



UNIVERSITÀ DI PISA

**Corso di Laurea in Informatica
Umanistica**

**The Role of Software Journals in the Advancement
of Open Science**

Candidato

Alessia Madonna

Relatori

Prof. Vittore Casarosa

Prof.ssa Enrica Salvatori

Dott. Leonardo Candela

Anno Accademico 2024-2025

*“Quelli come te, che hanno due sangui diversi nelle vene,
non trovano mai riposo né contentezza;
e mentre sono là, vorrebbero trovarsi qua,
e appena tornati qua, subito hanno voglia di scappar via.
Tu te ne andrai da un luogo all’altro,
come se fuggissi di prigione, o corressi in cerca di qualcuno;
ma in realtà inseguirai soltanto le sorti diverse
che si mischiano nel tuo sangue,
perché il tuo sangue è come un animale doppio,
è come un cavallo grifone, come una sirena.”*

Elsa Morante, L'isola di Arturo

Table of content

Introduction.....	3
Chapter I “Defining the Roles of Research Software in Scientific Research”	5
I.I The roles of research software: uses and definitions.....	5
I.II Research software or software used in research? A fundamental distinction.....	7
I.III Recognizing Research Software as a Scientific Product.....	8
I.IV Barriers to Code Sharing in Scientific Research.....	10
I.V FAIR Principles Applied to Research Software: Towards Greater Transparency and Reusability.....	14
I.V.I Key Issues in the Application of FAIR Principles for Software.....	15
I.VI From FAIR Principles to the Open Science Movement.....	15
I.VI.I The Phenomenon of Open Science in Europe.....	16
I.VI.II Strategy and Initiatives for the Implementation of the FAIR Principles and Open Science in Italy.....	17
Chapter II Software Journal: History and Theory of a New Paradigm in Scholarly Communication.....	19
II.I "As We May Think" and the Precursor Vision of Software in Scientific Research	20
II.II The Origins of Software Journals: The First Pure and Hybrid Publications.....	22
II.II.I Computer Physics Communications and CPC Program Library.....	22
II.II.II ACM Transactions on Mathematical Software (TOMS) and CALGO.....	23
II.III The GNU Project.....	25
II.IV The Rise of Electronic Journals and the Early Practice of Software Code Sharing in the 1990s.....	26
II.IV.I Journal of Statistical Software.....	27
II.IV.II Open Source Initiative.....	27
II.V From the 00s to the present day.....	28
II.V.I The Advent and Diffusion of Software Repositories.....	30
II.V.II The Proliferation of Software Journals in the 21st Century: Case Studies and Editorial Goals.....	30
SoftwareX.....	31
Journal of Open Source Software (Joss).....	32
The Journal of Open Research Software (JORS).....	32
Chapter III Software Papers in the Academic Publishing Landscape: Comparative Analysis of Pure and Hybrid Journals.....	34
III.I Research Questions for Analyzing Software Journals and Software Papers.....	35
III.II Methodology for Identifying and Classifying Software Journals.....	35
III.III Software Journals and Software Papers Proliferation and Uptake.....	45
III.III.I The birth of Software Journals: a quantitative analysis.....	45
III.III.II The development of software paper: a quantitative analysis.....	46
III.IV Disciplinary Distribution of Software Journals.....	51

III.V Software Hosting Approaches.....	55
III.VI Software Documentation Practices.....	60
III.VI.II Software Paper Template.....	63
III.VI.III The Length of Software Papers.....	64
III.VII Article Processing Charges (APC) and Access Models in Software Journals..	66
.....	66
III.VIII Software Paper and Software License.....	68
Chapter IV Conclusion, Prospects and Future Directions.....	72
References.....	78

Introduction

In the current landscape of scientific research, computational tools have taken on a dominant role, surpassing their initial purpose as mere analytical aids to become essential elements of the scientific investigation process. Within this framework, research software is recognised as a crucial component of the contemporary methodological infrastructure, fostering innovation and enabling the dissemination of research results by the principles of Open Science. The shift toward increased openness and transparency in the sharing of scientific software far exceeds any question of mere technical detail; it reflects profound cultural change among academic communities, which holds profound implications for issues of research reproducibility and scientific innovation but also for dynamics of intra- and interdisciplinary collaboration.

In response to this evolution, new categories of scientific publication have begun to emerge over recent years—ones specifically dedicated to documenting and sharing research software. The publications are characterised by their particular emphasis on the description and dissemination of novel software, which is usually accompanied by the source code itself, typically hosted in specialised repositories. This particular type of publication, which for the purpose of this discussion will be named *software paper*, has not only enabled the establishment of specialised journals but also motivated many traditional scientific journals to include special sections for this type of contribution. This development required the establishment of new editorial practices to manage this particular genre of scientific publication, thus bringing with it new challenges and opportunities in the field of scholarly communication.

The objective of this study is to examine this phenomenon, starting with a broad examination of the meaning of research software in modern scholarly research and followed by the analysis of *software journals* and *software papers*. The study—carried out in the framework of an academic internship—relied on the systematic documentary analysis of resources available on the internet, mainly focusing on official sites of scholarly journals that publish *software papers*. The study involved an in-depth review of several aspects, from diverse terminologies used for this type of publication to policies related to repository management, from licensing options—both of the articles and of the software—to handling updates of the software and Article Processing Charges (APCs) practices. This methodological framework allowed for the identification of common trends, the establishment of best practices, and areas that are expected to improve in the future within the field of scientific software publishing.

The aim of this study is to present an overview of the current situation in the domain of software literature, identify its critical aspects, and highlight interesting elements for future research on this particular type of scholarly publication.

This thesis, in addition to this introduction, is structured into four chapters, allowing for a comprehensive exploration of the subject matter.

Chapter I, 'Defining the Roles of Research Software in Scientific Research'

It discusses the phenomenon of research software: how it has become an integral element in modern scientific research. It points at some of the challenges related to the management, documentation, and preservation of software and its role within *FAIR* and *Open Science* principles. It also underlines the efforts being made internationally and at the European level in the promotion of sharing and reuse of published software.

Chapter II 'Software journal: history and theory of a new paradigm in scholarly communication'

It investigates the phenomenon of *software journals* from the theoretical and historical point of view, tracing the development of such forms of scientific publication and analysing how the need for journals entirely devoted to publishing *software papers* has developed over time.

Chapter III 'Software papers in the academic publishing landscape: comparative analysis of pure and hybrid journals'.

Presents the results of research conducted during the university internship, offering a detailed analysis of academic journals that publish *software papers*, both in the case of pure journals (only *software papers* are accepted for publication) and hybrid journals (*software papers* are published together with “traditional” papers). The analysis covers aspects such as nomenclature used for this type of article, repositories employed, licensing policies, how updates are managed, article processing charge (APC), article templates, and the quantitative evolution of publications over time.

Chapter IV 'Conclusion, Prospects and Future Directions'

Provides a comprehensive summary of the key findings from previous chapters, offering a critical reflection on the current landscape of software publication. It discusses the challenges and opportunities identified throughout the study and outlines potential directions for future research in the field.

Chapter I “Defining the Roles of Research Software in Scientific Research”

In an ever-changing scientific landscape, research software has emerged as a focal point, transforming from a mere supporting component to an essential constituent of scientific research. This chapter aims to delineate and examine the various facets of research software, discussing its definitions, uses, and challenges to the scholarly community.

The chapter starts with a review of the various functions of research software, outlining its key roles and most frequent definitions in the current scientific landscape (cf. [Sec. I.I](#)). Next, an essential difference between "research software" and "software used in research" is presented, describing the conceptual and practical differences that influence their application and scholarly appreciation (cf. [Sec. I.II](#)). Attention is brought to acknowledging research software as an acceptable scientific product in every regard while presenting the problems and possibilities concerning its assessment and citation in academic contributions (cf. [Sec. I.III](#)).

The chapter then presents a detailed examination of the obstacles to code sharing in scientific research, an issue of paramount importance for result transparency and reproducibility. (cf. [Sec. I.IV](#)). Lastly, FAIR principles applicable to research software are also discussed, and approaches for enhancing the transparency and reusability of scientific software are evaluated (cf. [Sec. I.V](#)). Research software is discussed in the context of Open Science, and the leading European and Italian moves for favouring its adoption are examined, highlighting how such policies are helping in managing research software, data sharing and open access to scientific results within an integrated framework of transparency and reusability that adds value to the effectiveness of the FAIR principles.

I.I The roles of research software: uses and definitions

The use of software in the field of scientific research has become an integral part of it, often thought to be a requirement. The application of advanced computational tools allows for the automation and improvement of many tasks and processes, such as data collection and analysis, which otherwise, without these technologies, would demand considerably longer periods of time and considerable resources. In a nutshell, increased data complexity and volume with which researchers are working, in addition to the requirements of speed and accuracy, have made software an indispensable tool for overcoming contemporary challenges in scientific research. Moreover, the automation of processes will not only enhance efficiency in the use of time and resources but will

also reduce the chances of human error and thus increase the quality and reproducibility of the results obtained.

Over time, various definitions for 'research software' have emerged, each with somewhat different emphases on the concept. Some of the more prominent ones include:

“Research Software is a well identified set of code that has been written by a (..) research team. It is software that has been built and used to produce a result published or disseminated in some article or scientific contribution. Each research software encloses a set (of files) that contains the source code and the compiled code. It can also include other elements such as the documentation, specifications, use cases, a test suite, examples of input data and corresponding output data, and even preparatory material.” (Gomez-Diaz and Recio, 2022)

“Software that is used to generate, process or analyse results that you intend to appear in a publication (either in a journal, conference paper, monograph, book or thesis). Research software can be anything from a few lines of code written by yourself, to a professionally developed software package.” (Hettrick et al., 2014)

“A computer-based application that converts inputs into outputs to support the user in one or more research tasks.” (Morais and Brailsford, 2019)

The above definitions are further enriched by broader insights that have emerged in academic discussions, such as those from the international workshop *The Future of Research Software*, organised in November 2022 by the Research Software Alliance (ReSA) in collaboration with the Netherlands eScience Center (GRUENPETER ET AL., 2021). On this occasion, the multifaceted roles that software plays in research were documented and categorised along different operational axes:

- *Research Software is a component of our instruments.*
- *Research Software is the instrument.*
- *Research Software analyses research data.*
- *Research Software presents research results.*
- *Research Software integrates existing components to a working whole.*
- *Research Software is infrastructure or an underlying tool.*
- *Research Software facilitates distinctively research oriented collaborations.*

These definitions, although differing in scope and focus, have some basic elements in common. First, they all acknowledge the function of software as a critical tool for data production and analysis in science. Second, there is common agreement on the significance of the relationship between software and scientific outcomes, stressing the connection between code and knowledge creation. However, significant differences also

emerge: some definitions emphasise the nature of software as a set of files and documentation (Gomez-Diaz and Recio, 2022), while others focus on its function in the research process (HETTRICK ET AL., 2014) or on its role as a research tool (Morais and Brailsford, 2019). Van Nieuwpoort's approach, instead, further broadens the perspective, describing software not only as an analytical tool but also as an infrastructure and catalyst for scientific collaboration.

In summary, the concept of research software is located at the intersection of code, methodology and scientific infrastructure. While software can be seen as a mere technical tool, it also plays a broader role in the construction of scientific knowledge, influencing not only how results are obtained but also how they are shared, interpreted, and reproduced.

I.II Research software or software used in research? A fundamental distinction

It is important to clearly distinguish between the distinct roles played by research software and software used in research for several reasons. A precise characterisation of these concepts not only furthers understanding of the various classes of software involved in the research process but also helps to better define responsibilities, management policies, and practices conducive to transparency, reproducibility, and sustainability of scientific results. Without such a distinction, there might be ambiguity in how to treat software in an academic context, which would affect its documentation, sharing, and evaluation.

Research software is built with the intention of solving specific scientific problems, and thereby, it helps to produce new findings or answers to research questions. It is not an ancillary tool but an integral part of the knowledge production process itself—indeed, deeply integrated within the objectives and methods of the research project. It has been designed to satisfy the specific needs of a scientific area, enabling experiments, simulations, or analyses that would otherwise be very difficult or impossible to realise.

On the other hand, the software used in academic research includes general-purpose tools not developed for any specific research project but widely used by researchers for data development, management, and analysis (GRUENPETER ET AL., 2021). Such tools include programming languages like Python and R, operating systems like Linux and Windows, statistical analysis programs like SPSS and Excel, database systems like MySQL, and machine learning frameworks like TensorFlow. The above tools are generic in nature and can be used in different scenarios; thus, they help researchers establish their projects but do not specifically deal with certain scientific problems and

do not belong to the key parts of the scientific knowledge creation toolchain (OZGUR ET AL., 2020).

I.III Recognizing Research Software as a Scientific Product

Although research software is increasingly popular among scientists, its visibility in academia as a scientific product remains inadequate and is often underappreciated. Unlike other forms of research outputs, such as articles, datasets, or monographs, software is seldom regarded as an independent contribution, although it is a vital part of the process of scientific knowledge production. This imbalance not only constrains the scholarly contribution of the work but also undermines the transparency, reproducibility, and sustainability of its scientific findings. Research software is not merely a technical tool; rather, it embodies scientific methods, ideas, and protocols that frequently determine the overall validity of an experiment or analysis. Yet, to be appreciated for what it is worth, the scientific community needs to have well-defined policies regarding the documentation, citation and appraisal of software. For instance, the publication of DOIs for software, as is done for articles and datasets, and adopting certain peer-reviewing criteria are indispensable steps in bringing it entirely within the purview of scientific communication (JAY ET AL., 2021). With this advancement, there has been a staggering growth in recent years of new forms of scientific publications dedicated to documenting and publishing research software. The above-discussed works, which may be classified as *software papers*, are characterised by their detailed description of the newly developed software along with the source code included. These works are typically kept in specific repositories and are beyond mere presentations of the practical realisation of the software. They explore several aspects, like the process of developing the software, the source code, documentation, its applications, and its scientific consequences. The emergence of *software papers* has not only given rise to specialist journals but has also encouraged traditional journals to include sections for this type of contribution. This shift in scientific publishing culture has implications for both new challenges and possible opportunities for scholarly communication. Journals such as the Journal of Open Research Software¹, among others, have shown that software can be considered a scientific object in its own right, with *FAIR* (*Findable, Accessible, Interoperable, Reusable*) metadata and full documentation (WEBER, 2022). Beyond making it easier to distribute and reuse the software, this approach contributes to making research software more visible in academia and helps move software forward as a valid scientific product. Besides this, it involves recognition of the work of software developers as researchers, integration of software into professional

¹ <https://openresearchsoftware.metajnl.com/>

development standards, and facilitating joint interdisciplinary collaboration through code sharing. Not only do these activities legitimise the role of software in scientific research, but they also enhance the sustainability and innovation potential of the research activities themselves.

I.IV The Central Role of Research Software

The use of software allows for the automation of complex procedures, increasing operational efficiency and reducing the chances of human error, hence helping in the reproduction of findings. A recent survey at 15 universities in the Russell Group found that 92% of researchers use research software, and 69% say that their work would not be possible without it. Moreover, 56% reported having written their own software, 70% of which are men and 30% women (HETTRICK, 2014).

However, despite this ubiquity, there is a great deal of diversity in the tools used by academics. An initial analysis of an alternative survey, run between 2014 and 2016 with 1,261 respondents, collected 2,958 unique responses about the software used (HETTRICK ET AL., 2016). After data cleaning processes, the number of individual pieces of software was reduced to 1,097, highlighting a wide variety of tools. However, the results also show a strong concentration on a relatively small number of software packages: only 35 tools were mentioned by more than 10 people, and the top five accounted for almost a quarter of all responses (665 out of 2,958). This concentration points to the dominance of certain tools (Figure 1)—such as Python, MATLAB, and R—that have become industry standards due to their robustness, flexibility, and community support. Moreover, a substantial number of the packages (2,654) received merely a single citation, indicating that despite the usage of various tools, a considerable segment of researchers depends on a limited selection of prevalent software packages.

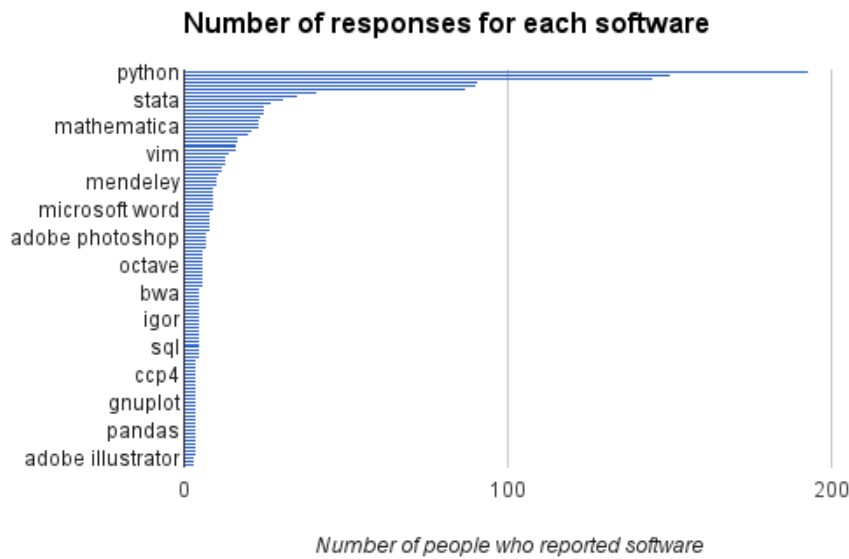


Figure 1. *Programming Languages Used in Research Software*

I.IV Barriers to Code Sharing in Scientific Research

Scientists face several challenges in sharing data and code. A study conducted as part of the SORTEE conference² explored these issues, leading to the publication of the paper “*Why don’t we share data and code? Perceived barriers and benefits to public archiving practices*” (GOMES ET AL., 2022). During this discussion, the primary challenges associated with data and code sharing were elucidated, drawing upon the personal experiences of the researchers involved. Subsequently, participants engaged in three supplementary discussions aimed at reinforcing and elaborating on the identified barriers while incorporating insights from the extant literature. Each author contributed a distinct section, which was subsequently subjected to collective evaluations. The findings categorised the motivations into three broad categories: *barriers to knowledge, reuse concerns, and career disincentives* (Figure 2).

² The SORTEE conference, launched in December 2020, is an event organized by the Society for Open, Reliable, and Transparent Ecology and Evolutionary biology (SORTEE), an international organization that brings together researchers committed to improving reliability and transparency in research in ecology, evolutionary biology, and related disciplines (“ABOUT,” N.D.)

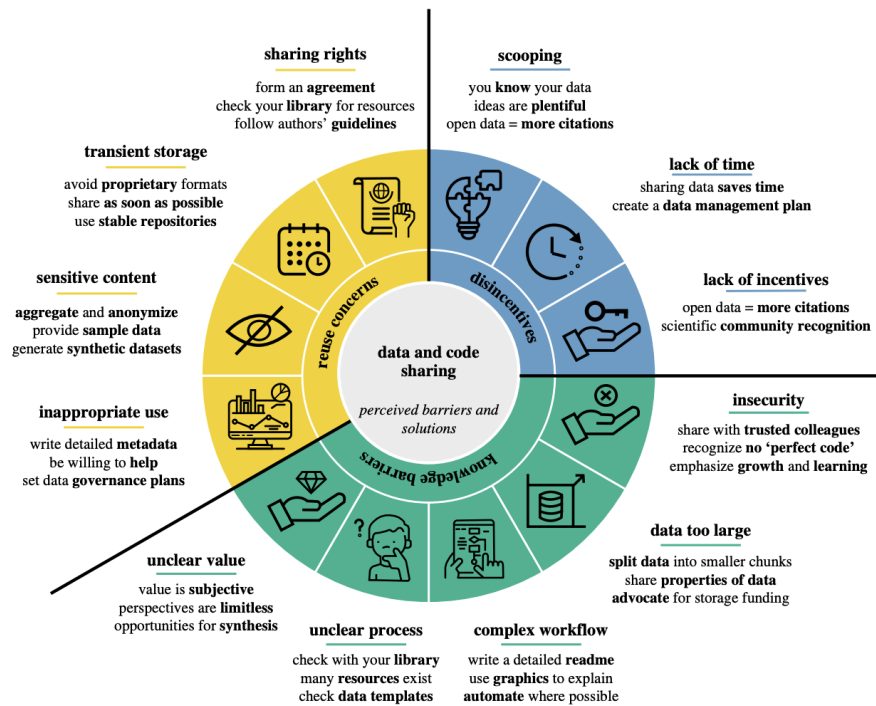


Figure 2. The main barriers and solutions for data and code sharing identified during the SORTEE conference

Barriers to knowledge: the lack of proper training in data sharing and storage is a significant hindrance. The majority of researchers lack the necessary expertise to store and organise their data effectively, and choosing the most appropriate repository becomes daunting with the vast number of available choices, each with detailed guidelines and exacting requirements. This ambiguity gives rise to suspicion in the sharing process, and many become more conservative in their approach.

These are supported by empirical data: 46% of the respondents point to the existence of structural barriers in enforcing protocols for both data usability and accessibility, while 33% point to the selection of hosting platforms and repositories as a critical barrier to systematic data sharing. Another major challenge lies in research workflows, which are often sophisticated in nature and comprise a chain of manual operations. Where these workflows are not properly documented, it is hard for others to replicate or even understand how they were executed, thus detracting from the reproducibility of results. Their use of specialised or proprietary software only exacerbates the problem because not everybody possesses access to this software or to training in it. The majority of

researchers are discouraged from publishing their work for fear of exposing errors or potential flaws in their methodologies.

One of the largest challenges is the growing size of data sets. The growth in volume and complexity of data makes management and storage more difficult, especially without the right tools. In addition, the movement of large data sets becomes cost-prohibitive, even more so in terms of infrastructure and expense, and adds additional challenges to security and privacy, especially when sensitive information is involved.

The influence of personal insecurity is also profound. The fear of being criticised or scrutinised, particularly among first-career scholars, creates a reluctance to offer information. The possibility of errors jeopardising professional reputation also discourages them from engaging with the wider public.

