

Article

# Opening Software Research Data 5Ws+1H

Anastasia Terzi <sup>†</sup> and Stamatia Bibi <sup>\*,†</sup>

Department of Electrical and Computer Engineering, University of Western Macedonia, 50100 Kozani, Greece; a.terzi@uowm.gr

\* Correspondence: sbibi@uowm.gr

<sup>†</sup> These authors contributed equally to this work.

**Abstract:** Open Science describes the movement of making any research artifact available to the public, fostering sharing and collaboration. While sharing the source code is a popular Open Science practice in software research and development, there is still a lot of work to be done to achieve the openness of the whole research and development cycle from the conception to the preservation phase. In this direction, the software engineering community faces significant challenges in adopting open science practices due to the complexity of the data, the heterogeneity of the development environments and the diversity of the application domains. In this paper, through the discussion of the 5Ws+1H (Why, Who, What, When, Where, and How) questions that are referred to as the Kipling's framework, we aim to provide a structured guideline to motivate and assist the software engineering community on the journey to data openness. Also, we demonstrate the practical application of these guidelines through a use case on opening research data.

**Keywords:** open data; software engineering; open software; kipling method

## 1. Introduction

According to Russell [1], “openness” is a synthesis of technology and ideology, representing the combination of technology, democracy, and innovation. The openness spirit is constantly gaining ground globally, motivating both the scientific and the industrial communities to share whatever they consider relevant, from intangible artifacts such as methodologies and “learned lessons” to tangible results such as data, code, and algorithms. The culture of openness started from science, as expected, at the beginning of the 17th century, with the advent of academic journals. At the beginning of the 90s, the first big change occurred with the invention of arXiv, where scientists could freely share their research papers prior to publication. Since then, several other initiatives have taken place to “democratize” access to knowledge, such as Open Access to journals (2004), Open-Funded Research (2013) and, of course, the Open Data movement (2008). The latter started by sharing government data and in recent years has expanded to all data-intensive domains [2]. Open science is a movement aimed at making all artifacts resulting from scientific research activities accessible to anyone, without any barriers [3]. As an umbrella term, open science encompasses several facets of openness, including open access, open data, open source, open government, open notebooks, and open standards [4]. Open access refers to publications that are freely available on the public Internet without financial, legal or technical barriers, allowing individuals to read, download, copy, distribute, print, search, or link to the full texts of publications for any legal purpose [5]. Open data is similar, but applies to any data produced during research activities, such as raw data from controlled experiments. The level of openness may also present various accessibility concerns, sometimes requiring specific requests and consents to access the full dataset. Open source in open science mirrors the concept well known in the computer science community: making source code freely available for use and modification. Given the fact that research and practice have become increasingly data-driven, data now, more than ever, play a critical role in enabling



**Citation:** Terzi, A.; Bibi, S. Opening Software Research Data 5Ws+1H. *Software* **2024**, *3*, 411–441. <https://doi.org/10.3390/software3040021>

Academic Editor: Ian Gorton

Received: 10 September 2024

Revised: 20 September 2024

Accepted: 24 September 2024

Published: 26 September 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

and accelerating open science. Both sharing and using open research data can contribute to scientific advancement.

In software engineering, the opening movement centers around the importance of software products to be distributed alongside the constructive data. While platforms like GitHub are widely used by developers to manage and store software code, they are not used for storing the full range of data produced during the software development process. Such data include version control data, bug reports, technical and non-technical requirements, architecture models, specifications along with test data. On the other hand, there are also software metadata that can be useful, such as file creation dates, author names, storage locations, wikis, reports, etc. The software metadata is heterogeneous, disconnected and defined at different levels of detail [6] from different users, rendering the process of opening software data highly challenging. Other issues that may hinder software data disclosure include the origin of data (data from sensitive environments e.g., highly confidential data), and the involvement of human participants that may pose privacy constraints. Additionally, since software research applies horizontally to various interdisciplinary domains (from mathematics to psychology and sociology) there are many cases where both qualitative and quantitative data are produced depending on types and theories emerging from backgrounds other than software engineering, a fact that also creates significant challenges.

