable, and available for review.

ECM introduces a bounded structure that focuses the assessment of evidentiary sufficiency between two key points: the starting and ending artifacts. The starting artifacts are the *input data* explicitly identified in the publication, such as publicly available datasets, experimentally collected measurements, or algorithmically generated values. The ending artifacts are the resulting *visual claims*: figures, tables, or reported statistical values such as correlation coefficients or confidence levels that readers interpret as evidence in the publication. This range makes clear that ECM is designed to ensure that, once input data are declared and used in software-mediated processes, the resulting claims can be traced and evaluated for evidentiary sufficiency. The compliance of non-software-mediated procedures and the integrity of their data collection processes fall beyond the scope of ECM.

To link the starting and ending artifacts, ECM provides an abstracted evidence chain (Figure 2). First, *input data* are transformed into output data through the *experimental or analytical process(es)*. These *output data* can include processed datasets, trained model archives, or other intermediate data elements. A subset of the output data constitutes the *visual data*, which are then converted via the *plotting or summarizing process(es)* to produce the visual claims. Equally important is the *documentation* maintained across these artifacts, which explains both the overarching structure and specific linkages, ensuring that the trace remains coherent and interpretable.

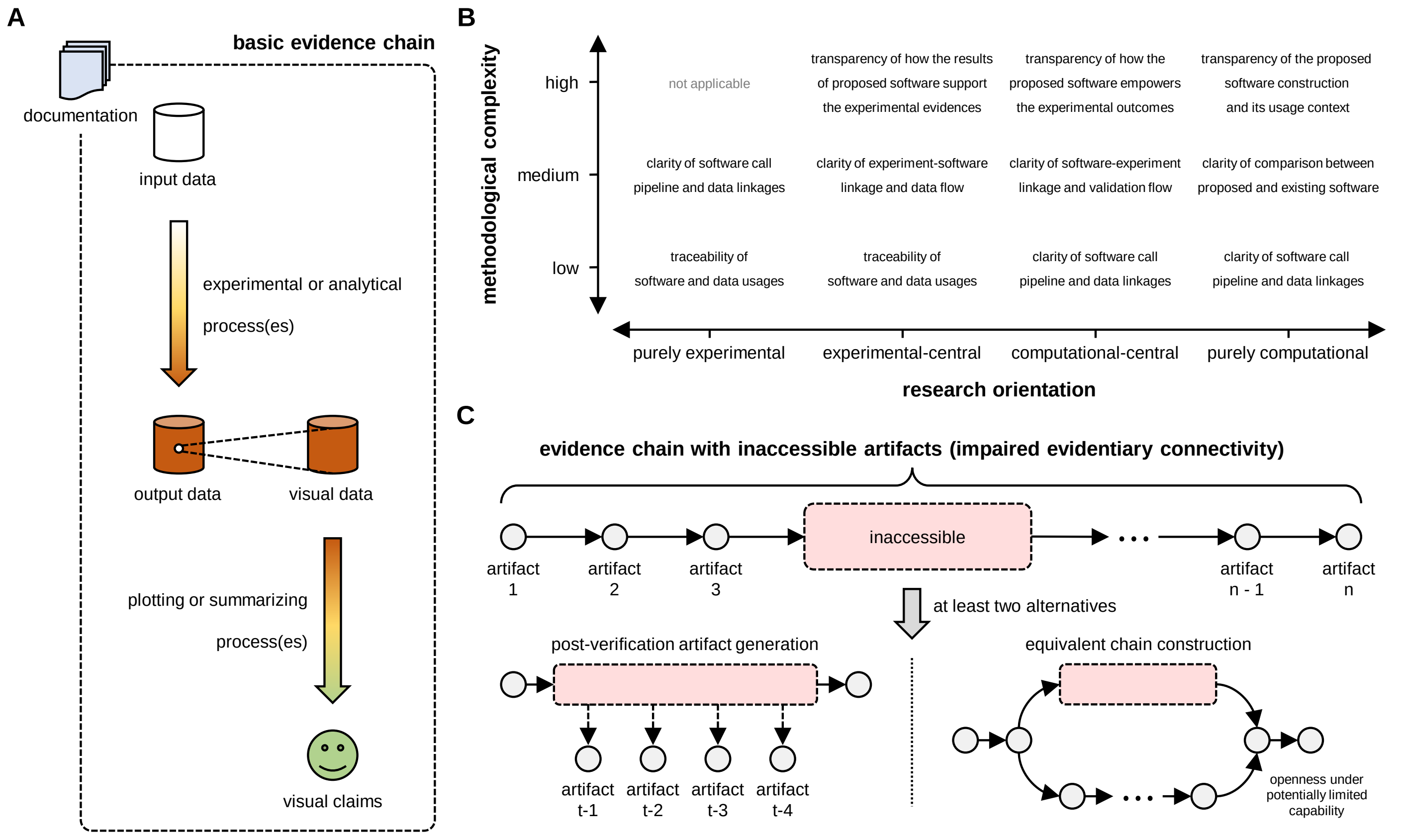
A
basic evidence chain
documentation
input data
experimental or analytical process(es)
output data
visual data
plotting or summarizing process(es)
visual claims
B
methodological complexity
high
medium
low
not applicable
transparency of how the results of proposed software support the experimental evidences
transparency of how the proposed software empowers the experimental outcomes
transparency of the proposed software construction and its usage context
clarity of software call pipeline and data linkages
clarity of experiment-software linkage and data flow
clarity of software-experiment linkage and validation flow
clarity of comparison between proposed and existing software
traceability of software and data usages
traceability of software and data usages
clarity of software call pipeline and data linkages
clarity of software call pipeline and data linkages
purely experimental
experimental-central
computational-central
purely computational
research orientation
C
evidence chain with inaccessible artifacts (impaired evidentiary connectivity)
artifact 1
artifact 2
artifact 3
inaccessible
artifact n - 1
artifact n
at least two alternatives
post-verification artifact generation
artifact t-1
artifact t-2
artifact t-3
artifact t-4
equivalent chain construction
openness under potentially limited capability