Finally, most scientists underestimate their data as secondary in significance compared to that from more high-profile research. This perception further reduces the motivation to publish their research. In some disciplines, current intellectual property cultures exacerbate these concerns, evoking fears about losing control over findings or needing to contend with abuses of their research.

Reuse Concerns: Most researchers fear that open sharing of code and data may result in misinterpretation or misuse by others. This is because future users may lack a complete understanding of the nuances, assumptions, or confounding variables associated with the data, which may result in drawing wrong conclusions. Moreover, even detailed documentation may fall short in bridging these gaps, since some critical information is only known to the original data collectors. Furthermore, the fact that data and code are attributed to individuals or groups of individuals creates resistance to letting go of sole ownership over their utilisation, hence the additional resistance. These fears are linked to complicated intellectual property issues. Data are frequently legally owned by institutions or funding agencies, not by individual researchers, or are derived from sources with alternative rights, hence making it difficult to establish who has the right to disseminate them. Once published, challenges are amplified: some publishers demand the assignment of copyright, even data and code, thus restricting their usage. Furthermore, when such research products are placed on publishing platforms, they can be subject to paywalls, further restricting author control and public access. Another significant concern pertains to the publication of sensitive data, which calls for serious consideration of ethical and social risks. In some fields, such as biology, the information distribution can have harmful impacts, such as the facilitation of poaching or ecosystem destruction. Similarly, publishing data with identifiable personal details, for example, interview responses or personal information, creates ethical problems, especially when it comes to engagement with traditionally marginalised communities. Such groups are often vulnerable to exploitation, and so they understandably lack faith in scientific study, an attitude which is only amplified by the history of their data or natural resources abuse. Publishing without clear permission or without contextualisation may

contribute to unethical behaviour and endanger the data subjects, increasing tensions between research and society.

Long-term data and code archiving poses several challenges. The lack of a stable archiving infrastructure is one of the significant challenges. Transient sites such as GitHub or cloud computing are most commonly used by researchers. These sites are not guaranteed long-term existence. Their loss can occur in the case of account suspension or researchers quitting their academic positions.

Proprietary formats are another concern. Papers created with specific software, such as Microsoft Excel or SAS, can become obsolete with the lapse of time, thus threatening future access. Even if source code is available, it may be difficult to recreate a work due to differences in execution environments, different versions, or unsupported software dependencies. Such cases undermine reproducibility, which is one of the essential aspects of scientific inquiry.

Finally, the lack of adequate funds in place for long-term conservation is also a disincentive. Lack of funds discourages researchers from taking the effort and time to make their research output available in the future.

Career Incentives: One of the biggest hindrances to sharing code and data is fear of being scooped, which refers to the act when another researcher takes publicly available data and conducts analyses that the original author intended but had not yet performed. To this concern is added the fear that making code available for free will reduce the potential for collaboration and co-authorship with colleagues who may have an interest in the work being continued. Another concern is that early release of information can undermine the researcher's ability to publish in a timely manner, especially in a culture where the number of published papers is seen as an important measure of productivity and competitive position. The lack of publication outlets thus represents a significant expense for the majority of researchers.

A second justification for not sharing data is the expected time demands, which can take the form of both short- and long-term requirements. In the short term, the demands of structuring, preparing, and annotating data, code, and metadata for archiving can be particularly onerous, especially if such materials were not well organised from the beginning of the project. In the long term, researchers are also worried about having to keep checking and helping people who want to reuse their data or code since this is time-consuming.

Finally, in addition to the perceived cost of sharing code and data that has already been mentioned, there is a profound lack of perceived incentives for most researchers. Historically, there have been few career benefits for data and code sharing, which have also contributed to resistance to such an activity. In the absence of official recognition or tangible rewards, most researchers are not able to perceive any immediate benefit in putting their research output in the public domain.

I.V FAIR Principles Applied to Research Software: Towards Greater Transparency and Reusability

Besides streamlining scientific efforts, research software is also important for the reproduction and verification of results. This is because clear code and a number of well-documented tools allow colleagues in the field to verify the findings that are attained, hence fostering a much more open and collaborative research environment. However, this means that research software should adhere to the FAIR principles: Findable, Accessible, Interoperable, and Reusable, ensuring continued usability and impact within the broader scientific community (FELDERER ET AL., 2023).

Originally developed for the handling of research data, the FAIR principles are increasingly being applied in the context of research software in order to enhance its transparency, accessibility, and reusability (BARKER ET AL., 2022). The global community adopted the FAIR principles and developed them, creating the FAIR4RS Principles for research software. They were laid by the FAIR4RS Working Group through an international joint activity of the Research Software Alliance (ReSA), Future of Research Communications and E-Scholarship (FORCE11), and the Research Data Alliance (RDA).

- **Findable:** Software should be easily findable with proper metadata and dedicated catalogues. Sharing code in open repositories, complemented with unique persistent identifiers (e.g., DOIs), supports this traceability. The FAIR4RS Working Group takes the view that adherence to this principle helps to overcome the challenges of discoverability faced by scientific software.
- **Accessible:** The software should be accessible for all, both legally and technically. This includes the use of appropriate licenses and the availability of source code on open platforms. Also, metadata should be accessible even when the software is no longer available, which provides continuity of information.
- **Interoperable:** The design of the software should allow it to work with other tools and data, facilitating exchange through standard formats and defined protocols.
- **Reusable:** Complete documentation of the software is a must, enabling its use in different contexts. A clear license is needed in order to define the acceptable uses of the software. Organisations like the efforts of the eScience Center in the Netherlands have put in place Software Management Plans that ensure that code will satisfy reusability criteria, including compliance with provenance and attribution policies.

I.V.I Key Issues in the Application of FAIR Principles for Software

"Software is data, but it is not just data" (KATZ ET AL., 2021).

This intrinsic characteristic of software makes the implementation of FAIR principles in software development particularly complex and full of challenges. These include the intrinsic nature of software systems, the need for consistent implementation, and the integration of FAIR principles with existing workflows and technologies.

- **Inconsistency and Complexity:** FAIR principles are broad directives that may be understood in different ways, leading to varied implementations within different communities. This inconsistency affects interoperability and the general acceptance of the FAIR principles (JACOBSEN ET AL., 2020; LAMPRECHT ET AL., 2020).
- **Integration with Current Frameworks:** Most scientific computing communities, particularly those utilising high-performance computing (HPC), are struggling to integrate FAIR principles because of the intricacy of their workflows and the details of their systems. They include handling metadata, persistent identifiers, and reproducibility;
- **Metadata and Persistent Identifiers:** One of the main challenges is the volume of metadata and persistent identifiers required to facilitate software reuse and reproducibility. This involves determining what to publish, how to "FAIRify" data, and the level of granularity of persistent identifiers (POUCHARD ET AL., 2022; POUCHARD, 2023);
- **Interdisciplinary Cooperation:** It involves cooperation from various stakeholders, including data scientists, user experience researchers, and software developers, to implement FAIR principles. This is necessary to resolve both technical as well as socio-technical challenges but is hard to attain due to varying priorities and knowledge (RYAN ET AL., 2023; SOREMEKUN ET AL., 2022);
- **Inadequate Standardized Solutions:** Standardized solutions and approaches that may be reused across communities are essential to enable the scale-up of FAIR principles. Without standardised solutions, there can be ad hoc implementations that are unsustainable or non-scalable (LU ET AL., 2023).

I.VI From FAIR Principles to the Open Science Movement

The FAIR principles are fundamental to Open Science, a global movement that promotes collaboration, transparency, and the open sharing of data and scientific results.

Open Science can be defined as: *"(...) an approach to the scientific process based on collaboration, open and timely sharing of results, dissemination methods based on digital network technologies, and transparent methods of validation and evaluation of research products. Open Science enhances the effectiveness of collaboration and the reproducibility of research results. It increases collaborative potential with the*

possibility of accessing data and reusing it for new analyses, including interdisciplinary studies, as well as for scientific teaching, and it enhances the accessibility of scientific knowledge in a transparent manner for the benefit of society” (ROSSI ET AL., 2022).

Open Science also includes, besides data, the research software critical for the generation and validation of scientific discoveries. In many respects, similar to data, the software has to be open and transparent, thus reusable, to reproduce the methods applied in the research and results verification or expansion by other scientists. The process of making software accessible makes it possible not only to replicate results but also to improve the methodologies of science over time, feeding a virtuous cycle of innovation in which every researcher can participate in the ongoing refinement of research tools. Indeed, Open Science includes a wide range of practices addressing different aspects of the entire research cycle, typically including open access to publications and open scientific communication methods.

In fact, Open Science encompasses a variety of practices that cover different aspects of the entire research cycle and usually includes (“I PRINCIPI, GLI OBIETTIVI, I BENEFICI E LE PRASSI DELLA SCIENZA APERTA - SISTEMA BIBLIOTECARIO DI ATENEIO — SBA,” N.D.):

- *Open access to publications and open scientific communication methods*
- *Open research data managed according to the FAIR principles (Findable, Accessible, Interoperable, Reusable) and a data management plan*
- *Free software*
- *Open workflows, notes, experiments, and research projects*
- *Participatory science (citizen science)*
- *Open resources for learning and teaching*
- *Alternative methods for research evaluation, including open peer review.*

I.VI.I The Phenomenon of Open Science in Europe

Europe has taken up a lead role in resourcing and framing policies to promote the adoption of the FAIR principles within the context of the global Open Science movement. Most notably, a number of large-scale European initiatives have been designed to create a research infrastructure capable of facilitating access, sharing, and reuse of data, software, and scientific publications for the promotion of Open Science globally.

In this regard, Open Research Europe (ORE) and the European Open Science Cloud (EOSC) are two concrete initiatives that embody Europe's commitment to building a research system that is more transparent, open, and collaborative.

Open Research Europe (ORE) is an open access publishing platform funded and managed by the European Commission, set up to host publications from EU-funded projects in all scientific disciplines (EUROPEAN COMMISSION. DIRECTORATE GENERAL FOR RESEARCH AND INNOVATION., 2020). While publishing in ORE is not mandatory for EU-funded projects, it offers a convenient way to meet Open Access requirements, supports open peer review practices, and allows the early sharing of results without charges for authors. Submissions are immediately published as preprints under a CC-BY license after a formal screening and are then reviewed by external referees. The reviews and reviewer identities are publicly available, which provides transparency in the review process.

The European Open Science Cloud (EOSC) is a European Union project initiated in 2016 to promote the use of the FAIR principles—that is, Findable, Accessible, Interoperable, and Reusable (“EUROPEAN OPEN SCIENCE CLOUD (EOSC) - EUROPEAN COMMISSION,” 2024). Its official definition states that EOSC is a “*federation of research infrastructures (data) designed to enable the Web of FAIR Data and Services and to support researchers in performing Open Science, promoting the openness and use of data, publications, and code.*” All elements in the EOSC strategic plan fully support the workflows of scientific research beginning in 2023.

I.VI.II Strategy and Initiatives for the Implementation of the FAIR Principles and Open Science in Italy

The relevance of Open Science in the European context, which is taken due heed by bold policy and program formulation like the European Open Science Cloud (EOSC) and the Open Research Europe (ORE), reverberates down to the level of individual countries, where nations like Italy have established strategic plans—namely, the “*Piano Nazionale per la Scienza Aperta*”³ (PNSA)—to achieve common goals for the open, accessible, and implementable FAIR principles.

PNSA is one of the fundamental parts of the National Research Programme for the years 2021-2027, differentiating specific strategies that purport to underpin the transition to an open research scheme consonant with European efforts. The set objective of the plan is to support the implementation of the FAIR (Findable, Accessible, Interoperable, Reusable) principles through actual actions, which are the purpose-built national infrastructures for scientific data management and dissemination and the uptake of the European Open Science Cloud (EOSC).

One of the main objectives of the strategy within the PNSA is the establishment of an environment that envisages timely open access to nationally-financed results of scientific works in an interoperable manner, as well as the implementation of policies

³ It translates to “*National Plan for Open Science*”

regarding the forced deposit of publications in open repositories. Therefore, the strategy provides for the setting up of a national institutional repository that would adhere to European services, apart from the strategies of transformative contracts in the advocacy for fostering sustainability in open scientific publishing.

Relatedly, the PNSA advocates for the generation of FAIR-by-design approaches in developing research data, including advanced methodologies like automatic metadata creation, in the preparation of data management plans, and in the training of staff, such as data stewards and data scientists. Other important aspects to be improved that the PNSA acknowledges include the facilitation of inter- and multidisciplinary research, the development of working collaboration frameworks, and improvements in the most advanced technological infrastructures, such as high-performance computing and artificial intelligence services.

The policy will aim to increase Italian participation in the EOSC at the European Union level and bring national policy into line with global standards. It will focus on the management of open science practices for the assessment of measurable progress in raising the transparency of research work. Besides, the PNSA mentions the establishment of a national point of access dedicated to the FAIR sharing of COVID-19 epidemiological and clinical data to foster health emergency management and research in public health (ROSSI ET AL., 2022)

Chapter II Software Journal: History and Theory of a New Paradigm in Scholarly Communication

This chapter examines the historical development of *software papers*. The form of academic paper has developed significantly over the years, propelled by the emergence and growth of specialised scientific journals referred to as *software journals*. Such journals, which are either partially or entirely devoted to research software, have been instrumental in turning the *software paper* into a valuable means of enabling transparency, reproducibility, and sharing of scientific software. Figure 3 presents a timeline highlighting the most significant events in the history of *software journals* and Open Science. Each event marked on the timeline is further examined in the corresponding sections of this chapter.

This chapter's historical trajectory is organised into a number of key phases. The first section is concerned with a theoretical approach, critically examining the seminal work of Vannevar Bush articulated in his article *As We May Think* (cf. [Sec. II.I](#)). In this work, Bush presented a revolutionary future of scientific scholarship, anticipating basic principles of knowledge management and sharing that would, years later, make advancements in the software paper possible. The chapter then examines the scholarly and historical background that led to the establishment of two foundational *software journals*: *Computer Physics Communications (CPC)*, the first *pure software journal* focusing on its editorial practice, organisation, and role of the *CPC Program Library* and *ACM Transactions on Mathematical Software (TOMS)*, the first hybrid software journals with attention to its editorial approach and the role of *CARGO* (cf. [Sec. II.II](#)). Another landmark in the evolution of software documentation is the start of the adoption of the *GNU General Public License (GPL)* during the 1980s (cf. [Sec. II.III](#)). Although not directly related to publishing scholarship, the GPL played an important role in the sharing and distribution of *software papers* in that it helped to foster a sense of openness and cooperation. As such, this milestone indirectly supported the evolution of the idea of the *software paper* in itself. During the 1990s, the gradual digitalisation of scientific journals brought a dramatic change in the publication and dissemination practices of code (cf. [Sec. II.IV](#)). The fourth paragraph emphasises the transformation that scholarly publication experienced in this period, with a shift toward electronic media that progressively diminished the distribution of paper. It further provides the history of a model example of the *software journals* during the 1990s, the *Journal of Statistical Software (JSS)*, as the sole *pure software journal* established during that decade, coinciding with the launch of the Open Source Initiative (OSI). In the new millennium, the scientific publishing world continued to undergo more transformation

due to the proliferation of digital repositories (cf. Sec. [II.V](#)). Platforms like *GitHub* and *Zenodo* have transformed how software is shared, making it more open and reusable. In the following, the journals *Journal of Open Research Software (JORS)*, the *Journal of Open Source Software (JOSS)*, and *SoftwareX* are analysed mainly in relation to the history of *software journals* and *software papers*. The three cases have been chosen since they all present clear editorial objectives in their first issues, thus giving valuable information about the genesis of this particular publishing model.

This chapter starts on a path beginning with the initial theoretical viewpoints and moves through the major events and technological breakthroughs that have been instrumental in the emergence and evolution of *software journals*. It will enable an understanding of how *software papers* have become ever more a means of recognition and communication of software in research in academia.

II.I "As We May Think" and the Precursor Vision of Software in Scientific Research

In the historical evolution of software recognition in scholarly journals, it is interesting to look at Vannevar Bush's⁴ article, "As We May Think" (BUSH, 1996), not merely as a prophetic glance at *software paper* but a thoughtful examination of the problems of scientific communication that remain part of the modern debate.

His vision was for the power of technology to revolutionise scientific information management. The "memex," the hypothetical machine he wrote about, was imagined as a mechanised desk that used microfilm to create "trails" between documents. Technically different from modern digital systems, the principles behind the Memex—i.e., the creation of associative links between documents similar to human thought—prefigured key aspects of knowledge management systems today.

Bush recognised that the continuous expansion of scientific literature made it necessary to develop more effective methods for storing, organising, and correlating information. He conceived technology as a tool for enhancing human capabilities. Although initially framed through the use of analogue and mechanical devices, this idea remains relevant today in the era of data analytics software and open science platforms.

⁴ Vannevar Bush (1890–1974) was an American engineer and scientist who played a key role in science policy during and after World War II. He directed the Office of Scientific Research and Development (OSRD), coordinating technological and scientific research, including strategic projects such as the Manhattan Project.

Timeline of events related to phenomenon of Software Journals and Open Science

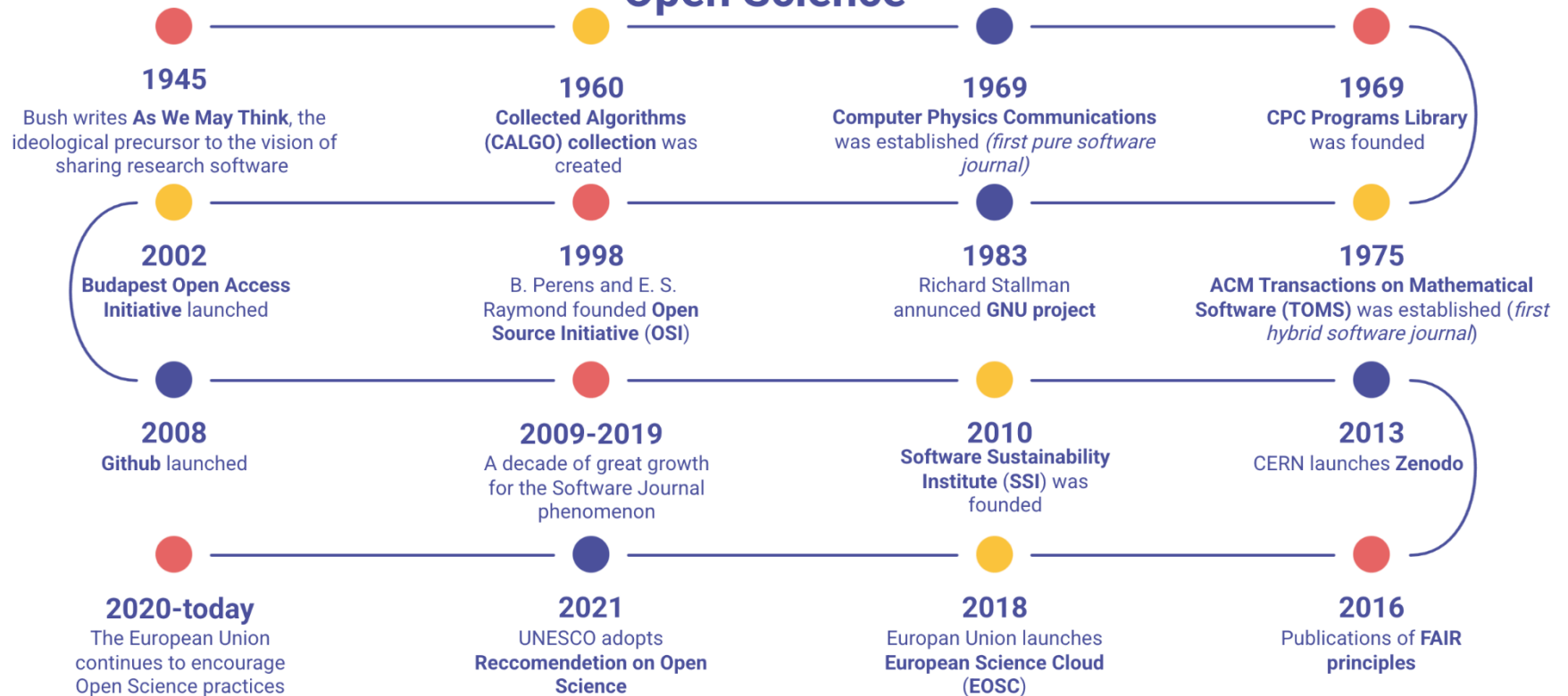


Figure 3. Timeline of events related to the phenomenon of Software Journal and Open Science

The comparison between Bush's conceptual design and modern *software journals* highlights several similarities. First, both share a structural isomorphism: the *Memex*, designed as a system for linking and organising scientific knowledge, parallels *software journals*, which serve to systematically document and interconnect computational resources used in research. Moreover, Bush's vision of making science more reproducible and accessible is reflected in the primary goal of *software journals*, ensuring the reusability and verifiability of scientific software. *Software journals* undergo challenges that Bush had anticipated, albeit in a different setting: how to make scientific work reproducible, accessible, and built upon the shoulders of past research. Dissemination of scientific software remedies these problems by making code—a now central instrument of inquiry—part of the published, citable scientific record. This instrument realises, in an unexpected but powerful way, Bush's vision of a more networked and open scientific community.

The influence of “As We May Think” on *software journals* is not merely a straightforward prediction of technological progress. Rather, it lies in its ability to highlight fundamental challenges in scientific communication—challenges that still require creative solutions today. This demonstrates its remarkable relevance over time.

II.II The Origins of Software Journals: The First Pure and Hybrid Publications

II.II.I Computer Physics Communications and CPC Program Library

Scholarly software publishing began to come into its own in the 1960s and 1970s, as exemplified by *Computer Physics Communications* (1969), which, as described in the analysis reviewed in [Chapter III](#), was the first journal that published *software papers*. These articles, with their inclusion of algorithms and source code snippets in the appendices, were quite an innovation for their day. Software, indeed, at that time was believed to be a simple tool to validate experimental or theoretical results rather than a scientific contribution in its own right. The code available was often poorly documented and nonstandard, thus making it hard for other researchers to use. Also, as compared to theory writing, software would not typically go through the formal peer review process. Lastly, prior to the advent of the digital age, software used to be distributed using physical media, e.g., magnetic tape, hence making sharing and accessibility hard among the scientific community.