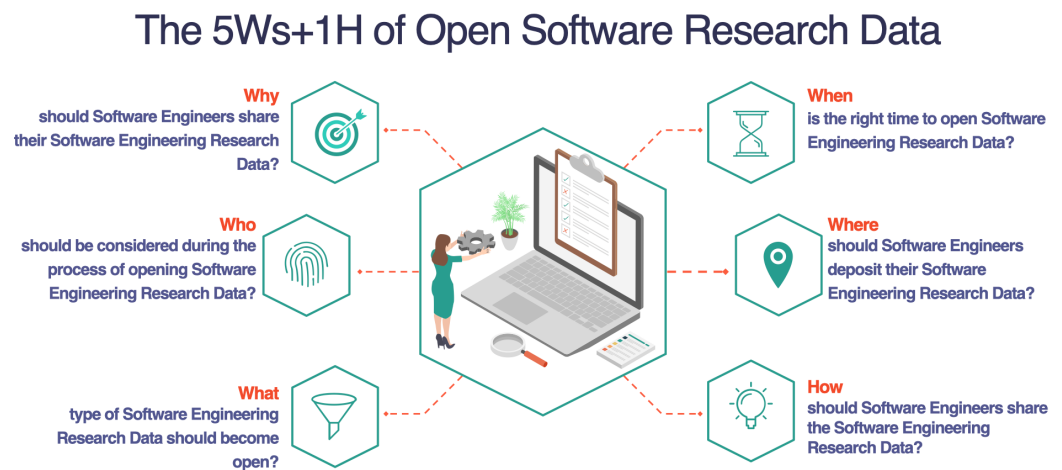
Therefore, given the aforementioned challenges, it is important to define a data openness strategy that will guide the efforts of the software community towards sharing data. While an extensive amount of work exists on the field of open data research, few approaches have paid attention to the openness of software research data and none of them to the software development process as whole.

As we move towards this future where research is widely accessible to a diverse audience—from academic experts to everyday users—the demand for transparent processes and detailed documentation is increasing. The literature primarily focuses on specific artifacts produced during the software development lifecycle, with one of the most recognizable artifacts of open science being the software itself [7–9]. Scientific software is key for the reproducibility of scientific results, as it helps others understand how a data product has been created or modified as part of research and avoids replicating effort.

Many discussions on the topic emphasize the importance of making code findable, accessible, interoperable, and reusable (FAIR principles) [10], encouraging scientists to fairly describe and share the software metadata. While code is the most widely used aspect of open data, there is also a growing demand for artifacts related to testing code [11–14], and supporting documentation, including maintenance guides [11,15], technical specifications [16–18], and quality metrics [19], those methods are being used as part of research replication processes or even for research implementation. Furthermore, an increasing number of researchers are highlighting the significance of proper citation and the sharing of data, underscoring the role of transparency and collaboration in advancing both software engineering and open science practices [20,21]. The rising demand for transparency on software process led scientific community, academic publishers, and public stakeholders to start taking measures towards the complete opening of software research, making more and more software artifacts available to public demand. Recently, the literature has shifted focus toward lesser-discussed artifacts that play crucial roles in software research. At their research Wnuk et al. [22] discuss how openly sharing requirements can alter the landscape of Software research. Similarly, the studies by Ho-Quang et al. [23] and Hebig et al. [24] they discovered that sharing artifacts such as UML diagrams UML diagrams, enhances understanding of software and increases the levels of possible adoption. More researches on the topic [25] has demonstrated that involving users in the development phases—through open sharing of processes—leads to more accurate outcomes and higher-quality products, while also increases trust and loyalty toward the researcher [9,19].

Recent reports [26] note that the lack of guidelines for the opening of research data makes the opening of data a rather time-consuming process that also contributes to the lack of motivation associated with data sharing among researchers and practitioners [27]. The challenge extends beyond merely sharing research data; it encompasses the need for a structured plan that enables engineers to share every step of their research process knowing that data security is guaranteed through citation while ensuring ownership and accessibility in an intelligible way for all stakeholders. This paper aims to address the gap by developing a set of guidelines that could motivate and guide software engineering practitioners and researchers towards contributing to Open Science. The proposed framework is formulated around the 5Ws+1H management practices [28] and aims to capture all the factual elements needed for a complete and objective understanding of a software openness project [29] by answering the questions related to Why, Who, What, When, Where, and How. The 5W+1H practice can be applied in different contexts [30] and is widely utilized to provide a delicate analysis of a specific field or knowledge, such as production management, marketing, engineering [31].