Figure 2 | Evidence chain method. **(A)** Structure of the basic evidence chain in this method. This artifact structure enables observation and evaluation of evidentiary sufficiency within the software-mediated procedural range, as detailed in the Main Text. **(B)** The software disclosure objectives of the evidence chain are jointly constrained by research orientation and methodological complexity. Research orientation reflects a continuum from purely experimental to purely computational studies. In purely experimental studies, software is pre-existing and used only for analyzing data obtained from laboratory procedures. Experimental-central studies rarely introduce new software; they often involve independent analyses (such as molecular dynamics simulations) to further support experimental results, or apply existing tools to newly generated datasets to extract specific findings. Computational-central studies, by contrast, are primarily software-driven and advance computational claims, while including limited experimental outcomes that provide empirical support for those claims. Purely computational studies do not generate new experimental data and focus entirely on algorithmic, modeling, or theoretical contributions. In practice, some studies may contain both experimental and computational claims, and their boundaries along this continuum are therefore not always sharply defined. Meanwhile, methodological complexity depends on the interaction or comparative relationships among software components, whether new software is proposed, and whether the new software requires dedicated construction or can only operate within a specific contextual setting. Low complexity studies do not involve the construction of new software; they typically rely on one or several previously reported tools, depending on the research orientation. Median complexity differs across orientations: computational-central and purely computational studies in this category often introduce new methods or extensions of existing ones. High complexity studies frequently involve AI-based research, where the extent to which such systems can substantiate scientific claims depends critically on how they are constructed and used in practice. Field-specific discussions on software disclosure practices are provided in the Supplementary Information. **(C)** The connectivity of the evidence chain can be affected by external disclosure barriers, which may interfere with evidentiary sufficiency. To mitigate direct conflicts between evidentiary sufficiency and such barriers, we offer two alternative strategies. Post-verification artifact generation provides unidirectional statistical outputs that enable verification of the inaccessible chain through supplementary information without exposing sensitive components. Equivalent chain construction establishes a substitute evidence chain that maintains functional equivalence while differing in performance (e.g., computational efficiency or requirements) or scope (e.g., exclusion of biosecurity-sensitive inputs in certain biological contexts). Because these strategies introduce additional workload, they inherently involve trade-offs with evidentiary sufficiency. Specific trade-off discussions are provided in the Supplementary Information.

While these seven artifacts form a basic evidence chain, different academic fields can introduce additional complexity, especially within the *experimental or analytical process(es)*. Depending on whether the scientific claims are algorithm-centered, this stage may range from simple software invocations to sophisticated pipelines that incorporate the authors' proposed algorithms alongside baseline or comparator algorithms. Further, additional data elements such as hyperparameter configurations, random seeds, and initialization routines also become critical to evidentiary interpretation. These differences are not merely technical; they reflect distinct disclosure expectations shaped by variations in research orientation and methodological complexity (Figure 2B). ECM can accommodate such variation through field-specific extensions, enabling scientific claims to remain evidentially sound even as academic fields diverge. Representative differences and their accommodation within ECM are further elaborated in the Supplementary Information.

In practice, full disclosure of evidentiary software components and data elements may not be feasible due to ethical, legal, security, or intellectual property constraints. ECM allows trade-off presentations (Figure 2C), such as providing partial outputs, operational logs, intermediate output data with evidential significance, and/or typical case presentations that preserve evidentiary traceability without unrestricted access to all underlying components. These mechanisms help reviewers and editors assess scientific claims when direct disclosure is constrained. Representative strategies and their evidentiary rationale are discussed in Supplementary Information.

Overall, the basic evidence chain, field-specific variants, and trade-off presentations in ECM collectively instantiate the *scope* and *form* dimensions of EOP. To extend ECM toward the *timing* dimension, one can simulate selective non-disclosure by omitting or withholding a given component to evaluate its precise impact on evidentiary sufficiency. This enables the stakeholder community to assess whether, and how, delayed or withheld disclosure compromises the ability to substantiate scientific claims.