The *Computer Physics Communications* journal was initiated to promote the sharing of computer programs and technical expertise in the field of physics, as outlined in its inaugural introduction (BURKE, 1969). The primary objective was to publish detailed accounts of relevant physics software, modifications of established programs,

contributions directed towards specialised applications, and notices of data compilations. Besides, the journal was also meant to act as a platform for research articles on best practices in programming, criticism, and review of hardware and software products from vendors. One key feature emphasised at the time was its purpose as a forum dedicated to developing and propagating effective programming methods, with the expectation of impacting future directions in scientific computing technologies. As the first introduction indicated, the journal had two principal components: a "Summary" and a "Detailed Description" ("INSTRUCTIONS TO AUTHORS," 1969). The Summary, based on a standardised form, provided an initial determination of the program's relevance, specifying essential technical details such as the computing environment and dependencies. The Detailed Description (or *Long Write-up*), restricted to 16,000 words, is aimed at presenting all the information necessary for the replication and comprehension of the program's functionality. Rather than publishing complete program listings, the journal focused on documenting execution test results, which were included in the publication as photo reproductions to facilitate verification. The journal also established a rigorous submission and verification process for software contributions. Programs were initially submitted as manuscripts, and execution results were to be reviewed by referees. Upon acceptance, authors were required to submit their program decks, initially on punched cards (or paper tape for ALGOL programs), ensuring a standardised format. Programs were catalogued, and their corresponding metadata (e.g., dependencies and machine requirements) were recorded for easy reference.

Computer Physics Communications is also known for its *CPC Program Library*, a collection of programs described in the articles it published. This library was originally hosted by Queen's University Belfast and was later transferred in 2016 to the *Mendeley Data repository*, containing over 3,500 programs from 1969 to that year (SCOTT ET AL., 2020). This initiative played a significant role in promoting the availability and sharing of scientific software.

The future objectives between the 1960s and 1970s were to encourage collaboration among physicists, thereby developing a network dedicated to sharing knowledge and methodology, and to guard the interests of the scientific community in the computer science field, promoting technological advancements in scientific computing.

II.II.II ACM Transactions on Mathematical Software (TOMS) and CALGO

Another one of the significant milestones along the way toward accepting software as a scientific contribution came with the establishment of *ACM Transactions on Mathematical Software (TOMS)* in 1975, which, as will later be discussed in more detail in [Chapter III](#), was the first *hybrid software journal*. In his inaugural editorial, John R. Rice characterised it as an editorial undertaking to accept mathematical software as a

distinct scientific entity (RICE, 1975). This editorial project was motivated by the absence of a clear academic platform for the critical evaluation and allocation of computational mechanisms, which traditionally have been relegated to marginal positions in scholarly discourse. The methodological innovation of TOMS developed along two dimensions: a vertical dimension that accounts for seven levels of software analysis—algorithmic processing to maintenance—highlighting the intermediary construction, analysis, and critical evaluation stages; and a horizontal dimension that covers several computational domains, including polynomial manipulation, numerical optimisation, statistical computation, and symbolic algorithms, thus promoting an interdisciplinary discourse that had otherwise been disjointed. One of the distinguishing features of the publishing infrastructure is the two-publication strategy, separating "person-to-person" (representing software articles) and "machine-to-machine" (representing the distribution of the software) communication, thereby prefiguring modern practices of scientific dissemination by the Algorithms Distribution Service (ACM TOMS STAFF, 1975). The service is not a channel of distribution in the form of a straightforward platform; instead, it is an ordered system intended to provide access to certified computational tools in consideration of the considerable financial and intellectual investment that goes into software development, estimated at 5 to 20 dollars per line of code.

Rice's foresight, well in advance of his contemporaries, anticipated and influenced modern methods of scientific dissemination. It is part of a larger context that creates a paradigm that views software not as a bare-bones technical tool but as a distinctive and primary intellectual property critical to the progress of scientific knowledge.

A seminal example of the TOMS model is provided by the *Collected Algorithms (CALGO)*⁵ collection, an algorithm repository whose roots stretch back to 1960, a full fifteen years before TOMS itself was founded (HOPKINS, 2002). CALGO was initially associated with *ACM Communications* until 1975 when it was formally subsumed under TOMS as a core part of its editorial agenda. It published all 331 of its algorithms in Algol 60 only, mirroring the dominant technology trends of the day. Editorial policy changed dramatically in September 1966 with the acceptance of the Fortran language, although the first Fortran code was not published until June 1968. It should be mentioned that at this early period, code was distributed in the form of direct author submissions without peer review. Quality control was a communal effort: several interest groups replicated the code, ran independent tests, and communicated inconsistencies or requests for modifications necessary to enhance performance, along with other significant concerns such as security and reliability. The CALGO method preceded contemporary code-sharing practices by incorporating such features as version control, standardised documentation, and validation tests—features that were novel in the 1980s and 1990s.

⁵ <https://calgo.acm.org/>

II.III The GNU Project

The advent of special journals in *software papers* with the practice of releasing the software itself would not have been possible without certain events in history that marked the evolution of open source software. The spread of source code in the scientific community, indeed, is intimately connected with these events. A model of this process is embodied in the GNU project⁶, initiated by Richard Stallman in 1983. The project was conceived as a reaction to the increasing proprietary closure of software, a situation that restricted users' freedom and prevented the ability to study, change and share the source code. Throughout his years at MIT, Stallman experienced great hardship in getting access to the source code of tools that, up to that point, were freely modifiable and improvable by the scientific and academic community. To reverse this trend, Stallman advocated the GNU project to develop a totally free operating system founded on the ideals of sharing and cooperation. In 1985, to continue to aid this effort, Stallman established the Free Software Foundation (FSF), an organisation dedicated to the promotion and protection of free software, guaranteeing freedoms to use, modify and distribute software worldwide.

One of the significant aspects of the GNU project was the formulation of the GNU licenses, specifically the GNU General Public License (GPL), a copyleft license, which mandates that any derivative work of the original software must be distributed under the same terms. In practice, among the key technical provisions of the GPL, we have:

- **Source code distribution requirement:** All distributions of the software, both in its original and modified versions, should be accompanied by the full source code.
- **Copyleft restriction:** It's not possible to re-license the software under more restrictive terms; all derivative works must carry the same license, thereby ensuring the perpetuation of the freedoms afforded initially.
- **Documentation of changes:** Any deviations from the original code must be thoroughly documented to facilitate the traceability of introduced changes.

Three versions of the GPL have been published since its inception, each responding to the need for refinement, enhancement, and adaptation to emerging challenges and advancements in technological and legal frameworks. GPL v1, released in 1989, established the fundamental principles of copyleft, guaranteeing free software distribution along with access to the source code and requiring any derivative work to be released under the same conditions as the original license, thus laying the foundation of a model of open and collaborative development, founded on sharing and modifiability of software ("GNU GENERAL PUBLIC LICENSE, VERSION 1," 1989). Later, GPL v2, which was published in 1991, solidified those principles, achieving more

⁶ <https://www.gnu.org/gnu/thegnuproject.html.en>

regulatory precision and enhancing defence against patent litigation and narrow interpretations that might undermine the freedom to modify and utilise software, thereby establishing a stronger legal framework that better aligns with the newer demands of the free software community (“GNU GENERAL PUBLIC LICENSE, VERSION 2,” 1991). GPL v3, released in 2007, continued to develop the legal framework. Notably, this version added specific protections against tivoization⁷, dealing more explicitly with patent and digital rights management (DRM) concerns and enhancing the international compatibility of the license (“GNU GENERAL PUBLIC LICENSE, VERSION 3,” 2007).

As will be discussed later in Chapter III, it is customary for journals of software to apply some licenses when publishing a *software paper*, and GNU licenses are widely adopted in such a case.

II.IV The Rise of Electronic Journals and the Early Practice of Software Code Sharing in the 1990s

In the 1990s, the dissemination of software code through academic journals remained occasional and lacking in strict control. However, nascent trends that would grow to gain acceptance in coming decades were only beginning to develop. During this time, academic publications relied heavily upon the printed medium, with the vast majority of works adhering to traditional structures. But, as listed in the *ARL Directory of Electronic Journals, Newsletters, and Academic Discussion Lists*, the list held just 27 electronic journals in 1991 (LENARES, 1999). Since 1993, there has been a noticeable growth in the number of journals from 45 to 181 as of 1994. As of the year 1995, the number was 306, subsequently going up to 1,093 as of the year 1996 and then to 2,459 in 1997. This rapid increase has been driven largely by the proliferation of the Internet and the World Wide Web; however, much of the growth, particularly after 1996, has been possible because commercial publishers with commercial interests at stake have led the way in creating digital journal editions. The advent of electronic journals marked a significant step forward in the evolution of *software paper* publishing. The transition to digital medium facilitated the distribution of source code and data, transcending the intrinsic limitations of the print medium, which was not well suited for the dissemination of computational resources.

Sharing source code was not the normal practice; however, a few researchers began distributing code segments during the early period of online databases through direct

⁷ The concept of "Tivoisation" was first introduced in relation to the TiVo device, a digital video recording (DVR) system (“An Introduction to Tivoization,” n.d.). The device took steps that prevented users from changing and reinstalling modified versions of its software, despite the fact that it ran Linux, an open-source operating system. It thus denotes a process of using open source software in an appliance; however, the usage comes with modifications that restrict the freedom of the user to distribute or change it, even though there is a possibility that the source code is actually available. The term is usually applied within the context of open source licenses, and especially the GPLv3, which has anti-Tivoisation provisions that try to secure the freedom of the users to modify and distribute the software on the devices they utilize.

requests via electronic mail or hard media. In specific fields of research, theoretical computer science, and software engineering, it was occasionally regular practice to append the code as part of an article or to clearly state it using algorithms and pseudocode. This tendency was most substantial in specialised journals and methodological studies, where the merit of reproducibility of findings started to be appreciated as a constituent component of scholarly research. Even with these early signs of receptiveness, sharing software code was still a small-scale and not heavily administered undertaking. A few journals did start experimenting with citation policies for software, but there were no communal protocols for reviewing and archiving code.

II.IV.I Journal of Statistical Software

A case in point for how editorial standards relating to a *software journal* developed during the 1990s is the establishment of the *Journal of Statistical Software* (JSS) in 1996 by Jan de Leeuw at the University of California, Los Angeles (LEEUEW, 2009). As will be shown in the analysis in [Chapter III](#), JSS stands out as the only pure *software journal* founded in the 1990s, making it particularly significant for our historical overview. Leeuw initiated this journal to fill the gap created by the closure of the software section of *Applied Statistics* and the lack of a structured review for software contributions in the academic field. From its very beginning, the JSS adopted an editorial format of papers with supporting source code, usually presented as user guides that could elucidate the relationship between statistical theory and computer implementation. Among the characteristics that defined the JSS was the implementation of a peer review process that was not only applied to the papers but extended to the code, which is an essential element in ensuring the quality, validity, and reproducibility of the methods described. The journal has enabled the dissemination of software in various commercial and open-source settings; however, it has increasingly joined the open-source community in supporting openness and the dissemination of software distributed under open licenses. Notably, the phenomenal expansion of the R community during the 2000s created a dramatic spike in submissions addressing this programming language, ultimately accounting for approximately 70% of the contributions. Technically, the JSS has undergone several phases of infrastructural improvement. It used to operate on a stack of Apache, PHP, and MySQL; however, in 2007, the site moved to Ruby on Rails to enhance the management and usage of the articles, along with the addition of an RSS feed for automatic updates.

II.IV.II Open Source Initiative

In 1998, the Open Source Initiative (OSI) was established (“HISTORY OF THE OSI,” 2006). Bruce Perens and Eric S. Raymond founded the OSI and came up with the Open Source Definition (OSD), which is a list of ten criteria that determines the character of

open-source software (“THE OPEN SOURCE DEFINITION,” 2024). They include the freedom to distribute the software, the freedom to access the source code, and the freedom to create and distribute derivative versions. This project has made the difference between free software and proprietary solutions clearer, delivering a legal and conceptual basis, as demonstrated in Section [III.VII](#), has been applied by academic *software journals* very widely.

One can also observe that the application of open source values is in alignment with the spirit of Open Science, which promotes open access to research results, openness, collaboration, and reproducibility and, similarly, the application of open source licenses and publication of source code facilitates free and open access to software resources and thereby allows other researchers to adapt, change, and use such resources for other scientific contexts.

II.V From the 00s to the present day

During the 2000s, significant developments played a crucial role in the advancement and formalisation of the *software papers* paradigm, as will become evident by examining the events discussed in this paragraph. The evolution of the Internet has fostered code sharing, enabling the development of various tools—such as repositories—that have significantly improved access to software and transformed how it is distributed and developed.

The *Budapest Open Access Initiative (BOAI)*, born from a meeting organised by the Open Society Institute and launched in 2002, is one of the first founding documents of the Open Access movement, which outlined two complementary strategies for the diffusion of Open Access: self-archiving, which encourages authors to deposit their works in open archives, and the creation of open access journals, which make content freely available without cost to readers (“BACKGROUND,” N.D.).

Another significant milestone was the establishment of the *Software Sustainability Institute (SSI)* in 2010 in the UK (“ABOUT THE SOFTWARE SUSTAINABILITY INSTITUTE,” N.D.). SSI emphasises its primary mission is to promote sustainability, proper documentation, and the sharing of software, with the goal of making it recognised not simply as a tool but as a scientific product in its own right. The institute's slogan, “*Better Software, Better Research*,” reflects this philosophy, emphasising the fundamental link between quality software and research progress while also pushing for the recognition of the role of software in research, improving software education for the research community, and recognising the role of people who develop research software. Additionally, SSI has established an environment that enables developers of scientific software to interact among themselves, researchers, institutions, and funders and assist in developing guidelines for creating sustainable software (“GUIDES FOR RESEARCHERS,” N.D.). The initiative has highlighted the necessity to make software usable and maintained over the long term, which is essential to enable research to

become more reproducible and develop it. However, the ease of publishing has brought about new challenges related to software persistence; while code can be easily distributed on the internet, ensuring its long-term availability and maintenance remains an open problem.

In the wake of software persistence, long-term availability, and maintenance issues, initiatives such as *Software Heritage* have emerged to ensure the permanent preservation of software (DI COSMO AND ZACCHIROLI, 2023). Inria-funded *Software Heritage* is a project that obtains, preserves, and shares all publicly available source code and thus ensures a global repository that makes the loss or obsolescence of the software impossible. It indexes software from repositories, package managers, and community contributions to construct an essential infrastructure that offers guaranteed access to research software in the long term. This work contributes to the broader goal of enhancing the reproducibility of research and preservation of scientific software as a stable and citable component of the scholarly process.

In 2021, UNESCO adopted the Recommendations on Open Science⁸, marking a milestone as it represents the first time that an international body of political relevance has officially recognised the importance of open science. As stated in the document: “*The UNESCO Recommendation on Open Science provides an internationally agreed definition, as well as a set of shared values and guiding principles for open science. It also identifies a set of actions conducive to a fair and equitable operationalisation of open science for all at the individual, institutional, national, regional and international levels*” (“UNESCO RECOMMENDATION ON OPEN SCIENCE,” N.D.).

Lastly, as already analysed in Chapter I (paragraphs [I.V](#) and [I.VI](#)), these were also the years in which the FAIR, ORE and EOSC principles were developed, as well as the Open Science movement. The European Union continues to actively promote and expand Open Science, supporting its integration into research policies and infrastructures to enhance transparency, accessibility, and collaboration in scientific advancement.

⁸ Recommendations on Open Science is available with this DOI: doi.org/10.54677/MNMH8546

II.V.I The Advent and Diffusion of Software Repositories

One of the key drivers behind the spread of software code publication in academia is the appearance and expansion of open-source repositories dedicated to preserving and disseminating code.

In 2008, the launch of GitHub⁹, which is based on Git, the distributed version control system developed by Linus Torvalds in 2005, made code sharing and versioning much easier due to an easy-to-use interface and social features that promote collaboration among researchers and developers. In quite a limited time, the site has emerged as a significant reference point for code sharing, facilitating open-source projects, versioning over time, and integrating tools for code evaluation. The fast uptake of GitHub within the academic community has prompted several scientific journals to mandate or suggest code publication via this site. To further cement this practice, it has partnered with Zenodo¹⁰, an open access repository built within the European OpenAIRE project and operated by CERN since 2015 (“ABOUT ZENODO,” N.D.). A key aspect of this platform is the assignment of persistent DOIs to each version of the software, an advantage that not only simplifies citation within the research community but also allows proper traceability and systematic identification of the scientific output. Meanwhile, integration with GitHub enables automatic archiving of repository releases and thereby assists in the reproducibility and preservation of experimental results. To accomplish this objective, Zenodo uses a versioning and historicization mechanism that allows for an exact software development timeline and thereby assists in making experiments replicable and enabling strict comparative studies of results. Furthermore, there is the potential for enriching each submission with metadata containing author information, dependencies, bibliographic citations, and licensing, thereby allowing authors to have control over the utilisation and dissemination of their software. Nowadays, the array of solutions for storing research software is large and varied (cf. Sec. [III.V](#)).

II.V.II The Proliferation of Software Journals in the 21st Century: Case Studies and Editorial Goals

As highlighted in Chapter III, the phenomenon of *software journals*, which in the previous century might appear to be an isolated phenomenon, has experienced a significant proliferation after 2000, with the majority of those kinds of journals being created after that year, especially during the decade of 2010. For the aims of a historical reconstruction of the birth and evolution of *software journals* and *software papers*, this

⁹ <https://github.com/>

¹⁰ <https://zenodo.org/>

paragraph considers three notable examples of pure *software journals* that have explicitly declared their editorial objectives in the first editorial issues: *SoftwareX*, *Journal of Open Research Software*, and *Journal of Open Source Software*.

SoftwareX

Founded in 2015, *SoftwareX* adopts a distinctive approach to evaluating research software, emphasising not only technical quality but also the scientific impact of the software in advancing research. The journal's primary evaluative criterion is driven by the question: "*How has this software contributed to the advancement of scientific research?*"

To assess scientific impact, *SoftwareX* considers multiple factors, including the novel findings facilitated by the software, its role in enabling new research directions, its adoption within the academic community, and its contributions to methodological innovation. Notably, the journal does not prioritise the complexity of the software as a criterion for evaluation; both long-term, large-scale software projects and relatively simple tools, such as standalone scripts, may be equally considered if they demonstrably provide significant value to the scientific community.

While scientific impact is paramount, *SoftwareX* also acknowledges the importance of software quality, recognising that different research contexts require different standards. For instance, software designed for controlling laboratory equipment is subject to more stringent reliability and verification requirements compared to software intended for exploratory data analysis. The journal encourages adherence to best practices in software development, including automated testing, comprehensive documentation, and community engagement, while maintaining flexibility to accommodate disciplines where formalised software engineering methodologies have been less commonly applied.

Furthermore, *SoftwareX* aligns with initiatives such as the UK Software Sustainability Institute's "Better Software, Better Research" campaign, underscoring the broader principle that high-quality software contributes to the robustness and reproducibility of scientific research. Articles published in *SoftwareX* enable proper scholarly attribution for software developers by rendering software citable, integrating it into bibliometric frameworks, and ensuring its long-term preservation through dedicated repositories.

Journal of Open Source Software (Joss)

JOSS, established in 2016, is a journal that aims at publishing short *software papers*, which enable developers to present the key functionalities of their software and include all information necessary for getting academic credit without requiring them to craft lengthy scientific publications (KATZ ET AL., 2018). These papers are short and straightforward, merely outlining the software abstraction, mentioning the authors, and including the link to the software repository.

JOSS utilises GitHub as the underlying platform, leveraging its numerous features (including issue tracking for reviews, quick communication among authors, reviewers, and editors, and notification systems) and guaranteeing that authors and reviewers will typically have familiarity with this software. Peer review of JOSS is done entirely in the open on GitHub, with an emphasis on improving the documentation of, rather than the content of, the code, making the software more usable, and contributing to building a community of researchers who are dedicated to enhancing the accessibility and usability of scientific software. The journal is open access, and the copyright for the authors' work remains with them. The software parts of publications are put out under an OSI-approved license, and JOSS is sponsored by NumFOCUS, a nonprofit which advocates for open source scientific computing. JOSS is also designed for research software in general and is not specific to one area of science. To make life easier for authors, the process of publication in JOSS is very fast: preparation and submission can be accomplished in under an hour for well-documented software. For reviewers, JOSS has developed a checklist system that reviews conflicts of interest, code and documentation, and *software paper* quality. It is also linked to other code review systems, like rOpenSci, to further ease the process of publishing software that has already undergone a stringent review process.

The Journal of Open Research Software (JORS)

Joss is a peer-reviewed academic journal on software that was initiated in 2014 as a partnership between the Software Sustainability Institute and Ubiquity Press, functioning as a metajournal (CHUE HONG, N.D.). The aim is to provide a formal and simplified way of publishing software as a research output. Further, the program was initiated with the final aim of enabling researchers to create a sustainable, citable, and open-access repository for their software, thereby promoting proper academic credit while enabling the repurposing of software in the field of scientific research. Software articles featured in JORS are designed as concise descriptions of research software that reflect its nature, the context of its creation, and the potential for reuse. Their emphasis is more on the readability and correctness of metadata than on the novelty of scientific content, making them an effective means of documenting major software versions in conjunction with conventional research articles. The special reuse section in JORS is

another distinguishing characteristic, and it serves to highlight the adaptability of the software in various research settings, thereby facilitating wider usage and increased visibility. As an open access journal, JORS ensures that all documentation of software and metadata are made freely available in line with general research openness and open science standards. It upholds a robust peer review procedure that is largely centred on inquiring about the accessibility of software, verifying the completeness of metadata, and testing documentation readability. In contrast to conventional *software journals* requesting research novelty together with the supporting code, JORS reduces effort barriers in submissions but sets high expectations on the usability and documentation of the software. In doing so, it encourages the appropriate maintenance of research software and authorship and avoids imposing undue burdens on authors.