In the context of our study, the 5Ws+1H questions regarding Open Software Research Data are being presented in Figure 1 and are formulated as following:



**Figure 1.** The 5Ws+1H of Open Software Research Data.

#### **Why should Software Engineers share their Software Engineering Research Data?**

Promoting an open and collaborative culture in software engineering research requires awareness of the benefits of open research. The literature has focused extensively on the broad benefits of open data, such as increased openness, explainability, reusability, and resilience. However, most studies concentrate on individual artifacts created during software development, rather than taking into account the full software development life-cycle. With respect to each phase of software research development, our study aims to integrate existing knowledge and emphasize the specific benefits of data sharing at each step of software development. By doing so, we want to motivate engineers to step forward and open more data to gain the maximum benefits.

#### **Who should be considered during the process of opening Software Engineering Research Data?**

When sharing data, there are several types of stakeholders to consider: those who participated/ contributed to the data creation during the development phase, those who will use the product as a whole and those who will reuse and build upon the shared artifacts. This question underscores the importance of selecting and sharing data based on the unique requirements, understanding, and access points of each stakeholder group.

### **What type of Software Engineering Research Data should become open?**

Identifying the types of data that should be shared is essential to standardize open data practices. Software development generates a diverse range of artifacts. Although source code is the most common artifact, other outcomes include binary code, version control system metadata, bug reports, developer conversations via several media, code reviews, test data, and documentation. In this question, we will discuss effective data sharing considering two key factors: (a) the data that researchers can share, considering the data's state and any disclosure constraints, and (b) the intended purpose or expected use of the research data.

### **When is the right time to open Software Engineering Research Data?**

Proper timing ensures that the data is available when it is most needed. We suggest a three-phase opening process for each phase of the software development life cycle, including sharing valuable artifacts from the initial requirements and business planning stages through design, implementation, testing, deployment, and maintenance.

### **Where should Software Engineers deposit their Software Engineering Research Data?**

To maximize the reach and impact of shared data, choosing the appropriate repositories for depositing Software Engineering Research Data is important. Data archives can be used for different purposes, for the discovery of data, and/or for publishing data. Depending on the type of Data that the Software engineer intends to share, we explore the most suitable venues for data storage. There are a plethora of options, ranging from informal peer-to-peer exchanges and hosting on project or institutional websites to depositing in specialized data centers and submitting to data journals for publication. The repository decision is based on the level of support provided, the maturity of the research and the target audience.

### **How should Software Engineers share the Software Engineering Research Data?**

Methods and practices for sharing data are critical to maintaining its quality, usability, and compliance with legal and ethical standards. To effectively manage and share software research data, it is important to develop a Data Management Plan (DMP) and a Software Management Plan (SMP). These plans detail how data will be captured, managed, stored and shared, as well as how the integrity, security, and confidentiality of data will be maintained throughout and beyond the research project. We provide a list of requirements for these processes, along with suggestions for open tools that can support effective data management and sharing.

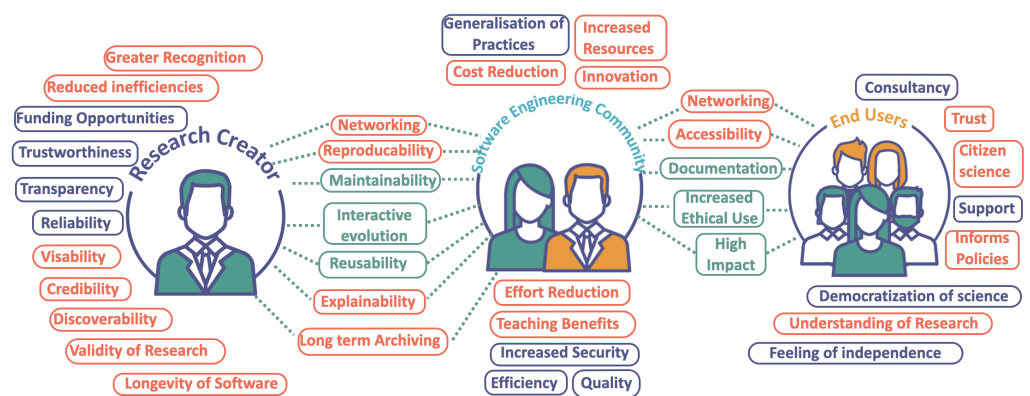
The application of our proposed framework to a previous work of ours, "Using Code from ChatGPT: Finding Patterns in the Developers' Interaction with ChatGPT" [32], is presented herein. The goal of this use case is to demonstrate how the proposed framework can be applied at any stage of the research cycle and how developers can take full advantage of it. For the aforementioned example, the exact path following the 5Ws+1H questions was tested, providing even deeper insights into the framework. That very example, is a simplified overview of previous work, neglecting some challenges we had to face that are out of scope of this research.