Bridging Implementation and Evaluation

Although ECM offers a concrete operationalization of EOP, it presents many practical challenges. On the implementation side, ECM places substantial demands on authors. It requires proficiency in software design and documentation practices, as well as the ability to present them in a form that supports evaluation. These are skills that many researchers, especially those without formal training in software engineering, may lack. On the evaluation side, ECM remains an early-stage method. Its field-specific variants have yet to be systematically developed, and no large-scale quantitative studies have been conducted to assess its applicability or evidentiary effectiveness across academic fields.

To reduce the practical barriers to the implementation of ECM, we are developing educational tools that reflect how researchers actually engage with software in their academic fields. In addition to static tutorials, these resources include layered guides and workshop-style modules built around case studies, each designed to foreground the evidentiary chains most relevant to a given field. We are also testing ways to embed ECM into existing scientific workflows and exploring whether AI agents might help researchers identify and annotate relevant artifacts during software development.

In parallel, we are working with software quality assurance communities and academic publishers to build the structural conditions necessary for ECM to take root. This involves developing editorial guidelines for reviewers and editors, and partnering with platforms to prototype features that support evidence chain visibility. These efforts aim not just to make ECM technically feasible, but to embed evidentiary review into the publishing process in ways that are practical and scalable.

We have provided several examples of EOP-compliant software implementations in the Supplementary Information and observed that such practices often introduce additional data-storage challenges. To further support the development of EOP-compliant software, we are also addressing the additional data-management demands it generates. In particular, the intermediate artifacts and evidentiary traces required by ECM can be hosted on established research data repositories such as Figshare or institutional equivalents, ensuring their long-term accessibility and reproducibility.

Conditional Full Openness

While EOP aims to address routine disclosure practices, we recognize that exceptional situations might warrant more decisive measures. We suggest that, prior to manuscript acceptance, authors predefine which designated audiences (e.g., regulatory bodies, ethics boards, or expert oversight panels) would be granted full access to their research software under specified conditions – namely, when such work attracts widespread attention and becomes the subject of significant technical doubts after publication. Although this proposal extends beyond the technical scope of EOP itself, it touches on related concerns about evidentiary accountability and might warrant consideration in the development of disclosure policies.

Outlook

This work reflects a concerted effort by members of the stakeholder community to address the evidentiary ambiguity that persists across research software practices. It seeks to clarify the gray zone created by the absence of consensus on disclosure, where disclosure practices are inconsistently interpreted and selectively applied, and to offer principles for mitigating its effects. The introduction of EOP and ECM is not intended to replace earlier replicability-focused efforts, but to complement them by acknowledging the real-world constraints and the need for claim-centered, disclosure-aware evaluation.

We hope this work will serve as a catalyst for broader engagement across the stakeholder community in developing shared expectations around evidentiary sufficiency. Supported by incentives for structured software disclosures, such expectations can foster a research ecosystem in which software is not merely a computational tool but a verifiable contributor to scientific knowledge. Achieving this vision will require not only technical and institutional readiness, but also a shared commitment to treating research software as a site of evidentiary accountability – and acting on that commitment. Hence, we call on publishers, funders, and oversight bodies to pilot evidentiary specifications for research software disclosure in the upcoming review cycle.

References and Notes

1. M. Gruenpeter et al. *Defining research software: a controversial discussion. Zenodo* (2021).
2. M. Barker et al. Introducing the FAIR principles for research software. *Scientific Data* **9** (2022), 622.
3. H. Hunter-Zinck et al. Ten simple rules on writing clean and reliable open-source scientific software. *PLoS Computational Biology* **17** (2021), e1009481.
4. T. Staubitz et al. CodeOcean – A versatile platform for practical programming excercises in online environments. *IEEE Global Engineering Education Conference*. (2016), 314–323.
5. G. Wilson. Software carpentry: getting scientists to write better code by making them more productive. *Computing in Science & Engineering* **8** (2006), 66–69.
6. S. Wankowicz et al. *AlphaFold3 Transparency and Reproducibility. Zenodo* (2024).
7. H. Zhang et al. Reviewability and supportability: New complementary principles to empower research software practices. *Computational and Structural Biotechnology Journal* **23** (2024), 3989–3998.
8. B. A. Nosek et al. Promoting an open research culture. *Science* **348** (2015), 1422–1425.
9. Confederation of Open Access Repositories. *COAR Community Framework for Good Practices in Repositories, Version 2. Zenodo* (2022).
10. N. A. of Sciences et al. *Reproducibility and replicability in science*. National Academies Press, 2019.
11. G. Miller. A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science* **314** (2006), 1856–1857.
12. A. Trisovic, M. K. Lau, T. Pasquier, and M. Crosas. A large-scale study on research code quality and execution. *Scientific Data* **9** (2022), 60.
13. J. M. DuBois. *RCR Casebook: Stories about Researchers Worth Discussing*. U.S. Office of Research Integrity. (Visited on 06/16/2025).
14. N. B. Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes* **35** (2010), 8–13.
15. C. S. Liew et al. Scientific workflows: moving across paradigms. *ACM Computing Surveys* **49** (2016), 1–39.