In parallel with the emergence of such specialised journals, some established scholarly journals have begun incorporating sections specifically devoted to software documentation. A good example of such a trend is the journal *PLoS Computational Biology*, a prominent journal in the area of computational biology, which in 2011 launched the *Software Articles* section, with a format close to that of journals exclusively focused on software. The papers, indeed, concentrate on the description of novel software for biological data analysis, and they should provide extensive details on the availability of the code. Authors are actually required to release the software in an open repository, ideally under an open-source license. The reproducibility of the code is also mandatory and needs to be accompanied by sample data and installation and run instructions. Lastly, technical documentation: writers must include comprehensive information about the software architecture, dependencies and limitations, if any.

Chapter III Software Papers in the Academic Publishing Landscape: Comparative Analysis of Pure and Hybrid Journals

This chapter presents a comparative analysis of academic journals publishing *software papers*. The aim is to explore the characteristics and practices promoted by these journals in relation to the publication of software, identify the differences between *pure* and *hybrid software journals*, evaluating their impact on the diffusion and recognition of software as a research product.

The chapter is organised as follows. The first section presents the key research questions driving the analysis and forming the foundation for structuring the study (cf. [Sec. III.I](#)). The second section describes the methodology used to identify and classify *software journals*, detailing the queries employed for searching in Google and academic databases outlining the process for determining whether a journal is *pure* or *hybrid* (cf. [Sec. III.II](#)). The section culminates in a table listing all identified *software journals* (*pure* and *hybrid*), including publisher, year of establishment, the terminology used for *software papers*, and the total number of publications as of December 2024. The third section examines the annual publication trends of *software papers*, analysing fluctuations in publication rates over time through graphical representations (cf. [Sec. III.III](#)). The fourth section focuses on the subjects covered by selected journals, investigating whether specific disciplines dominate in *software paper* publication (cf. [Sec. III.IV](#)). The fifth section explores the most frequent repositories promoted by *software journals* for storing the research software they publish, distinguishing between general-purpose, institutional, and software repositories, and concludes with an analysis of methodology for ensuring the long-term preservation of software within these repositories (cf. [Sec. III.V](#)). The sixth section analyses various editorial practices related to software documentation, including the terminology used in *software paper* titles to identify recurring trends, the structure of *software paper* templates revealing a core template shared with other academic articles while incorporating elements specific to software, the average length of software papers, and how software updates are managed at the editorial level (cf. [Sec. III.VI](#)). Additionally, the seventh section investigates the most common access models in *software journals*, the average cost of APCs, cost ranges, and how these financial aspects influence journal sustainability (cf. [Sec. III.VII](#)). The concluding section discusses the licensing models applied both to *software papers* and the software itself, emphasising the importance of harmonising these licenses (cf. [Sec. III.VIII](#)).

III.I Research Questions for Analyzing Software Journals and Software Papers

It was necessary to define a series of guiding questions aimed at exploring the different aspects of the Software Journals phenomenon, allowing us to extrapolate the data necessary for the analysis.

The study was initiated by the following overarching question: *Which academic journals publish software papers, and how can they be classified based on their editorial focus (Pure Software Journal and Hybrid Software Journal)?* While this does not constitute a research question in the strictest sense, it provided the fundamental conceptual framework that guided the data collection process and the construction of the sample.

Based on this initial inquiry, the following research questions were formulated:

- RQ1: What is the quantitative evolution of the publication of *software papers* over time? And of *software journals*?
- RQ2: What are the main thematic areas covered by journals that publish *software papers*?
- RQ3: Which approach is promoted for hosting and archiving the software?
- RQ4: What are the editorial characteristics of journals that publish software papers in terms of article designation, template structure, paper length, and software update policies?
- RQ5: What are the access models and APCs chosen by *software journals*? What is their range and their average value?
- RQ6: What licenses are used for the publication of the article? And of the software?

III.II Methodology for Identifying and Classifying Software Journals

The identification of existing scientific journals, i.e. journals publishing *software papers*, was carried out through searches in bibliographic databases and on the Internet and manual scrutiny on publishers' websites. To search on the Internet, a series of queries specifically tailored to find those journals that could be classified as *software journals* were employed.

The queries we used were based on the following strings: "*Software papers*"; "*Software journal*"; "Software engineering research"; "Code sharing in *software papers*"; "Reproducibility in software research"; "Open source *software journals*"; "Software

documentation in academic journals"; "Software development journals"; "Journals publishing code and software tools"; "Software for scientific research"; "Software publishing in academia".

All the search engines were utilised with the aim of identifying *pure software journals* and *hybrid software journals*. In the case of academic databases like **DOAJ**, **Scimago Journal Rank**, **Web of Science**, and **Scopus**, the same searches were applied wherever the search bar was present. Nevertheless, given the large number of available journals, a more targeted search was conducted using filters that allowed for the selection of journals which, while covering a wide range of disciplines (as will be discussed in section [III.IV](#)), included among their topics or keywords a particular focus on technology, computer science, and software. This approach was necessary to refine the search within websites and databases, ensuring a higher probability of identifying journals that published articles that, according to our criteria, could be classified as *software papers*.

Lastly, publisher websites, including *IEEE Xplore*, *ACM Digital Library*, *Springer Link*, *Wiley*, and *Taylor & Francis*, were searched via direct exploration for additional *software journals*. These publishers were selected not only because they are among the major academic publishers but also because, as observed in our previous analysis through the Internet and academic databases, they host several journals that we identified as *software journals*. This led to the assumption that if a publisher already published a *software journal*, it was more likely to host additional ones.

In order to assess whether an academic journal is a *software journal*, one has to examine a series of elements:

- In a *pure software journal*, the **Aims & Scope** section clearly emphasises the close connection with software alongside the main subject of the journal. In *hybrid journals*, which include a broader range of articles, the focus on software is less explicit but must still be evident within the scope of topics covered. If the connection to software is not sufficiently highlighted in the Aims & Scope, it would be grounds for excluding the journal from the category of *software journals*.
- The **Types of Articles** promoted by the journal include *software papers* (no matter the name they use). Those journals that solely accept articles on software artefacts are categorised as *pure software journals*, whereas those accepting many types of articles, including *software papers*, are classified as *hybrid software journals*. Journals publishing research articles having software as additional material are not in scope;
- The journal policy has a specific **Data and Code Sharing policy**, where code sharing and reproducibility of results are mandatory—two fundamental principles for a *software paper*. Even when there isn't a special section on code

sharing, sometimes this information is given anyway in the **Author Guidelines** under article types.

- The **Author Guidelines** present an article structure that encompasses technical information about the software and stresses the need for full documentation, software availability and exploitation;
- The **Peer Review Guidelines** might include criteria specifically conceived to verify and assess features of the software accompanying the paper;

As a result of the analysis, eleven *pure software journals* and twenty-three *hybrid software journals* were identified. Although these figures cannot be considered “certainly inclusive” because data collection was performed in 2024, and there is a chance that nowadays existing data journals were not identified by our query, we are confident that the collected sample is representative of the *software journal* phenomenon. As far as we are aware, no official list of journals supporting *software papers* exists. Therefore, the identification of these journals represents a key outcome of this study.

The following table provides a summary of selected data collected in our investigation of the identified *pure* (Table 1) and *hybrid software journals* (Table 2). Each table describes the journals by reporting their title, publisher, year of the first issue, the article title assigned to *software papers*, the total number of articles published up to December 2024, the license paper and software, the APCs, the type of access, and the first editorial (which in some cases is the journal’s inaugural editorial and in others a later one) or the first announcement indicating the journal’s introduction of *software papers*. The *Software Paper Titles* column indicates how each journal classifies *software papers* within its editorial framework. Hybrid journals generally assign a specific label to these articles (e.g., "Original Software Publication" or "Application Note"). As for *pure software journals*, some have designated a specific title for *software papers*, while others have not adopted a particular denomination, simply publishing *software papers* without a formal categorisation.

Table 2 List of Hybrid Software Journals (as of December 2024)

Journal Title	Publisher	Year of Foundation	Software Paper Titles	Total Publication (as of December 2024)	First Software Paper Editorial or Announcement	License Paper	License Software	APC	Access Type
Journal of Statistical Software ¹¹	American Statistical Association	1996	<i>n.a.</i>	1584	<i>n.a.</i>	CC-BY	GNU	NO	Open Access
Software X ¹²	Elsevier	2015	Original Software Publication	1166	Editorial	CC-BY	Open Source Initiative (OSI)	760 USD	Open Access
Journal of Software for Algebra and Geometry ¹³	Mathematical Sciences Publishers	2009	<i>n.a.</i>	75	<i>n.a.</i>	CC-BY	GNU	NO	Open Access
Journal of Open Source Software ¹⁴	Comm. Service	2016	<i>n.a.</i>	172	(Smith, 2016)	CC-BY	Open Source Initiative (OSI)	NO	Open Access
IPOL Journal (Image Processing)	Supported by various universities	2011	<i>n.a.</i>	244	(Morel, 2010)	CC BY-NC-SA	GNU	NO	Open Access

¹¹ <https://www.jstatsoft.org/index>

¹² <https://www.softxjournal.com/>

¹³ <https://msp.org/jsag/>

¹⁴ <https://joss.theoj.org/>

Online)									
The R Journal	The R Foundations	2009	Packages, Reviews and proposals, Comparisons and benchmarking, Applications, Special issue articles	954	(Carey, 2009)	CC-BY	MIT	<i>n.a.</i>	Open Access
Journal of Open Research Software	Ubiquity Press	2013	Software Metapapers	283	Announcing the Journal of Open Research Software - a software metajournal	CC-BY	Open Source Initiative (OSI)	350 GBP	Open Access
Archive of Numerical Software	Universität Heidelberg	2013	Regular Articles, Short Communications, Library Introductions	24	<i>n.a.</i>	CC-BY	Open Source Initiative (OSI)	NO	Diamond Open Access
Software Impacts	Elsevier	2019	Original Software Publication	413	Editorial: Welcome to Software Impacts	CC-BY/CC-BY-NC-ND	Open Source Initiative (OSI)	480 USD	Open Access
Computer	Elsevier	1969	Computer	10539	(Burke, 1969b)	CC-BY/CC-	Open Source	3680	Gold open

Physics Communications			Programs in Physics (CPiP)			BY-NC/CC BY-NC-ND	Initiative (OSI)	USD	access or Subscription
Journal of Open Source Education	Comm. Service	2018	Computational learning modules, Open-source software	60	<i>n.a.</i>	CC-BY	MIT, Apache, GPL, MPL,	NO	Open Access

Table 2 *List of Hybrid Software Journals (as of December 2024)*

Journal Title	Publisher	Year of Foundation	Software Paper Titles	Total Publication (as of December 2024)	First Software Paper Editorial or Announcement	License Paper	License Software	APC	Access Type
F1000Research	F1000 Research Ltd	2012	Software Tool Articles	438	<i>n.a.</i>	CC-BY	Open Source Initiative (OSI)	1150 USD	Open Access
Open research Europe	F1000 Research Ltd	2021	Software Tool Articles	19	<i>n.a.</i>	CC-BY	Open Source Initiative (OSI)	NO	Open Access
Frontiers in Neuroinformatics	Frontiers Media SA	2007	Technology and Code	48	<i>n.a.</i>	CC-BY	Open Source Initiative (OSI)	3295 USD	Open Access

Journal of Cheminformatics	BMC	2009	Software	730	Editorial	CC-BY/CC0	Open Source Initiative (OSI)	2290 USD	Open Access
The Research Ideas and Outcomes	Pensoft Publishers	2015	Software Description	8	<i>n.a.</i>	CC-BY	<i>n.a.</i>	299 EUR	Open Access (exception accepted)
Journal of Machine Learning Research	Microtome Publishing	2000	Machine Learning Open Source Software Paper	189	<i>n.a.</i>	CC-BY	Open Source Initiative (OSI)	NO	Open Access
Bioinformatics Advances	Oxford Academic	2021	Application Note - software	24	Editorial	CC-BY	Open Source Initiative (OSI)	2620 USD	Open Access
Bioinformatics	Oxford Academic	1985	Application Note - software	6148	Cabios Editorial	CC-BY	Open Source Initiative (OSI)	3798 USD	Fully Open Access
Biodiversity Data Journal	Pensoft Publishers	2013	Software Description	30	Beyond dead trees: integrating the scientific process in the Biodiversity Data Journal	CC-BY/CC0	<i>n.a.</i>	650 EUR	Open Access
Biodiscovery	Pensoft Publishers	2012	Software Description	0	<i>n.a.</i>	CC-BY/CC0	<i>n.a.</i>	199 EUR	Open Access

Biodiversity Information Science and Standards (BISS)	Pensoft Publishers	2018	Software Description	0	<i>n.a.</i>	CC-BY	<i>n.a.</i>	480 / 560 EUR ¹⁵	Open Access
Estuarine Management and Technologies	Pensoft Publishers	2024	Software Description	0	Estuarine Management and Technologies: A way forward for estuarine research	CC-BY	<i>n.a.</i>	NO	Open Access
Food and Ecological Systems Modelling Journal (FESMJ)	Pensoft Publishers	2019	Software Description	0	Open Science meets Food Modelling: Introducing the Food Modelling Journal (FMJ)	CC-BY	<i>n.a.</i>	NO	Open Access
Metabarcoding and Metagenomics (MBMG)	Pensoft Publishers	2017	Software Description	7	Ready for the deluge! Introducing the new Metabarcoding and Metagenomics (MBMG) journal	CC-BY/CC0	<i>n.a.</i>	650 EUR	Open Access
NeoBiota	Pensoft Publishers	2011	Software Description	2	<i>n.a.</i>	CC-BY	<i>n.a.</i>	950 EUR	Open Access
One Ecosystem	Pensoft	2016	Software	2	One Ecosystem:	CC-BY	<i>n.a.</i>	550 EUR	Open

¹⁵ 480 EUR (for TDWG members in good standing) or 560 EUR (for non-members)

	Publishers		Description		Innovation in ecology and sustainability research publishing				Access
Viticulture Data Journal (VDJ)	Pensoft Publishers	2019	Software Description	2	Opening data and research objects in viticulture: The Viticulture Data Journal (VDJ)	CC-BY	<i>n.a.</i>	NO	Open Access
Plos Computational Biology	Public Library of Science	2005	Software	7	The PLOS Computational Biology Software Section	CC-BY	Open Source Initiative (OSI)	2926 USD	Open Access
Journal of Data Science	School of Statistics, Renmin University of China	2003	Computing in Data Science	38	<i>n.a.</i>	CC-BY	Creative Commons	NO	Open Access
Science of computer programming	Elsevier	1981	Software track	78	Call for Software	CC-BY	GNU, Open Source Initiative (OSI), MIT, Apache, BSD, MPL, Common Development and	930 USD	Gold open access or Subscription

							Distribution License, EPL		
Bioinformatics	BMC	2000	Software article	7440	<i>n.a.</i>	CC-BY/CC-BY-NC/CC-BY-NC-ND	<i>n.a.</i>	2090 GBP, 2890 USD, 2390 EUR	Open Access
Ecography	THE NORDIC SOCIETY OIKOS	1978	Software Notes	79	Editorial: launching Software Notes	CC-BY-NC-ND/CC-BY	<i>n.a.</i>	\$3,190 USD / £2,500 GBP / €2,730 EUR	Fully Gold Open Access
Transactions on Mathematical Software (TOMS)	ACM	1975	Algorithm paper	586	Purpose and Scope	ACM copyright/C-C-BY	GNU	NO	Fully Open Access

III.III Software Journals and Software Papers Proliferation and Uptake

The second chapter of this thesis provides a history of the emergence of *software journals* to put this experience into a broader context and examine the co-evolution processes that, although not directly related, have profoundly influenced their life. In this case, we intend to adopt a quantitative approach, exploring the diffusion of *software journals* by focusing on the number of available journals over time and the number of published *software papers*. This quantitative approach enables us to comprehend tendencies and numerical development, hence an additional understanding of the growth and stabilisation of this kind of scholarly publication. Therefore, it provides answers to Research Question N.1 (cf. Sec. [III.I](#)).

III.III.I The birth of Software Journals: a quantitative analysis

Up until the end of the 1990s and the beginning of the 2000s, the advent of journals specialising in the publication of *software papers* or simply supporting article types focusing on software was a limited phenomenon.

Known Pure Software Journals and Known Hybrid Software Journals

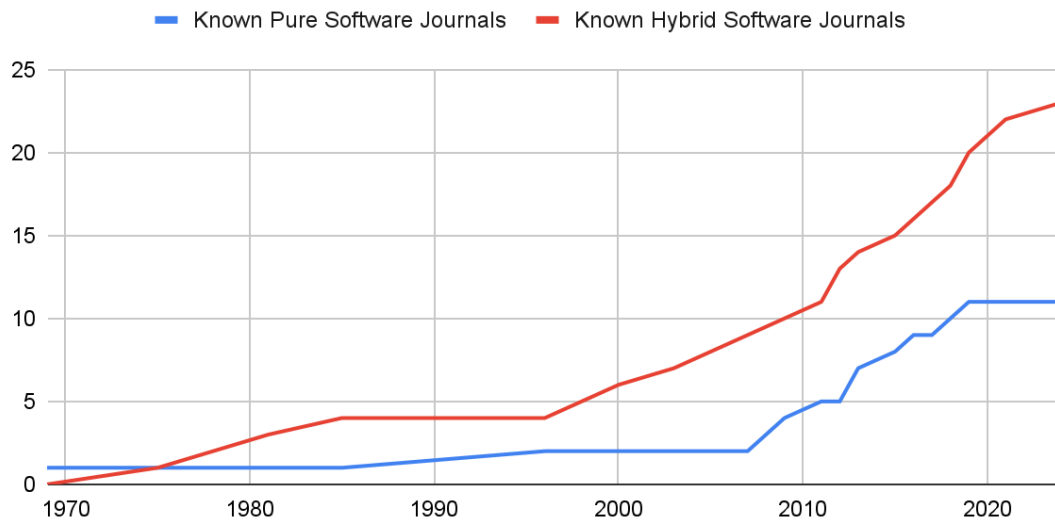


Figure 4. *Number of Pure and Hybrid Software Journals over the years*

As can be observed in Figure 4, the first journals emerged very early, both for hybrids and pure journals: Computer Programs in Physics (CPiP) was established in 1969, and among the hybrids, the ACM Transactions on Mathematical Software (TOMS) was

established in 1975 and *Ecography* in 1978. Other hybrid journals were launched in the 1980s, such as *Science of Computer Programming* in 1981 and *Bioinformatics* in 1985. This shows how, in the initial phases, the phenomenon manifested itself mostly within a broader context, represented by journals not specifically targeting *software papers* only, but publishing them alongside other types of articles. To see a true consolidation of *pure software journals*, one needs to look much further ahead in time. When performing the analysis, one can observe that a turning point for the advent of *pure software journals* occurred in the decade 2009-2019, when an exceptionally notable peak was recorded: nine of the eleven pure journals were created during this period. Similarly, within the scenario of hybrid journals, although their evolution appeared in a more temporal and gradual distribution, the period 2009-2019 represents a time of intense growth. During 1990-2000, eleven new journals publishing *software papers* were established, and in total, nearly half of the hybrid journals that are now available were initiated during these ten years.

The development of *software journals* between the 2009 to 2019 period can be attributed to two critical factors.

The first necessary condition, already explored in the first chapter, was the increasing acknowledgement of research software as a research output. So, during that decade, research software occupied a more and more core position, not only as a means to facilitate academic research, but also as a product of research activity itself, with immediate consequences on its documentation, evaluation, and dissemination. The necessity to ensure good software management, as stated in the FAIR and Open Science guidelines, underlined the necessity of specialised editorial sectors capable of fostering code sharing and reproducibility. Meanwhile, both on the international and European levels, actions in favour of publishing and reusing research software gained diffusion and, therefore, new specialised journals started to emerge.

The other essential element, which has been the subject of a thorough discussion in chapter two, concerns the establishment of infrastructures for software management and sharing (EUROPEAN COMMISSION. DIRECTORATE GENERAL FOR RESEARCH AND INNOVATION., 2020). The advances of platforms such as GitHub, Zenodo, Figshare, and various other disciplinary repositories have made software deposition and citation much simpler, facilitating researchers in publishing code and linking it with a DOI.

III.III.II The development of software paper: a quantitative analysis

The quantitative analysis of *software paper* publications reveals two distinct growth trajectories. While the emergence of *hybrid journals* (Figure 5) has exhibited a more steady and progressive growth, the increase in the number of published *pure software journals* (Figure 6) has followed a more uneven pattern, characterised by periods of slower development followed by more rapid acceleration in recent years.

Figure 5 shows that while *Computer Programs in Physics* (CPIP)¹⁶ marked the inception of *software journals*, the widespread adoption of this publication model occurred gradually. Although other pure and hybrid *software journals* emerged in the following decades, it was not until the 2000s that the phenomenon gained significant traction, with an increasing number of journals explicitly recognising software as a distinct scholarly output. This is because, being journals publishing only *software paper*, these journals have had to wait for the affirmation of this format within the scientific community to gain more visibility and status than the *hybrid software journals* that were already set up for the publishing of other article types beyond the range of *software paper*.

In contrast, as shown in Figure 6, the publication of *software papers* in *hybrid journals* has surged dramatically, with a notable increase since 2004. In that year, the total number of *software papers* published in *hybrid journals* rose from 215 to 444, doubling in only twelve months. The expansion is mainly attributed to the rise in publications in two specific journals: *BMC Bioinformatics*, which in 2004 published 137 *software papers*, and *Bioinformatics*, in which 297 *software papers* were published.

The trend of the *software papers* in the *hybrid software journals* (Figure 6) is characterised by rapid initial growth in the early years of the 2000s, followed by stabilisation and, more recently, by a gentle flexion. Since 2014, publications have been consistent at over 800 per year, with an overall integration of the *software paper* into the literature. A major decrease in 2024 *hybrid journal* publications to 621, from previously reaching a high of 935 in 2017. The drop may be due to saturation of the *hybrid journal* genre, reduced editorial interest in *software papers*, or increased publications in *pure software journals*, which may have drawn authors who were previously oriented towards *hybrid journals*.