The rest of the paper is organized as follows. Sections 2–7 discuss in detail the proposed 5W+1H framework for opening software engineering data, and Section 8 presents a practical application of the framework in a real-world case study within empirical software engineering research. Section 9 addresses the challenges, implications, and potential future work related to the openness of software engineering data. Finally, Section 10 concludes the paper.

## **2. Why Should Software Engineers Share Their Software Engineering Research Data?**

The literature has extensively focused on the broad benefits of open data, such as transparency, explainability, reusability, and the potential for maintenance and evolution. Additionally, researchers argue that open access is capable of providing social benefits, such as educational tools to train new scientists, shared common resources to promote sustainable research, and the ability to promote scientific public outreach and engagement. The literature also points to research-intrinsic motives connected with dissemination and

recognition, as the personal desire to publish data [33]. In Figure 2, we have adapted the context of “Benefits of Open Research Data Infographic” [34] to align more closely with software engineering and the specific objectives of this study. The red color has been used to highlight the attributes that remain unchanged from the original figure, signifying their relevance and applicability to the domain of software engineering. The aforementioned figure discusses mainly the academic benefits of open research data in terms of visibility, citations, collaboration, and societal trust. Our contribution on the other hand tries to bridge the aspects of technical and societal benefits towards a discussion on how open research data impacts software development practices directly. To this end, we have chosen to narrow down the stakeholder categories into: Researchers, Software Engineering Community and End users and attempt to highlight not just the academic benefits but also the technical aspects of open sharing.



**Figure 2.** Benefits of Sharing Software Engineer Data for different users.

One of the main concerns of software engineers is the opaque nature of Software Research Data that raises questions about the intention of processes running in the background. The benefits of sharing identified by **Research Creators** relate mainly to the impact of their work. Opening the Software Research Data will build confidence and trust among practitioners, raising the software’s credibility. With more people trusting the developers, the data undergoes tests by a diverse pool of developers and researchers, leading to the identification and rectification of bugs. The community contribution through tests helps with software evolution and improvement, increasing not only the security of data but also the longevity of software.

Open data reduces duplication of effort by providing a shared resource on which researchers can build. By utilizing openly shared existing datasets, the **Software Engineering Community** can reduce costs and effort on individual projects while promoting sustainable research practices. When artifacts are used by different users, security issues and vulnerabilities are identified and addressed more promptly, leading to more secure software solutions. Another advantage of sharing data openly is the promotion of best practices across the research community, facilitating the creation of proper guidelines that turn complex processes into easily understandable ones. Extended documentation (metadata) impacts the dissemination and adoption of findings. As researchers explore new ideas and approaches using shared resources, innovative findings are highlighted.

Opening up research data is not only beneficial for researchers, but also for society overall, as it provides a democratic scientific knowledge-sharing platform. For the **End Users**, opening process increases their trust in scientific research by allowing them to better understand the underlying processes. This transparency boosts public confidence in science, while also encouraging the support of the Public in research projects. This democratization of software engineering can build a community of educated individuals capable of evaluating research findings and providing useful input. Policymakers may use this information to make better decisions, resulting in a more effective and scientifically grounded government.



It is obvious that the opening processing creates a chain reaction of advantages for the stakeholders, strengthening not only their capabilities but also building a strong dependent network between them. This general view of the benefits of data sharing is raising the question of why open data is not yet a reality for Software Engineers researchers [35].