However, in contrast, for the past 10 years, journals have consistently had a steady growth rate until the year 2024, when there was an outstanding and dramatic increase, with a rise from 885 to 1491 publications within a single year. The phenomenon was mainly brought about by the spectacular growth of the *Journal of Open Source Software* from only 5 publications in the year 2023 to 550 in the year 2024.

One particular aspect of some interest that emerged during the analysis concerns those journals which, although declaring to support *software papers* as part of their editorial policy, generated few or no submissions in practice. A paradigmatic example is provided by the set of journals published by Pensoft, with the sole exception of *Research Ideas and Outcomes* (RIO). The numbers indicate that the authors publishing in these journals have either not found *software papers* a suitable publication typology

¹⁶The peak of publications in *Computer Programs in Physics* (CPIP) in 1984 is due to a large number of articles published in N.35, which collects index for papers and authors, and summaries for the programs which have been cleaning in volume No. 1-30

or the journals themselves have not made an effort to promote the use of this typology, thereby restricting their exposure and impact.

Lastly, the most significant aspect of this analysis is that, in spite of disparities in the durations and processes of development of *pure and hybrid software journals*, the overall amount of *software papers* generated by the end of 2024 is expected to be close to 15,000 units for both. This balance between the two types of journals is evidence that *software paper* is now an integrated and commonly accepted format, both in specialised and generalist magazines, testifying to its full integration in the environment of scientific research.

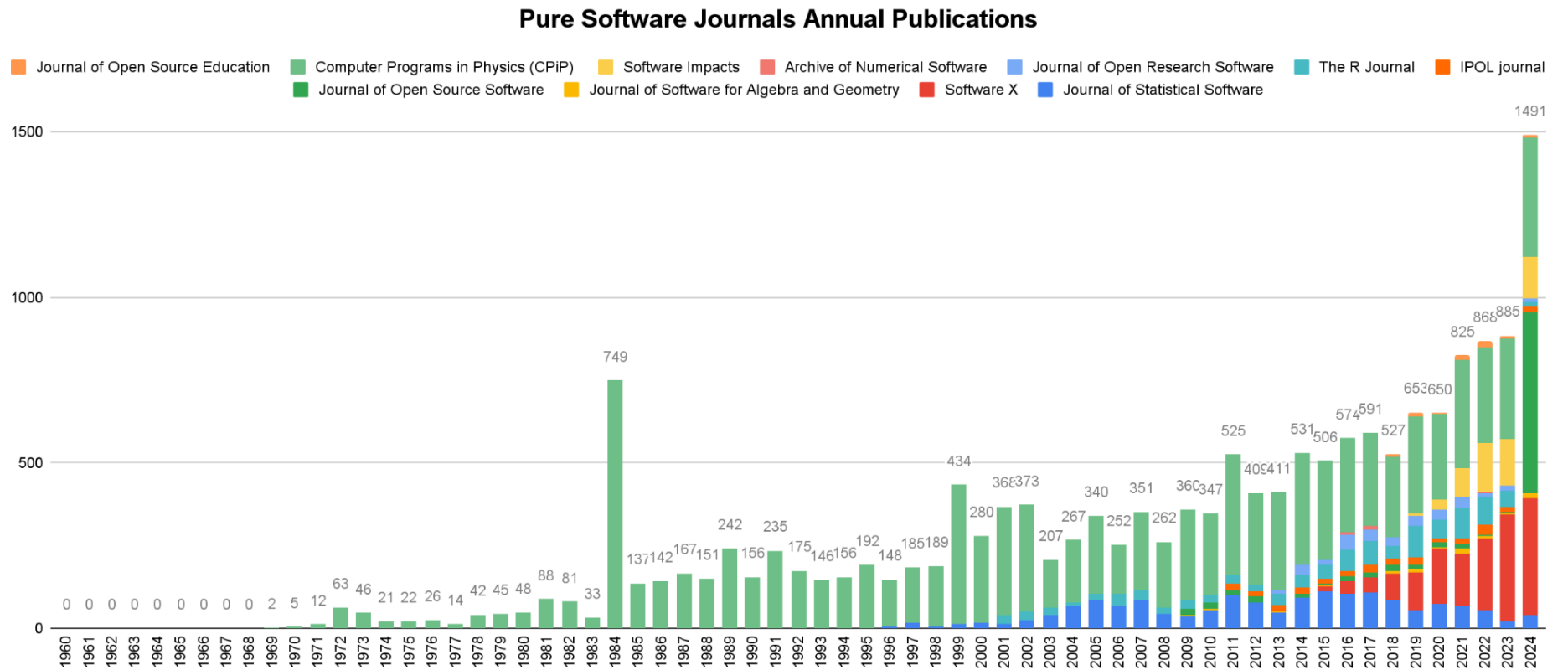


Figure 5. Annual publications of pure software journals

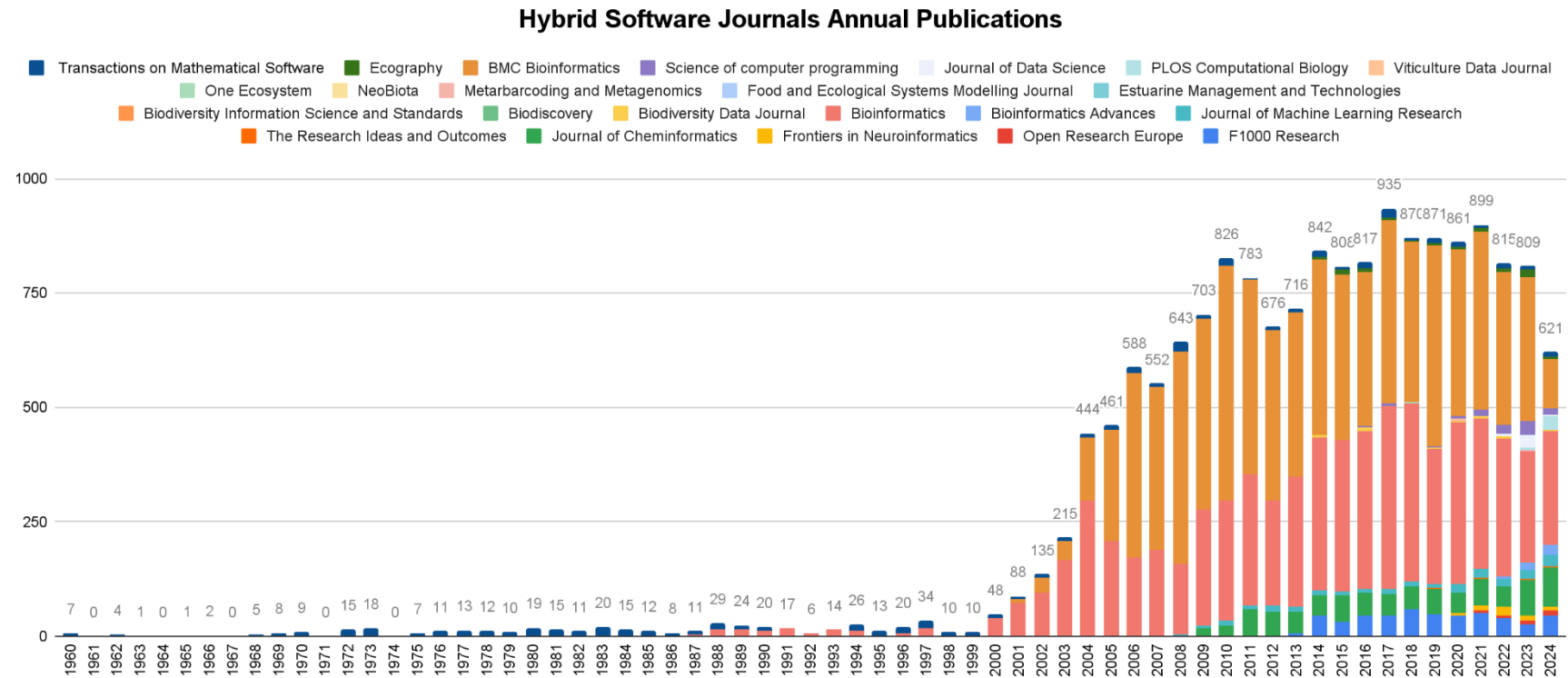


Figure 6. Annual publications of hybrid software journals

III.IV Disciplinary Distribution of Software Journals

The general analysis of *software journals* provides an initial overview of their distribution, but to gain a deeper understanding of the role of *software papers* in scientific research, it is also necessary to examine the disciplinary areas to which they belong, considering both pure and hybrid journals. This approach makes it possible to identify the fields in which the need to document and share software in accordance with scholarly communication practices is considered particularly relevant for academic and scientific research. In particular, this analysis was conducted to address Research Question 2 (cf. Sec. [III.I](#)).

Information on thematic areas was primarily obtained from the Scimago database¹⁷, which offers a detailed classification of journals across multiple sub-disciplines. However, to provide a clearer and more accessible overview, these classifications were aggregated into broader macro-disciplinary areas. For four hybrid *software journals*—*Biodiversity Information Science and Standards (BISS)*, *Estuarine Management and Technologies*, *Food and Ecological Systems Modeling Journal (FESMJ)*, and *Viticulture Data Journal (VDJ)*—disciplinary classifications were assigned based on a qualitative analysis considering the journal’s focus and the types of articles published, as this information was not explicitly stated in the sources consulted. A similar approach was adopted for three *pure software journals*—*Journal of Open Source Education*, *Software Impacts*, and *IPOLE Journal (Image Processing Online)*—whose thematic classifications were determined based on their editorial mission and the content of published articles. In contrast, for some hybrid journals—*The Research Ideas and Outcomes*, *Biodiscovery*, and *Journal of Data Science*—information on thematic areas was obtained from the Directory of Open Access Journals (DOAJ)¹⁸.

¹⁷ <https://www.scimagojr.com/>

¹⁸ <https://doaj.org/>

Pure Software Journal Subject

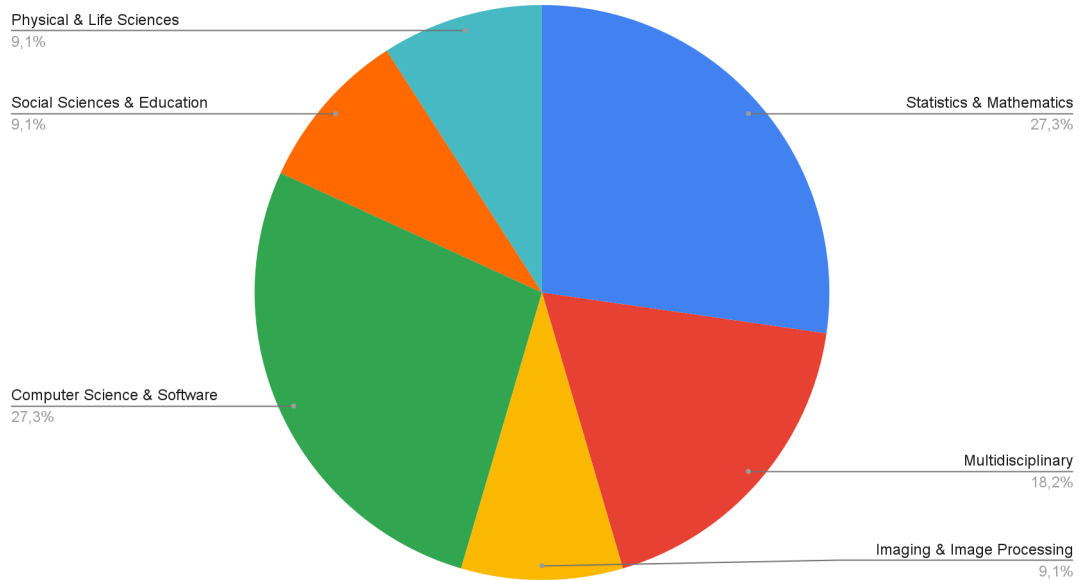


Figure 7. Pure software journal subjects

Hybrid Software Journal Subject

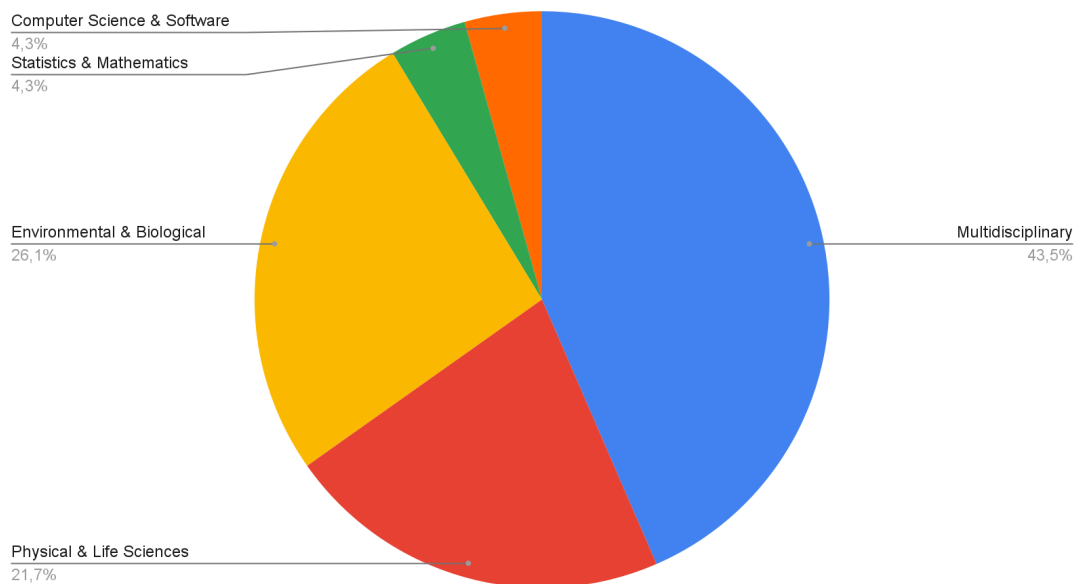


Figure 8. Hybrid software journal subjects

The analysis reveals notable differences between *pure and hybrid software journals* in terms of disciplinary distribution. Among *pure software journals* (Figure 7), the most represented macro-disciplinary areas are **Statistics & Mathematics** (27.3%) and **Computer Science & Software** (27.3%), followed by **Multidisciplinary** (18.2%), **Imaging & Image Processing** (9,1%), **Physical & Life Science** (9,1%) and **Social Science & Education** (9,1%). In contrast, *hybrid software journals* (Figure 8) are predominantly **Multidisciplinary** (43.5%), but also show a significant presence in **Environmental & Biological Sciences** (26.1%) and **Physical & Life Sciences** (21.7%)., followed by **Computer Science & Software** (4,3%) and **Statistic & Mathematics** (4,3%). Additionally, *pure software journals* tend to cover a broader range of disciplines compared to *hybrid journals*.

Software Journal Subject

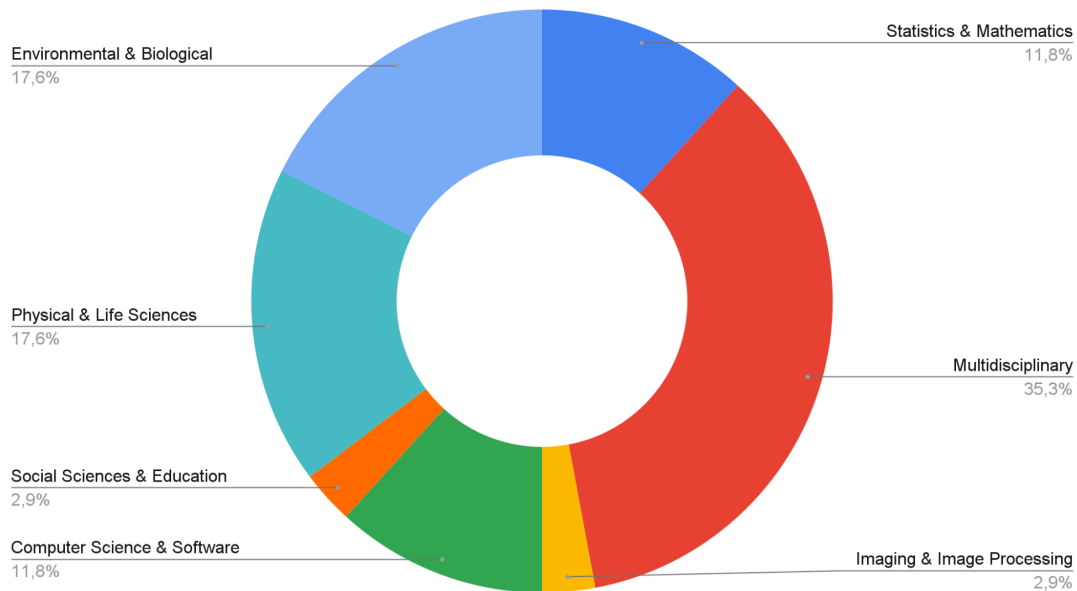


Figure 9. Overall Software Journal Subjects

An overall analysis (Figure 9) reveals that *software journals*, both pure and hybrid, cover a wide range of disciplines, with the most represented category being **Multidisciplinary** (35.3%). This underscores the significant thematic diversity and the broad applicability of these journals across various research areas. The remaining disciplines are also well covered, with **Environmental & Biological Sciences** (17.6%) and **Physical & Life Sciences** (17.6%) being the most prominent, followed by **Statistics & Mathematics** (11.8%) and **Computer Science & Software** (11.8%). Less frequently represented are **Social Sciences & Education** (2.9%) and **Imaging & Image Processing** (2.9%).

Looking at the subjects covered by *software journals*, we can notice that some scientific areas are missing or completely absent; namely, engineering, humanities and arts, and economics and finance do not seem to find a place in these journals. The first explanation is that the journals in these areas, although they meet the criteria of a *software journal*, were not detected in the analysis. Second, in certain domains, software may not be a central component of research, so there are fewer opportunities for the publication of *software papers*. Alternatively, software developed in these domains is typically shared via non-academic sources, i.e., corporate or institutional repositories, diminishing its visibility within scientific literature. There is also the likelihood that *software papers* covering these industries will be published in multidisciplinary journals instead of specialised *software journals*. Software in industries such as engineering and economics is also likely to be developed in corporate or governmental organisations with a production and dissemination reason closer to industrial practice than academic journal publication, thus restricting the coverage in this type of scientific journal. Others are fields lacking a tradition of publishing and divulging software.

The adjective "Multidisciplinary" does not describe any particular scientific discipline, but rather denotes that a journal lacks any clearly defined thematic profile. There may be a number of reasons for this wide classification. One, there are journals that accept software submissions from any discipline without imposing disciplinary restrictions, and as such, they automatically acquire a Multidisciplinary classification. This is despite the fact that the software published can have an unequal distribution across research disciplines. Furthermore, the decision of such a wide classification can be an editorial strategy: being Multidisciplinary, it is possible to attract a wider audience and gain a higher profile for the journal, whereas a narrower one would restrict its focus and decrease submissions. A further consideration is the structural make-up of the bibliometric databases, e.g., Scimago or DOAJ, which do not have enough categories for *software journals*. In some cases, belonging to the Multidisciplinary category can be more the outcome of a pragmatic choice than the reflection of a genuine lack of specialisation. Or, a journal's category may shift over time; some journals which started out with a specific disciplinary orientation may have later broadened their scope, welcoming software from other disciplines and ultimately defining themselves as multidisciplinary. Lastly, the inherent transversal character of scientific software is worth noting as it tends to be shared across transversal aspects. This can be particularly true for software tools related to machine learning, data visualisation, or computational modelling that find application in various domains. Correspondingly, the journals where such studies are published are commonly categorised as Multidisciplinary, although they may well possess a main focus on some areas. Here, the presence of various journals in this category does not necessarily imply that the field is lacking in specialisation; instead, it reflects the flexibility and broad applicability of the program in the field of scientific research.

III.V Software Hosting Approaches

The software artefact is an integral part of a *software paper*; actually, the software artefact is the subject of the software paper. Research question N.3 (cf. Sec. [III.I](#)) aims to study the approaches *software journals* promote to host and make available the software their papers are about. The approaches are divided into three categories (Figure 9): *general-purpose repositories*, *software repositories*, and approaches where the software is not archived in any repository but is directly embedded as an *attachment in the software paper*.

The *general-purpose* repositories (Figure 10) recommended by the journals in this study are platforms such as Figshare¹⁹, Zenodo²⁰, Mendeley Data²¹, and Bioinformatics.org²². As may be observed, most of the journals that promote *general-purpose repositories* mandate the deposition of the software in either an institutional or public repository without any additional limitations.

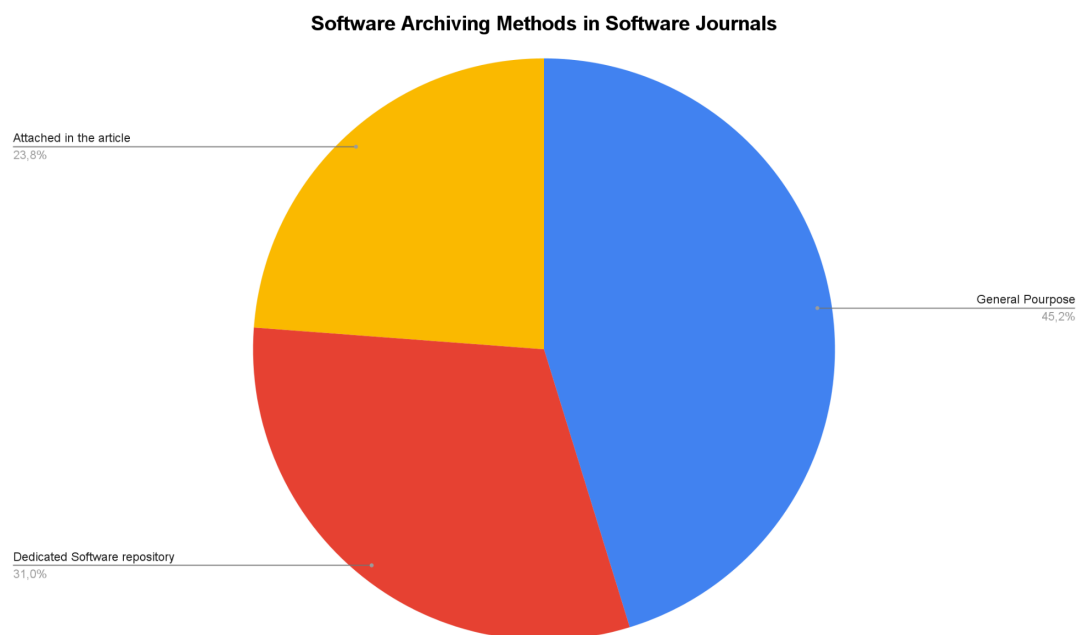


Figure 10. Software archiving methods promoted by Software Journals

¹⁹ <https://info.figshare.com/>

²⁰ <https://zenodo.org/>

²¹ <https://data.mendeley.com/>

²² <https://www.bioinformatics.org/>

Regarding *dedicated-software repositories*, i.e., repositories that exist solely for the purpose of preserving software, many have been cited as examples, e.g., CRAN²³, Bioconductor²⁴, PyPI²⁵, GitHub²⁶, GitLab²⁷ and other Git-based repositories, e.g., Assembla²⁸, Google Code²⁹ (now deprecated), Launchpad³⁰, Savannah³¹, SourceForge³², Open Bioinformatics Foundation (OBF)³³ and Code Ocean³⁴. Some *software journals* explicitly require authors to publish their software in a git-based repository, aligning with a practice widely adopted in open-source software communities. *Dedicated-software repositories* can further be classified into two subcategories: those based on Git and those that are language-specific, such as R and Python, which have more niche applications in the context of software archiving and distribution (Figure 11).

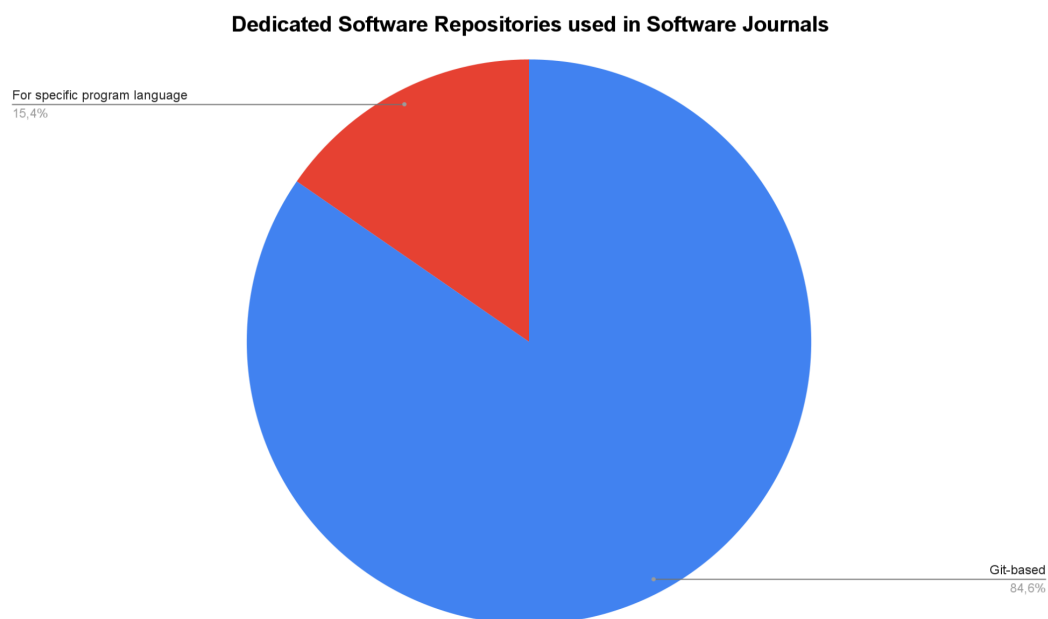


Figure 11. *Dedicated Software Repositories used by Software Journals*

-
- ²³ <https://cran.r-project.org/>
 - ²⁴ <https://www.bioconductor.org/>
 - ²⁵ <https://pypi.org/>
 - ²⁶ <https://github.com/>
 - ²⁷ <https://about.gitlab.com/>
 - ²⁸ <https://get.assembla.com/>
 - ²⁹ <https://code.google.com/>
 - ³⁰ <https://launchpad.net/>
 - ³¹ <https://savannah.gnu.org/>
 - ³² <https://sourceforge.net/>
 - ³³ <https://www.open-bio.org/>
 - ³⁴ <https://codeocean.com/>

It must be noted that the sole *software journal* for which a clear process for the deposition of the software couldn't be identified was Biodiscovery. While the journal officially encourages the publication of software articles, it has not published papers of this kind in practice. Therefore, it is probable that, in the absence of specific demands for the publication of software, concrete guidelines for its sharing and deposit have not been developed yet. The flexibility in repository selection varies significantly across *software journals*. Notably, only a limited number of journals provide authors with a broad range of options for software deposition. *The Journal of Open Research Software (JORS)* offers the most extensive selection, allowing authors to choose from GitHub, Assembla, Figshare, Google Code, Launchpad, Savannah, and Zenodo. The *Journal of Statistical Software* also grants substantial flexibility, supporting deposits in CRAN, Bioconductor, PyPI, GitHub, and GitLab. Similarly, *Research Ideas and Outcomes* accommodates multiple repositories, including GitHub, GitLab, Zenodo, and Bioinformatics. These three journals distinguish themselves by permitting a diverse range of repository choices, whereas most *software journals* either mandate deposition in a specific repository or limit the selection to a narrow set of options.

Among *pure software journals*, there is a fairly equal division in the utilisation of repositories. Specifically, 4 of the 11 *pure software journals* make use of *general-purpose repositories*, 5 use *dedicated software repositories*, and only a couple (*IPOL Journal* and *the R journal*) publish the software directly as an attachment in the paper. *Hybrid software journals*, in contrast, make extensive use of *general-purpose repositories*. This is likely due to the fact that *hybrid journals* publish more diverse types of articles and data and thus prefer repositories that accommodate various types of data. Indeed, 15 out of 23 hybrid journals use *general-purpose repositories*. Hybrid journals promote *dedicated-software repositories* less often, and the number of hybrid journals requiring or allowing the deposition of the software itself within the article is greater than for *pure software journals*.

We need a complete analysis of the repositories, which not only includes their classification by types but also looking at them as a whole. As can be observed in Figure 12, the most widely used repository is GitHub, chosen in 22% of the instances, with Zenodo coming in second at 14.6%, followed by Bioinformatics (12.2%), GitLab (9.8%), and SourceForge (7.3%). Other repositories like Mendeley Data, Savannah, Launchpad, Google Code, Figshare, Assembla, Git-based, PyPI, Bioconductor, CRAN, Code Ocean, and the Open Bioinformatics Foundation are less frequently used. Here again, it should be noted that the scientific journals may recommend the usage of multiple repositories, but not compel authors to use one platform.

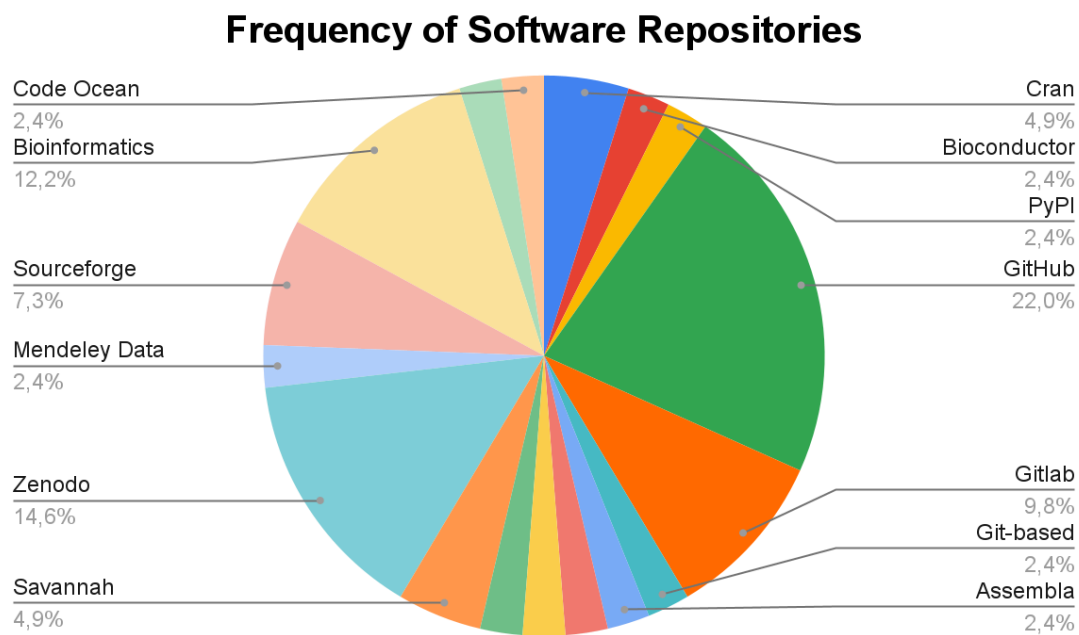


Figure 12. *Frequency of Software Repositories*

In addition to the classification of repositories promoted by *software journals*, it is also essential to consider the issue of long-term preservation of software. The durability of software in repositories is not to be taken for granted; factors in the continuation of accessibility, dependency management, and the ultimate retirement of archiving platforms can play a vital role in the finding and reuse of the code over the long term. Thus, in addition to implementing editorial measures of version tracking, it is important to critically consider the preservation policies that are adopted by the primary repositories where the software is archived. In the case where the software is not

deposited in a suitable repository but instead is directly appended to the paper, long-term access and preservation policies are even more critical. In the absence of a centralised versioning system, such as that offered by repositories, managing updates and later versions of the software can become challenging. It is therefore important that the publisher or journal have clear policies for ensuring that the accompanying software is kept secure and in a manner in which it will remain available in the future. Further, how access rights and licenses are treated must be made quite clear, to avoid conflict between the software license and the paper, and also to facilitate ease of use and redistribution of the software by other researchers. Finally, another important aspect concerns the visibility and citability of the software: without integration with a repository, other scientists may find it less easy to track and adequately cite the software, reducing its scientific impact. Therefore, it is preferable to utilise specialised repositories to provide safe and accessible maintenance of the software, but in case you wish to link it directly to the article, you must be careful about these issues in order to maintain its conservation, updating, transparency on access rights and licensing, and visibility and citability by other researchers.

A very good illustration of the challenges of long-term software preservation in repositories is the deactivation of Google Code. Launched in 2006 as an open-source hosting site, Google Code provided a home for thousands of software projects until 2016, when Google declared its permanent deactivation because of the increasing migration of developers to more widely used alternatives, such as GitHub and Bitbucket (CHUE HONG, 2015). The shutdown of this site caused some issues with software archiving. While Google provided migration and temporary archiving facilities using the Google Code Archive, some of the smaller or orphaned projects vanished or were hard to locate. This made it difficult to recover code or refer back to previous versions of software. Another issue is dependency management, as the closure of the platform caused issues with obtaining particular versions of libraries or tools that were authored exclusively on Google Code. Furthermore, with no backup copies or redundant repositories, developers were struggling to recover and enhance their software assets. Finally, another issue was the sustainability of documentation and issue trackers: besides the source code, documentation, discussion forums, and bug tracking systems were either lost or fragmented across different platforms; this fragmentation made it difficult for researchers and developers to collectively continue their work or fix issues.

The study showed that *software journals* do not provide clear guidelines and formalised processes; most journals do not bother to put in place adequate mechanisms for the long-term handling of software, often leaving this to the discretion of authors or the repositories themselves. This fact highlights how this phenomenon is still an underestimated factor, which calls for more attention to its preservation in the long term.

Among the *software journals* being examined, we notice that Bioinformatics Advances, for instance, asks authors to give guarantees of support and maintenance of the software at least three years beyond publication. PLOS Computational Biology, however, acknowledges the evolutionary nature of software development and strongly encourages authors to deposit and update versions of their source code in an aggregated software version control repository.

Another strategy was followed by the Journal of Machine Learning Research, in which no special provisions for long-term software maintenance are made in the repository but the authors are expected to maintain an active software project web page for its long-term availability.

III.VI Software Documentation Practices

In the following paragraphs, editorial practices concerning software documentation in journals that publish *software papers* are discussed. The purpose is to respond to Research Question Number 4 (cf. Sec. [III.I](#)), which addresses the editorial characteristics of these journals in terms of the naming of article types, template format, article length and software update policy. These are required to comprehend how the *software journals* handle the documentation part of the software publishing and to establish the potential conventions and guidelines applied.

III.VI.I Analysis of Software Paper Naming

When researching journals that publish software articles, an interesting consideration is the name given to this type of article. Knowing whether there are recurring patterns in the names used to refer to this typology of publication can help determine the occurrence of repeated terms or substantial differences in *software journals* and between *pure* and *hybrid journals*.

Software Paper Title in Pure Software Journals

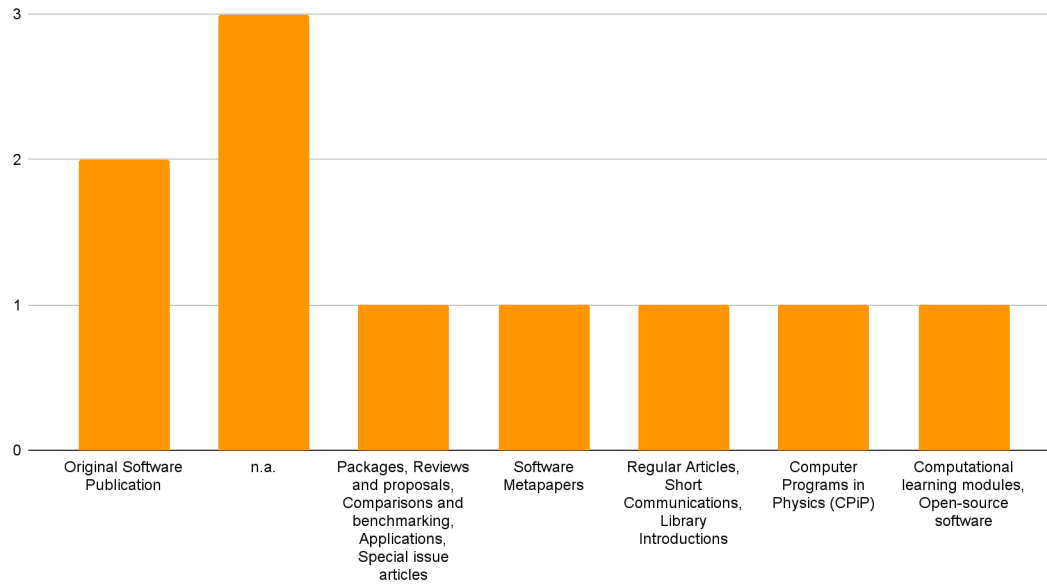


Figure 13. Article type names for software paper in Pure Software Journals

As shown in Figure 13, in the context of *pure software journals*, it has been observed that there is no consistently followed naming convention for *software papers*. Each journal, with the exception of *SoftwareX* and *Software Impacts* (both of which are published by the same publisher, Elsevier), has assigned a different denomination to the *software papers*. In addition, there are *software journals* that have not assigned a name to this type of article; there are several reasons for this phenomenon: the lack of a standard convention, since unlike other types of academic publications, such as review articles or research papers, articles describing software do not yet have a universally accepted denomination since they are a rather recent phenomenon, or another possible explanation could be that journals, lacking rigorously defined editorial standards, may not maintain a uniform classification of contributions, placing greater emphasis on the content of the article rather than its formal designation.

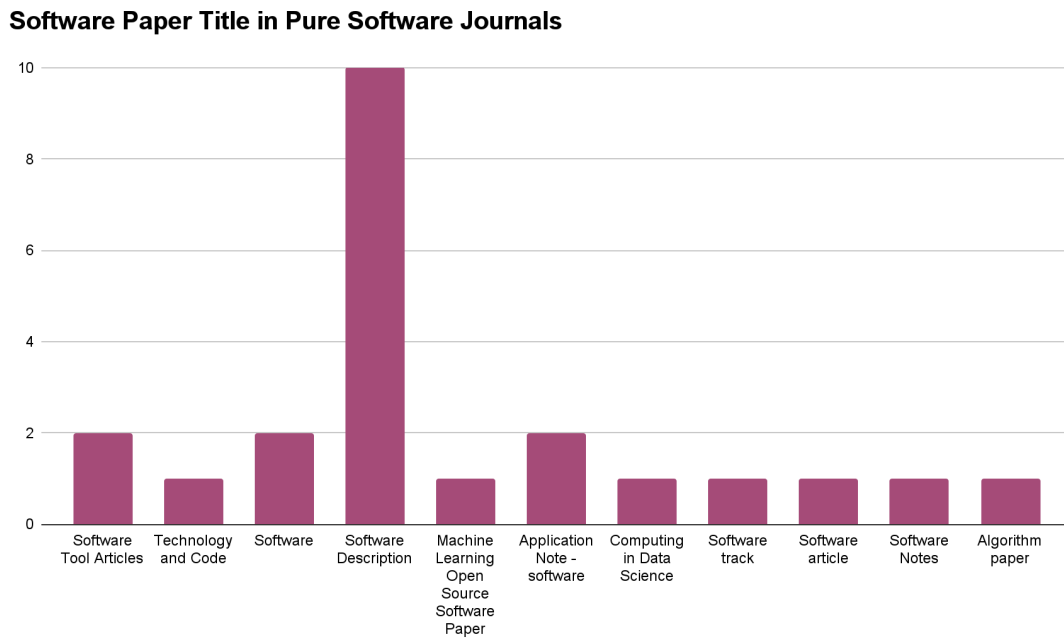


Figure 14. *Article type names for software paper in Hybrid Software Journals*

Figure 14 indicates the spread of the most frequent names in *hybrid software journals*, highlighting the prevailing terminological tendencies adopted.

In *hybrid software journals*, the terminology employed is characterised by a higher degree of standardisation. More specifically, standard titles such as *Software Description*, *Software Tool Article*, and *Application Note* appear across various journals. Notably, this standardisation appears to be influenced by the fact that many of these journals belong to the same publishers. For instance, the term *Software Description*, which is the most frequently used, is predominantly found in journals published by Pensoft Publisher. Contrary to *pure software journals*, in hybrid journals, there are no cases where *software papers* have not been assigned a specific title, this is likely due to the need to clearly distinguish these publications from other types of articles within the journal.

III.VI.II Software Paper Template

The analysis of the templates promoted by the journals for producing software papers enables the identification of two classes of information: standard information and software-specific information. The standard information contains, most significantly, the essential elements of a typical academic work—i.e., the title, the authors, the abstract, the body of the article, and the bibliography—that constitute the skeleton of any academic work. To these are added the synthesis and content sections, including the introduction, objectives definition, and main results presentation, as well as the ethical and regulatory aspects that comprise declarations of conflict of interest and adherence to the guidelines. Meanwhile, the templates offer a range of software-specific elements contributing to implement the body of the article that are necessary for complete and workable description of the software. These include:

- **Software technical information:** A concise explanation of the primary technical details related to the software. The software is described in detail, encompassing its functionality, architecture, and usage;
- **Software license:** Statement of the license under which the use, distribution and modification of the software are regulated;
- **Software availability:** Information helping to obtain the software artifact, usually including a link to the repository hosting the software;
- **Project home page:** a link to the web page for the software project, where more information and updates can be found;
- **Practical Examples and Applications:** An illustration of use cases and applications that demonstrate the effectiveness and versatility of the software;
- **Project Goals and Effects:** Explanation of the goals achieved, the requirements fulfilled and the impact generated by the project, highlighting the applicability and principal outcomes;
- **Future Directions, Updates, and Reuse:** This outlines the anticipated updates, the possibilities for reuse, and the possible plans for the evolution of the software.

Examination of *software journals* indicates that, even though hybrid journals tend to have more formal templates and more detailed sections dedicated to software compared to *pure software journals*, there are also *pure software journals* with a high level of specialisation in their organisational structure. One might presume that *pure software journals* would employ extremely defined and specialist templates, particularly as they report about software only. However, as observed through this analysis, hybrid journals create more complex templates. This is most likely because *hybrid journals* must format *software papers* in a way that distinctly separates them from other types of publications under coverage. In particular, the *Journal of Open Source Software* and the *Journal of Open Research Software* excel in their mature models that emphasise the

most important aspects of the software, ranging from technical documentation to reproducibility. In the case of hybrid journals, the most advanced is the *Science of Computer Programming*, which takes an exhaustive approach to software description and evaluation provided in the published papers.

III.VI.III The Length of Software Papers

Software papers can be classified into two primary macro-categories: one characterised by a preference for concise and synthetic contributions, and the other by a focus on more comprehensive and analytical works

On the one hand, there are articles characterised by a synthetic form and conciseness, usually between 250 and 1000 words or limited to a few pages. This type is represented by the *Journal of Open Source Software (JOSS)*, which allows submissions between 250 and 1000 words, and also by *Software Impacts*, which imposes a limit of three pages for the main body of the article. A similar model is applied to the Application Notes of *Bioinformatics Advances*, which have a maximum length of four pages, and to the *Software Original Software Publications* in *Software Impacts*, which are to bring up to date the status of already described software and should not be more than three pages in length. In others, such as the *Journal of Open Research Software (JORS)*, *Software Metapapers* do not impose a page limit. Instead, they are intended to offer a brief overview of the software, highlighting the availability of the code and how it can be accessed, and not into detailed descriptions of its structure or experimental validation. Especially, *JORS* includes a typology of "short paper," with an emphasis on a brief presentation of the research software and how it is available, without discussing scientific or methodological details.

On the other hand, there is a second type of *software paper* that is near the standard length of traditional research articles, thus offering a more in-depth analysis of the development of the software and its uses. Journals like *Frontiers in Neuroinformatics* set an upper limit of 12,000 words and require that a maximum of 15 figures or images can be used. *The R Journal* is more flexible by accepting submissions with a maximum of 20 pages, whereas the *Journal of Machine Learning Research (JMLR)* accepts submissions with a maximum of 50 pages with the caveat that an explanation for particularly long submissions should be included in the cover letter. Similarly, *One Ecosystem* imposes a limit of 40,000 words, including spaces, on the text body. In contrast, the *Journal of Statistical Software (JSS)*, like other journals, imposes no limits on length, thereby permitting very comprehensive articles.

Overall, there is a trend whereby *software papers* published in pure journals are more likely to take a more synthetic approach with a focus on the basic description of the

software. Hybrid journals, however, enable longer and fuller contributions where the development of the software is placed in the context of a broader discussion.

It is finally noted that *software papers* are generally shorter compared to conventional research papers. This is because, unlike conventional articles which are focused on theoretical, methodological, and experimental discussions, *software papers* are more concerned with describing the software and its usability.

III.VI.IV Software updates and software papers

In *Software Journals*, the management of software updates follows different strategies depending on the journal: in a few instances, there is no established process for updating the software described in the *software paper*; instead, in other journals, more structured systems of managing new releases of the software previously described by published *software papers* have been implemented. In the first case, any new software versions can only be reported indirectly, for example, through subsequent articles or the documentation provided in the repositories where the code is archived. In contrast, in the second case, software update articles are provided, that is, articles specifically dedicated to describing the advances and changes introduced in subsequent versions of the initially published software, avoiding the need for new complete publications.

The capacity to track updates in a systematic manner is of particular importance in the field of scientific research because the software utilised in research studies is not fixed; rather, it is continually being enhanced with the aim of enhancing the reproducibility of results, conforming to new procedures or data, and addressing any flaws capable of undermining the validity of the research studies. The absence of a uniform process of the updating or documentation of every version of a particular software by a particular *software paper* presents a significant challenge in effectively citing the version used; in the event of no proper documentation and proper citation of every version, researchers risk citing versions not representative of those used, and hence compromise on the transparency and reproducibility of research. The availability of editorial styles for updates therefore assists in ensuring transparency throughout the software life cycle, with every enhancement being recorded in detail and readily available.

Accordingly, most journals have adopted a number of solutions for updates management. For instance, Software X permits the publication of update papers, referred to as *Software Updates*, that are up to two pages long and describe the enhancements that have been accomplished. Similarly, Computer Physics Communications employs the *New Version Announcements* (NVA) format, characterised by brief communications exempt from peer review, with the objective of disseminating new versions of the programs in the CPC Program Library at high

velocity. The *Archive of Numerical Software* employs *brief communications* to announce *software updates* to code previously published in routine issues, whereas *Software Impact* provides space for more comprehensive *Software Follow-Up* papers. The second alternative is offered by F1000Research and Open Research Europe, which both employ an article versioning approach. In these journals, a revision can be published with the tag *[Revised]* if it incorporates the modifications suggested by the reviewers, or as *[Update]* for small additions published after indexing.

III.VII Article Processing Charges (APC) and Access Models in Software Journals

Article Processing Charges (APCs) are a component of the landscape of scientific publishing since they have a significant influence on both the availability and the financial sustainability of academic software publications addressing Research Question N. 5 (cf. Sec. [III.I](#)). The charges also have a close relationship with the type of access embraced by journals. A comparison of these two phenomena enables us to learn about the distribution of journals that charge or do not charge APCs, to recognise those that charge the most and assess how these fees are incorporated into journal publishing models, specifically with regard to open access.

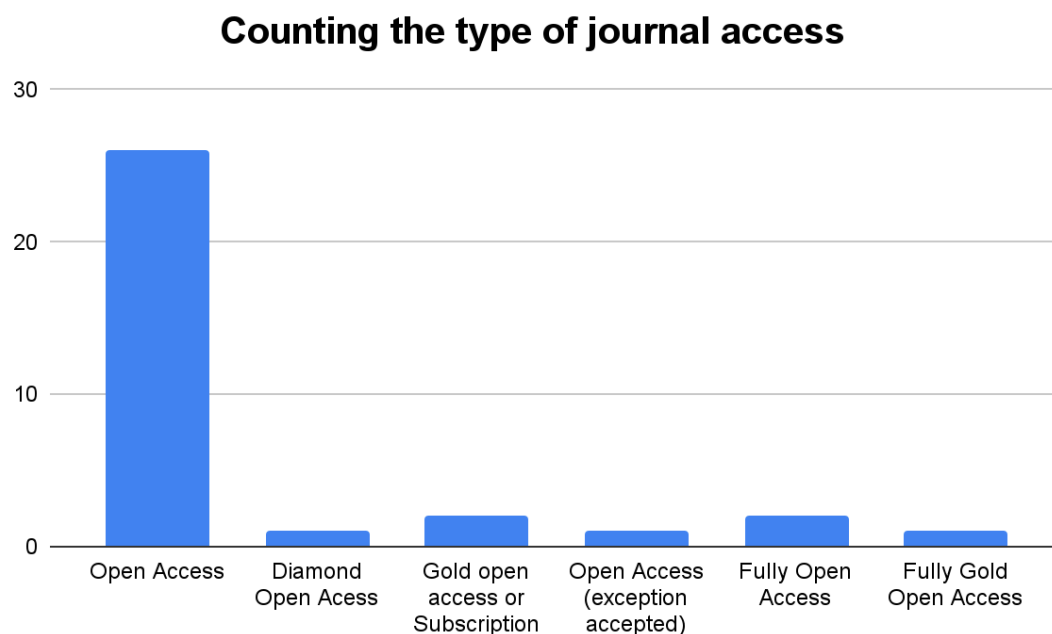


Figure 15. Types of access in software journals

As shown in Figure 15, the analysis conducted demonstrates a clear divergence of access policies in *software journals*. Our analysis reveals that twenty-six journals are Open Access, one is Diamond Open Access, two offer a choice between Gold Open Access and Subscription, one accepts Open Access but allows other access types as well, two are fully Open Access, and one is fully Gold Open Access.

Gold Open Access is the one where articles are open-access to readers without any fee; however, publication costs are transferred to authors in the form of APCs, thus imposing a burden on authors with insufficient funds. This approach is followed by journals such as Computer Physics Communications, where it has a cost of 3,680 USD, Science of Computer Programming, which has a requirement of a payment of 930 USD, and Ecography, which has a cost of 3,190 USD.

Diamond Open Access (FUCHS AND SANDOVAL, 2013) is a variant in which authors and readers are not required to pay direct fees because running costs are supported by institutions, consortia, or public funding. Diamond Open Access projects constitute a minority in our software journal sample; one such example is the Archive of Numerical Software which is hosted by the Heidelberg University Library.

The majority of *software journals* (pure and hybrid) are Open Access, providing free access to all of their published articles.

The hybrid model supports both Open Access articles and subscription-access articles, where authors can choose to pay an APC to publish their article so that it will be made freely available, while other papers can be accessed by subscribers only. Nevertheless, this model carries the risk of "*double dipping*" phenomena, where journals get both a fee of an Article Processing Charge (APC) for the Open Access article and another payment via journal subscriptions from libraries ("GOLD OA E OA IBRIDO," N.D.).

Another area of interest is the typical cost involved in the publication of an article within *software journals*. The analysis of charges shows considerable variability, ranging from 199 USD (*Biodiscovery*, Pensoft) to 3798 USD (*Bioinformatics*, Oxford), with an average expenditure of 1039 USD.

The other notable aspect concerns the presence of journals with and without Article Processing Charges (APC). The evidence collected indicates that, among the pure *software journals*, six do not have an APC, and four have a publication fee. For one journal, it was not feasible to collect information on the APC. For hybrid journals, seven have an APC, and sixteen do not have any direct charge to authors. It may be observed, therefore, that APCs are usually found in hybrid journals, and *pure software journals* do not tend to charge APCs or, if at all, at a reduced level.

Conversely, a paper published in the Journal of Open Source Software (JOSS) gives a comprehensive analysis of the expenses of operating an open-access academic *software journal* (KATZ ET AL., 2019). In practice, they calculate the real cost the “publisher” must sustain for a journal given by the Annual Crossref membership (\$275/year), the cost to have DOIs (\$1/accepted paper), the hosting of the website hosting (\$19/month) and domain name registration (\$10/year). Publishing 300 papers per year the overall cost is just \$813 in total, or \$2.71 per paper. Nonetheless, the Journal of Open Source Software (JOSS) receives extensive voluntary contributions from editors, reviewers, and developers; otherwise, the costs associated would be much greater. The research also points out that, should the journal be considered a for-profit entity, the charging of an Article Processing Charge (APC) would be unavoidable to account for the other expenses, including the maintenance of software infrastructure and financial services. It is estimated that this would translate into a cost per article of approximately 140 USD while keeping the profit margin relatively low. Besides, it indicates how an open-access journal is able to run at relatively low expenses without limiting the availability of articles, pending the introduction of a sustainability model founded on donations and voluntary payments.

III.VIII Software Paper and Software License

To discuss licensing in the context of *software papers* it is worth focusing on two distinct aspects: the license applied to the academic article and the license with which the software is distributed. This section, addressing Research Question N. 6 (cf. Sec. [III.I](#)), examines how license choice influences the diffusion, reproducibility, and transparency of scientific research, particularly in publishing software and in accordance with the principles of Open Science.

From the viewpoint of license distribution with respect to *software papers*, it is noted that most of the journals studied employ a CC-BY (Creative Commons Attribution) license, thus following the principles of open access. It permits individuals to copy, distribute, adapt, and use the work for commercial purposes on the condition that the paternity of the original author is acknowledged. This format promotes the maximum diffusion and adaptability of the contents (MARGONI AND PETERS, 2016).

However, many journals offer authors the option to choose between the CC-BY license and one of several more restrictive licenses, for example, CC-BY-NC (which restricts commercial use), CC-BY-NC-ND (which bans both commercial use and derivative works creation), or CC0 (which renounces all copyright, making the work available to the public free of any legal limitations on its usage, and is the most open of all licenses).

The only exception found is the ACM Transactions on Mathematical Software, which offers a specific ACM license, distinct from the Creative Commons licenses, among its available options.

It is possible to observe the distribution of *software paper* licenses in the following graphs, first in the case of *pure journals* (Figure 16) and then in the case of *hybrid journals* (Figure 17).

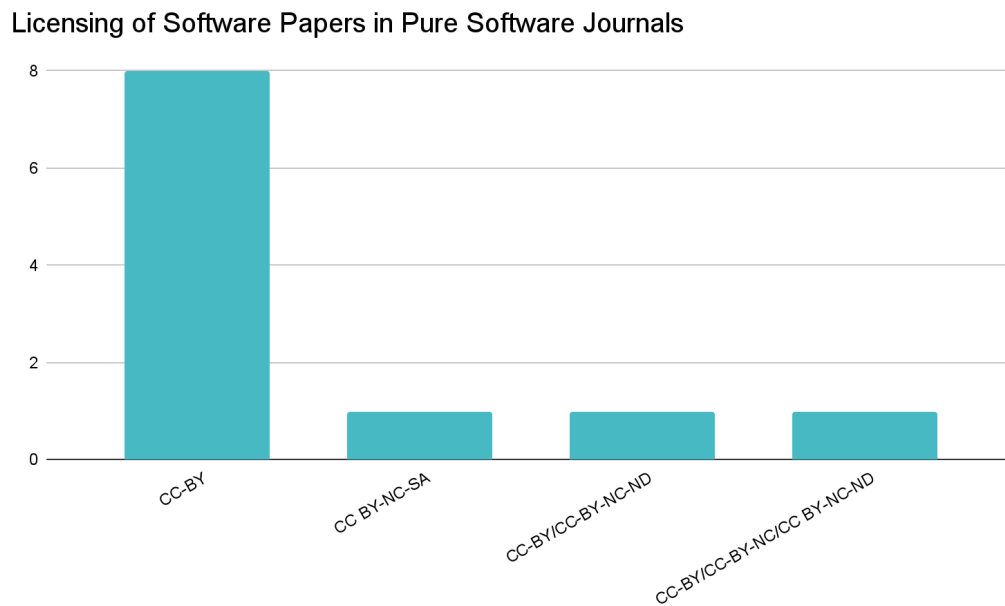


Figure 16. Article licences by Pure Software Journals

In Figure 16, it is observed that the most prevalent license is CC-BY, which is used by 8 journals. Additionally, one journal utilises the CC-BY-NC-SA license, another provides a choice between CC-BY and CC-BY-NC-ND, and a further journal offers a selection between CC-BY, CC-BY-NC, and CC-BY-NC-ND.

Licensing of Software Papers in Hybrid Software Journals

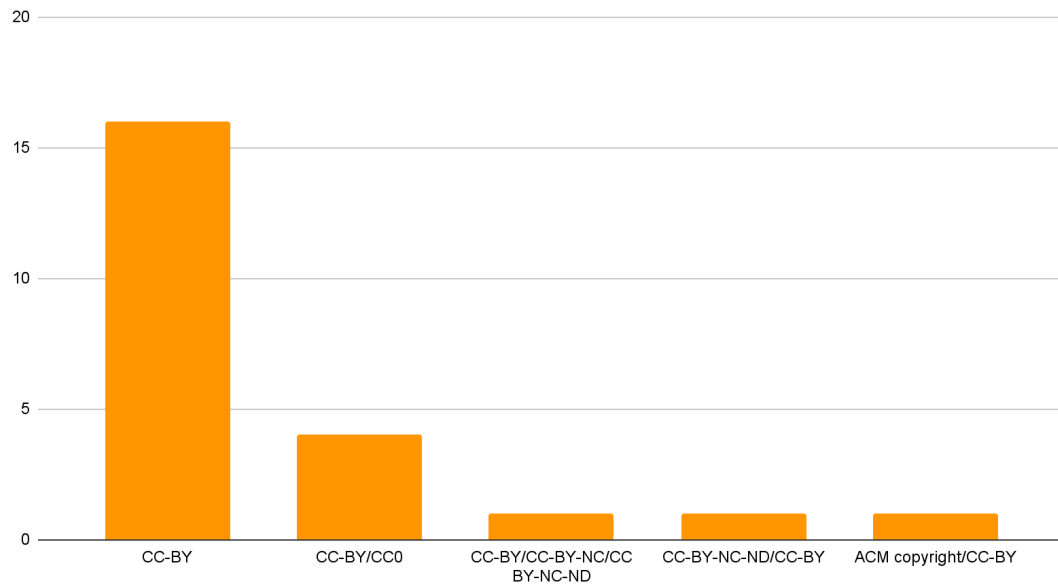


Figure 17. Article licences by Hybrid Software Journals

In Figure 17, the majority of journals, specifically 16, adopt the CC-BY license. Four journals provide a choice between CC-BY and CC0, one journal offers a selection between CC-BY, CC-BY-NC, and CC-BY-NC-ND, another allows the choice between CC-BY-NC-ND and CC-BY, and finally, one journal provides a choice between ACM copyright or CC-BY.

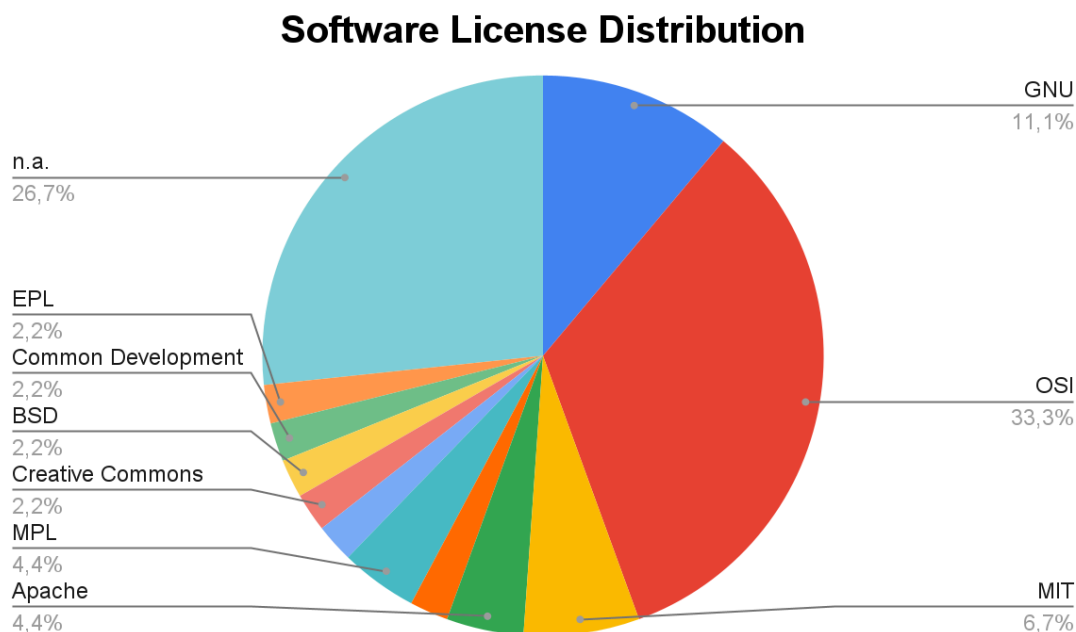


Figure 18. *Software licences by Software Journals*

Figure 18 illustrates that the majority of *pure* and *hybrid journals* (33.3%) demand the use of a software license OSI (Open Source Initiative) conformant. The most used licenses, following OSI, are the GNU (11.1%), the MIT (6.7%), the MPL (4.4%) and the Apache (4.4%). The other licenses that have been used are the EPL, the Common Development and Distribution License, the BSD, the Creative Commons licenses, CeCILL, and the GPL, each with 2.2%. It must be noted that, in a few journals, the choice of various licenses is offered with the possibility that authors might not be restricted by one. One other point is that a significant portion of journals (26.7%) have not stated the use of a software license, leaving it to the choice of the author, e.g. this is the case of *Pensoft* and *BMC Bioinformatics* journals.

Chapter IV Conclusion, Prospects and Future Directions

The path of this research has highlighted the growing visibility of research software publication in academic publishing cultures; however, it still faces many structural and cultural barriers. As such, research software has transformed itself from being only a supportive tool for data analysis to being an actual research product, which is now needed in many different disciplines. At present, in contrast to conventional scientific journals and academic papers, software artefacts have not yet advanced to the point of becoming a core factor in assessing researchers; therefore, their academic legitimacy is debatable, particularly because there are no established and universally accepted standards for documentation, citation, code management, evaluation, and impact assessment. This issue is further exacerbated by the fact that current research assessment methods still rely heavily on publication-based metrics such as citation counts, often failing to acknowledge a wider range of research contributions. There are initiatives aiming at overcoming this issue, namely the Coalition for Advancing Research Assessment (CoARA), a collective of over 700 research organisations, funders, and evaluation authorities committed to reforming the way research, researchers, and research organisations are evaluated (“ABOUT,” N.D.). Through the Accord on Reforming Research Assessment, published in July 2022, CoARA advocates for moving beyond traditional publication-based metrics to more inclusive evaluation criteria that recognise diverse research outputs, including software artefacts, thus filling a significant gap in current evaluation practices. In spite of this, through the years, various movements have emerged with the purpose of promoting the sharing and valorisation of research software, thereby advocating for the adoption of standards that ensure greater accessibility, reusability, collaboration, and credit attribution. In this sense, the FAIR principles, initially developed for scientific data and later broadened to software, are a fundamental aspect of ensuring that code is readily accessible, usable, and reusable by the scientific community. The FAIR principles are part of a broader Open Science context, although it is important to clarify that adopting the FAIR principles does not necessarily imply adopting open access practices. Open Science aims to promote the sharing of data, software, and methods, fostering a more collaborative and transparent approach to science, where collaboration and access to resources become key tools for scientific progress.

Simultaneously, this research has tried to reconstruct, almost like in a chronicle, the evolution of the editorial context of research software publication by software journals. Code has traditionally been handled over the years as a second-class citizen of research, printed in paper appendices to articles or circulated informally between researchers. The development of *software papers*, however, has enabled greater formalisation of software publication, highlighting that research software artefacts

are worth being “published” on their own rather than being published as a side effect of the publication of a research paper presenting research results obtained by a research software. This development is most directly the result of innovative *software journals*, which have provided an environment in which code is increasingly being treated as a valuable scientific product with an editorial identity of its own. One should also mention the significant role of *hybrid software journals* that have included *software papers* within traditional editorial frameworks, thereby recognising the role research software has on par with research results described in their research papers. Moreover, one can recognise that, during the second part of the twentieth century and early 2000s, a variety of events and advancements, while not directly related to software formalisation as a scientific product, played an important role in advancing the development and dissemination of *software papers*. This enabled the creation of a context conducive to a critical change in the perception and handling of scientific software, in which its formalisation is increasingly encouraged as part of research. Among these, one can remember, for example, the development of repositories like GitHub, Zenodo, and SourceForge that are critical for creating an ecosystem of sharing and collaboration where research software moves beyond the status of being a standalone unit and becomes part of an even more significant scientific and collaborative pipeline. There has also been remarkable progress in developing open-source licenses like GNU that have enabled a more unrestricted yet more secure dissemination of software with the guarantee that modifications and upgrades are fed back into the scientific community, creating a cycle of improvement and innovation. Furthermore, various initiatives have been established, such as Software Heritage (DI COSMO AND ZACCHIROLI, 2023), which gathers and provides access to the source code of all existing software, independent of their purpose or license. This initiative contributes to the creation of an integral archive of all the software versions, enabling consulting for research, development, or preservation purposes. In addition, initiatives such as the Sustainable Software Institute (SSI) aim to improve the sustainability of scientific software, tackling this challenge from both a technical and an organisational, managerial, and ethical perspective.

The third chapter of the research presents a technical analysis of the publication environment of *software papers* in the academic setting, comparing journals that exclusively publish articles dedicated to research software, referred to as *pure software journals*, with those that feature such articles alongside conventional publications, referred to as *hybrid software journals*. A series of research questions characterised this part of the project to discover the distinguishing characteristics of these two types of journals and the processes they implement to promote research software publishing. Aside from this, it also examines the basis for differentiating *hybrid software journals* from *pure software journals*. The study then shifted to other domains pertinent to more precisely delineate the phenomenon of *software papers*. In particular, we analysed: the chronological evolution in quantitative terms, considering both the inception of *software journals* and the publication frequency of

software papers per year; the disciplines covered by the publications; the repositories promoted to store the software; the editorial choices, including the nomenclature adopted by journals for the typology of articles dedicated to *software papers*, the size, the information about software expected to be reported in the paper narrative, the cost of publishing these articles and guarantee open access, the access modalities for software and the accompanying paper; and, lastly, the licenses adopted for the article and the software.

Towards the end of this chapter, one needs to question if the phenomenon of software journals has an impact on Open Science and whether it is helping to promote open sharing and reuse of code, fostering the value of research software. From the analysis conducted, it transpires that software journals are a key element in Open Science dissemination because they offer an editorial infrastructure enabling one to record, archive and distribute scientific software in a transparent manner. In particular, their presence encourages the academic community to release code under open licenses, to document it in detail, to make results reproducible, and to make the reuse of software by other researchers possible—both prerequisites for a more verifiable and usable scientific ecosystem.

Despite this, the *pure* and *hybrid software journals* are not free from criticisms such as the absence of editorial standards, issues with citing software, or Article Processing Charges and equal publication opportunities.

One of the primary hindrances is the absence of generally accepted editorial criteria for *software papers*. In contrast to conventional academic articles, which adhere to well-known standards of structure, citation, and evaluation, *software papers* have yet to adhere to consistent standards. Variability manifests in several areas of publication, for instance, in the naming convention of software articles, which differs from journal to journal, the templates utilised, the access methods utilised, the licenses utilised, and the repositories utilised. The differences in editorial conventions employed within software articles significantly affect the usability and predictability of information to readers, who tend to get confused regarding the nature of the content they are about to read. At times, the article might be highly technical in nature, with similarities to a software guidebook, whereas at other times, it might take a theoretical approach. Simultaneously, this inconsistency can also create issues in terms of traceability, as the software can be kept in multiple repositories or embedded in the article itself. A further issue is the delivery of updates: in the absence of explicit editorial guidelines, the reader is frequently left wondering whether the software under discussion is a stable release, whether more recent releases are available that include bug fixes or new functionality, or whether the code has been rendered obsolete. This situation paves the way for further study aimed at establishing more specific and stable editorial criteria. The objective would

be to establish editorial standards that could apply comparable, if not identical, criteria to all *software papers*. This would begin with a review of the standards that journals are applying so that it becomes possible to determine the most suitable models to enhance *software papers* and establish the most effective practices that may be universally applied.

Another major issue in scientific publishing is software citation, which is often unclear and inconsistent. Unlike books and journals, until now, there has been a lack of consistent methods for appropriately citing research software utilised by researchers in experiments and then in research papers, making them possibly ambiguous. Such a situation makes it difficult for other researchers to find and apply the same software, which disrupts experiment reproducibility. Secondly, the absence of correct citation minimises the likelihood of the developers being properly credited for their work, whereby they may fail to be given appropriate recognition. While *software journals* can be a useful tool to address this issue, the lack of uniform citation standards makes them subject to the same critical issue. To counter these problems, the Software Sustainability Institute has made a list of recommendations (JACKSON, N.D.). Among these elements, it is important to include essential ones in the software citation, such as authors' names, software title, version used, year of release, repository or primary source, and, if required, a persistent identifier such as a DOI. Having these details provides stable, verifiable, and traceable citations and precludes issues associated with incorrect or ambiguous references. The Software Sustainability Institute also points out the need to give a complete description of the software, including important details such as its primary features, the purpose for which it was developed, the necessary system requirements for using it, and the license. Moreover, it shows a typical trend: the tendency to reference a paper related to the software instead of the software itself. Although such an approach may offer helpful background, it may not contain valuable information, such as the precise version of the code or subsequent versions. Since software is a dynamic entity open to modification, simply referencing a paper does not guarantee that the code mentioned is the version which was actually employed in the research. Therefore, the institute recommends that the software be referenced directly, using persistent identifiers and stable repositories, in order to ensure stable access to the correct versions. Such complexity is added to by the fact that most bibliographic platforms and academic databases employ poor citation styles for software, failing to recognise it as an independent entity. Indeed, software is usually quoted in a general way as an "electronic document" or "website," without any systematic method to track updates and versions. Moreover, current citation styles, originally designed for books, articles, and datasets, lack dedicated fields for important information such as version, archiving source, and license. The absence of these fields results in incomplete and non-standardized citations, which restrict the visibility of software in academic indices and its assessment as a scientific output. To address these deficits, the

Software Sustainability Institute recommends adopting software-specific citation styles, e.g., those created by FORCE11 and Software Heritage (JACKSON, N.D.). These styles need to be incorporated into mainstream bibliographic management packages so that recognition becomes simpler and software is better evaluated in scholarly publications.

Lastly, another significantly relevant point arising from the analysis is the function of Article Processing Charges (APC) in the publication system of *software papers* and how they affect the application of Open Science principles in real life. As open access publishing is perceived to be an essential mechanism for the guarantee of openness and reusability of research software, the business model adopted by most publications presents an enormous obstacle to equitable access to publication. An overview of publication cost testifies to the great disparity between the actual costs of production of scientific articles and publishers' APCs. Research by Grossmann and Bremms (GROSSMANN AND BREMBS, 2021) has estimated that the actual costs of the publication of an article, taking into account all the stages of the editorial process—peer review, manuscript management, formatting, indexing, and archiving—might be as little as under 200 dollars in highly efficient platforms to around 1,000 dollars for highly selective journals. Yet, as noted above under Chapter III, *hybrid software journals*, and most open access journals, impose APCs ranging from 199 to 3,680 dollars, with a mean fee of 2,270 dollars. This observation indicates that a considerable portion of these charges cannot be associated directly with editorial tasks, but they have been ascribed to profit-maximising commercial endeavours. This discrepancy presents serious questions regarding fairness and financial sustainability. In fact, APCs are often paid by the researchers themselves or their institutions. As such, access to publication is made significantly unequal, favouring those who are better off with more finances and discriminating against researchers from institutions working on limited budgets or from nations that invest less in research. The danger lies in the open access principle being achieved only for individuals and organisations with the requisite financial capabilities, thereby focusing scientific production inside the walls of universities and more well-funded institutions. APCs are a funding obstacle that effectively bars those lacking access to adequate funding. This generates a dynamic in which visibility and dissemination of research are based not solely on the quality of the research work but also on the author's available economic resources. Consequently, the decision on where to publish a research paper might increasingly be based not on the suitability of the editorial context but on the publication fee. Grossmann and Brembs' study (GROSSMANN AND BREMBS, 2021) has pointed out that the exorbitant price of scientific journals cannot be solely the result of editorial effort but is affected by a variety of indirect costs as well as by the business model of the publishers. Besides the expenses involved in processing high-impact journals with high rejection rates, a significant portion of revenues for scholarly publishers is allocated towards branding,

marketing, lobbying, and paywall charges. These elements do not directly enhance the quality of the publication but play a dominant role in deciding the amount being paid by authors in the end. Furthermore, the opacity of editorial costs means that researchers and institutions cannot negotiate more favourable rates nor promote more sustainable models without difficulty. Consistent with aligning the publication of *software papers* with the principles of open science, the prevailing APC model needs to be reconsidered for more sustainable and equitable alternatives. Specific publishing schemes prove that authors can publish scientific articles without subjecting them to exaggerated costs; institutions such as SciELO³⁵, Open Library of Humanities³⁶ and arXiv³⁷ use fund models that do not require immediate payment from the authors but radically minimise financial thresholds. There are, nevertheless, some countries and organisations that have initiated trials of solutions in terms of covering publication expenses through central financing, thereby lowering the economic burden on individual scientists. A fundamental step would be to obtain greater transparency regarding publication costs, thereby enabling researchers to know the publication cost. To this end, the diamond publishing model presents a compelling alternative since it mandates that the costs of publication are met by other parties (e.g., funding agencies or public and private institutions) without imposing any cost on the authors.

Research software faces several challenges affecting its development, maintenance, and usability, including the lack of sustainable funding (research software is often developed as part of short-term projects with limited funds for maintenance), insufficient recognition and career incentives for developers (rewarding publications is more common than rewarding research software), limited software engineering best practices (research software developers usually lack formal training in software development). However, it is fundamental to modern science across nearly all disciplines, and thus, it is paramount that its management be improved. Effective research software management requires collaboration between developers, researchers, institutions, and funders. Integrating Open Science principles into research software management can enhance reproducibility, ensure long-term sustainability, and foster global collaboration. Open Science makes research software more FAIR, transparent, and impactful while providing recognition for software developers in academic research. In these settings, *software journals* can play a fundamental role in capacity building, mainly if they focus on promoting the dissemination, validation, and preservation of this specific kind of scholarly knowledge in cooperation with the other actors of the research ecosystem.

³⁵ www.scielo.org

³⁶ <https://www.openlibhums.org/>

³⁷ <https://arxiv.org/>

References

- About the Software Sustainability Institute [WWW Document], n.d. . Softw. Sustain. Inst. URL <https://www.software.ac.uk/about>
- SORTEE About [WWW Document], n.d. . SORTEE. URL <https://www.sortee.org/about/>
- CoARA About [WWW Document], n.d. . CoARA. URL <https://coara.eu/>
- About Zenodo [WWW Document], n.d. . Zenodo. URL <https://about.zenodo.org/>
- Acm Toms Staff, C., 1975. Algorithms Distribution Service. ACM Trans. Math. Softw. 1, 4. <https://doi.org/10.1145/355626.355628>
- An Introduction to Tivoization [WWW Document], n.d. . LINFO. URL <https://www.lininfo.org/tivoization.html>
- Background [WWW Document], n.d. . Bp. Open Access Initiat. URL <https://www.budapestopenaccessinitiative.org/>
- Barker, M., Chue Hong, N.P., Katz, D.S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L.J., Gruenpeter, M., Martinez, P.A., Honeyman, T., 2022. Introducing the FAIR Principles for research software. Sci. Data 9, 622. <https://doi.org/10.1038/s41597-022-01710-x>
- Burke, P.G., 1969a. Introduction. Comput. Phys. Commun. 1, v–vi. [https://doi.org/10.1016/0010-4655\(69\)90008-3](https://doi.org/10.1016/0010-4655(69)90008-3)
- Bush, V., 1996. As we may think. Interactions 3, 35–46. <https://doi.org/10.1145/227181.227186>
- Carey, V., 2009. Editorial. R J. 1.
- Chue Hong, N., 2015. Google Code is shutting down - what should you do next? [WWW Document]. Softw. Sustain. Inst. URL <https://www.software.ac.uk/blog/google-code-shutting-down-what-should-you-do-next>
- Chue Hong, N., n.d. Announcing the Journal of Open Research Software - a software metajournal [WWW Document]. Softw. Sustain. Inst. URL <https://www.software.ac.uk/blog/announcing-journal-open-research-software-software-metajournal>
- Di Cosmo, R., Zacchiroli, S., 2023. The Software Heritage Open Science Ecosystem, in: Mens, T., De Roover, C., Cleve, A. (Eds.), Software Ecosystems. Springer

International Publishing, Cham, pp. 33–61.
https://doi.org/10.1007/978-3-031-36060-2_2

European Commission. Directorate General for Research and Innovation., 2020. Scholarly infrastructures for research software: report from the EOSC Executive Board Working Group (WG) Architecture Task Force (TF) SIRS. Publications Office, LU.

European Open Science Cloud (EOSC) - European Commission [WWW Document], 2024. URL
https://research-and-innovation.ec.europa.eu/strategy/strategy-research-and-innovation/our-digital-future/open-science/european-open-science-cloud-eosc_en (accessed 1.18.25).

Felderer, M., Goedicke, M., Grunske, L., Hasselbring, W., Lamprecht, A.-L., Rumpe, B., 2023. Toward Research Software Engineering Research.
<https://doi.org/10.5281/ZENODO.8020525>

Fuchs, C., Sandoval, M., 2013. The Diamond Model of Open Access Publishing: Why Policy Makers, Scholars, Universities, Libraries, Labour Unions and the Publishing World Need to Take Non-Commercial, Non-Profit Open Access Serious. *TripleC Commun. Capital. Crit. Open Access J. Glob. Sustain. Inf. Soc.* 11, 428–443. <https://doi.org/10.31269/triplec.v11i2.502>

GNU General Public License, version 1 [WWW Document], 1989. . GNU. URL
<https://www.gnu.org/licenses/old-licenses/gpl-1.0.html>

GNU General Public License, version 2 [WWW Document], 1991. . GNU. URL
<https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

GNU General Public License, version 3 [WWW Document], 2007. . GNU. URL
<https://www.gnu.org/licenses/gpl-3.0.html>

Gold OA e OA Ibrido [WWW Document], n.d. . CNR Bibl. Dario Nobili. URL
<http://biblioteca.bo.cnr.it/index.php/it/open-access/pubblicare-oa/item/275-gold-oa-e-oa-ibrido>

Gomes, D.G.E., Pottier, P., Crystal-Ornelas, R., Hudgins, E.J., Foroughirad, V., Sánchez-Reyes, L.L., Turba, R., Martinez, P.A., Moreau, D., Bertram, M.G., Smout, C.A., Gaynor, K.M., 2022. Why don't we share data and code? Perceived barriers and benefits to public archiving practices. *Proc. R. Soc. B Biol. Sci.* 289, 20221113. <https://doi.org/10.1098/rspb.2022.1113>

Gomez-Diaz, T., Recio, T., 2022. Research Software vs. Research Data I: Towards a Research Data definition in the Open Science context. *F1000Research* 11, 118. <https://doi.org/10.12688/f1000research.78195.2>

Grossmann, A., Brembs, B., 2021. Current market rates for scholarly publishing services. *F1000Research* 10, 20. <https://doi.org/10.12688/f1000research.27468.2>

- Gruenpeter, M., Katz, D.S., Lamprecht, A.-L., Honeyman, T., Garijo, D., Struck, A., Niehues, A., Martinez, P.A., Castro, L.J., Rabemanantsoa, T., Chue Hong, N.P., Martinez-Ortiz, C., Sesink, L., Liffers, M., Fouilloux, A.C., Erdmann, C., Peroni, S., Martinez Lavanchy, P., Todorov, I., Sinha, M., 2021. Defining Research Software: a controversial discussion. Zenodo.
<https://doi.org/10.5281/zenodo.5504016>
- Guides for researchers [WWW Document], n.d. . Softw. Sustain. Inst. URL
<https://www.software.ac.uk/guide/guides-researchers>
- Hettrick, S., 2014. It's impossible to conduct research without software, say 7 out of 10 UK researchers [WWW Document]. Softw. Sustain. Inst. URL
<https://www.software.ac.uk/blog/its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers>
- Hettrick, S., Antonioletti, M., Carr, L., Chue Hong, N., Crouch, S., De Roure, D., Emsley, I., Goble, C., Hay, A., Inupakutika, D., Jackson, M., Nenadic, A., Parkinson, T., Parsons, M.I., Pawlik, A., Peru, G., Proeme, A., Robinson, J., Sufi, S., 2014. Uk Research Software Survey 2014.
<https://doi.org/10.5281/ZENODO.14809>
- Hettrick, S., Philippe, O., Chue Hong, N., Sufi, S., Silva, R., Peru, G., 2016. Software used in research based on combined surveys.
<https://doi.org/10.5281/zenodo.60276>
- History of the OSI [WWW Document], 2006. . Open Source Initiat. URL
<https://opensource.org/history>
- Hopkins, T., 2002. Renovating the collected algorithms from ACM. ACM Trans. Math. Softw. 28, 59–74. <https://doi.org/10.1145/513001.513005>
- I principi, gli obiettivi, i benefici e le prassi della Scienza Aperta - Sistema Bibliotecario di Ateneo — SBA [WWW Document], n.d. URL
<https://sba.unibo.it/it/almadl/open-access-e-open-science/principi-obiettivi-benefici-prassi-scienza-aperta> (accessed 1.18.25).
- Instructions to authors, 1969. . Comput. Phys. Commun. 1, x–xviii.
[https://doi.org/10.1016/0010-4655\(69\)90010-1](https://doi.org/10.1016/0010-4655(69)90010-1)
- Jackson, M., n.d. How to cite and describe software [WWW Document]. Softw. Sustain. Inst. URL
<https://www.software.ac.uk/publication/how-cite-and-describe-software>
- Jacobsen, A., De Miranda Azevedo, R., Juty, N., Batista, D., Coles, S., Cornet, R., Courtot, M., Crosas, M., Dumontier, M., Evelo, C.T., Goble, C., Guizzardi, G., Hansen, K.K., Hasnain, A., Hettne, K., Heringa, J., Hooft, R.W.W., Imming, M., Jeffery, K.G., Kaliyaperumal, R., Kersloot, M.G., Kirkpatrick, C.R., Kuhn, T.,

- Labastida, I., Magagna, B., McQuilton, P., Meyers, N., Montesanti, A., Van Reisen, M., Rocca-Serra, P., Pergl, R., Sansone, S.-A., Da Silva Santos, L.O.B., Schneider, J., Strawn, G., Thompson, M., Waagmeester, A., Weigel, T., Wilkinson, M.D., Willighagen, E.L., Wittenburg, P., Roos, M., Mons, B., Schultes, E., 2020. FAIR Principles: Interpretations and Implementation Considerations. *Data Intell.* 2, 10–29. https://doi.org/10.1162/dint_r_00024
- Jay, C., Haines, R., Katz, D.S., 2021. Software Must be Recognised as an Important Output of Scholarly Research. *Int. J. Digit. Curation* 16, 6. <https://doi.org/10.2218/ijdc.v16i1.745>
- Katz, D.S., Barba, L.A., Niemeyer, K., Smith, A.M., 2019. Cost models for running an online open journal. *J. Open Source Softw. Blog*. <https://doi.org/10.59349/g4fz2-1cr36>
- Katz, D.S., Gruenpeter, M., Honeyman, T., Hwang, L., Wilkinson, M.D., Sochat, V., Anzt, H., Goble, C., Subgroup 1, for F., 2021. A Fresh Look at FAIR for Research Software. <https://doi.org/10.48550/arXiv.2101.10883>
- Katz, D.S., Niemeyer, K.E., Smith, A.M., 2018. Publish your software: Introducing the Journal of Open Source Software (JOSS). *Comput. Sci. Eng.* 20, 84–88. <https://doi.org/10.1109/MCSE.2018.03221930>
- Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., Van De Sandt, S., Ison, J., Martinez, P.A., McQuilton, P., Valencia, A., Harrow, J., Psomopoulos, F., Gelpi, J.Ll., Chue Hong, N., Goble, C., Capella-Gutierrez, S., 2020. Towards FAIR principles for research software. *Data Sci.* 3, 37–59. <https://doi.org/10.3233/DS-190026>
- Leeuw, J.D., 2009. Journal of Statistical Software. *WIREs Comput. Stat.* 1, 128–129. <https://doi.org/10.1002/wics.10>
- Lenares, D., 1999. Faculty Use of Electronic Journals at Research Institutions.
- Lu, Q., Zhu, L., Xu, X., Whittle, J., Zowghi, D., Jacquet, A., 2023. Operationalizing Responsible AI at Scale: CSIRO Data61’s Pattern-Oriented Responsible AI Engineering Approach. *Commun. ACM* 66, 64–66. <https://doi.org/10.1145/3589946>
- Margoni, T., Peters, D., 2016. Creative Commons Licenses: Empowering Open Access. *SSRN Electron. J.* <https://doi.org/10.2139/ssrn.2746044>
- Morais, R., Brailsford, I., 2019. Knowledge Visualization for Research Design: The Case of the Idea Puzzle Software at the University of Auckland. <https://doi.org/10.4018/978-1-5225-7065-3.ch003>
- Morel, J.-M., 2010. . Off. Launching IPOL J. URL https://www.ipol.im/news/20100429_launching/

- Ozgur, C., Jha, S., Shen, Y., 2020. Software Programs Used in Research. Presented at the MWDSI, Bowling Green, Ohio.
- Pouchard, L., Islam, T., Nicolae, B., 2022. Challenges for Implementing FAIR Digital Objects with High Performance Workflows. *Res. Ideas Outcomes* 8, e94835. <https://doi.org/10.3897/rio.8.e94835>
- Pouchard, L.C., 2023. FAIR Enabling Re-Use of Data-Intensive Workflows and Scientific Reproducibility, in: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. Presented at the ICPE '23: ACM/SPEC International Conference on Performance Engineering, ACM, Coimbra Portugal, pp. 329–329. <https://doi.org/10.1145/3578245.3586012>
- Rice, J.R., 1975. Purpose and Scope. *ACM Trans. Math. Softw.* 1, 1–3. <https://doi.org/10.1145/355626.355627>
- Rossi, G., Caso, R., Castelli, D., Giglia, E., 2022. Piano nazionale per la scienza aperta.
- Ryan, S., Nadal, C., Doherty, G., 2023. Integrating Fairness in the Software Design Process: An Interview Study With HCI and ML Experts. *IEEE Access* 11, 29296–29313. <https://doi.org/10.1109/ACCESS.2023.3260639>
- Scott, N.S., Hibbert, A., Ballantyne, J., Fritzsche, S., Hazel, A.L., Landau, D.P., Walker, D.W., Was, Z., 2020. CPC's 50th Anniversary: Celebrating 50 years of open-source software in computational physics. *Comput. Phys. Commun.* 252, 107269. <https://doi.org/10.1016/j.cpc.2020.107269>
- Smith, A., 2016. Announcing The Journal of Open Source Software [WWW Document]. *Announc. J. Open Source Softw.* URL <https://www.arfon.org/announcing-the-journal-of-open-source-software>
- Soremekun, E., Papadakis, M., Cordy, M., Traon, Y.L., 2022. Software Fairness: An Analysis and Survey. <https://doi.org/10.48550/ARXIV.2205.08809>
- The Open Source Definition [WWW Document], 2024. . Open Source Initiat. URL <https://opensource.org/osd>
- UNESCO Recommendation on Open Science [WWW Document], n.d. . UNESCO. URL <https://www.unesco.org/en/open-science/about>
- Weber, N., 2022. Research Software Publication Policy Case Study. <https://doi.org/10.48550/ARXIV.2206.053>