Borgman [36] mentions four significant reasons for sharing research data: reproduction of research, release of public assets, leveraging investment in research, and promotion of research and innovation. Although there are numerous benefits and motivations associated with open data sharing, research by Reilly et al. [37] indicates that while researchers are generally willing to use others' data if they can validate it, they are often hesitant to share their own. Participants mentioned various barriers, whether real or perceived, as reasons not to share or not to trust shared data. Those barriers are closely related to the opaque nature of the Software Research Data, which might give uncertainty to a few processes behind the data. Most of the studies done on OSR tend to focus on the sharing of unique artifacts, with little consideration for the wide range of artifacts created throughout the software development life-cycle. Their attention has resulted in increased concern over the quality of data used in research; raising trust issues among practitioners.

To overcome this challenge by focusing on the seven phases procedure of software development, we have built the following table (Table 1). This table highlights the significance of openly sharing each artifact, which not only serves as part of the final software but also is a complete, autonomous product.

**Table 1.** What is the benefit of open sharing software research artifacts?

Artifact (What)	Benefits of Sharing (Why)
<b>Phase 1: Project Initiation</b>	
<b>Project Plan</b>	<ul style="list-style-type: none"> <li>- Provides clarity on the research objectives, methodology, and expected outcomes in advance.</li> <li>- Users are aware of upcoming milestones, which helps in coordinating efforts.</li> <li>- Funders are aware of how their resources will be utilized, increasing the likelihood of future support, ensuring confidence in the research.</li> <li>- Prevents duplication of effort by informing others of ongoing work in the field [8,38,39].</li> </ul>
<b>Phase 2: Analysis and Detailed Planning</b>	
<b>Business Requirement Documentation (BRD)</b>	- Sharing requirements ensures it cannot be used as a differentiator by competitors.
<b>User Requirement Specifications</b>	- Researchers can inject new requirements into the open source community by outsourcing the cost of prototype development.
<b>Software Requirement Specifications (SRS)</b>	- Public sharing of documents facilitates automatic extraction of requirements based on evolving needs [22,25].
<b>Configuration Management (CM) Plan</b>	- Increases reliability over project's longevity
<b>Phase 3: Design</b>	
<b>Detailed Design Specifications (DDS)</b>	<ul style="list-style-type: none"> <li>- Provides better understanding of the software structure.</li> <li>- Defines a set of constraints on subsequent implementation.</li> <li>- Provides consistent between intention and implementation.</li> <li>- Enables faster verification and increases user acceptance.</li> <li>- Improves communication and reduces ambiguity regarding structure and use.</li> <li>- Helps new contributors onboard effectively.</li> <li>- Provides better understanding on costs and delivery time of system [23,24,40,41].</li> </ul>

Table 1. Cont.

Artifact (What)	Benefits of Sharing (Why)
<b>Phase 4: Software Construction</b>	
<b>Unit Code</b> <b>Software packages Artifacts</b>	The research on Open Software has thoroughly analyzed the benefits of sharing source code, including enhanced quality, improved security, flexibility of use, reduced costs for improvements, increased collaboration, and better user support. Additionally, we propose the following benefits: <ul style="list-style-type: none"> <li>- Reduces code redundancy, preventing duplicated efforts.</li> <li>- Increases recognition and researchers' reputation.</li> <li>- Expands opportunities for reusability, allows users to replicate, validate, and build upon existing work [7–9,19].</li> </ul>
<b>Phase 5: Testing</b>	
<b>Code Inspection</b> <b>Test Summary Report</b>	<ul style="list-style-type: none"> <li>- Clarifies the intended behavior of the software at the time it was last programmed.</li> <li>- Helps ensure that proposed changes do not break other projects within the ecosystem.</li> <li>- Distributes the labor of maintaining code quality, reducing the burden on code reviewers.</li> <li>- Supports validation of submitted code and identification of errors by more contributors, increasing the likelihood of detecting and resolving bugs.</li> <li>- Enables test modifications and improvements without disrupting the main development process, supporting continuous integration and quality assurance [8,11–14].</li> </ul>
<b>User Acceptance</b>	<ul style="list-style-type: none"> <li>- Demonstrates a commitment to meeting users' needs and addressing their concerns</li> <li>- Highly involve users part on the development process</li> <li>- Increases research reputation and discoverability [9,19]</li> </ul>
<b>Risk Management</b>	<ul style="list-style-type: none"> <li>- Simplifies internal routines by providing external feedback.</li> <li>- Facilitates the exchange of standards by aligning practices with widely accepted benchmarks.</li> <li>- Allows researchers to compare their local routines with other implementations, leading to adaptation and potential improvement [42].</li> </ul>
<b>Phase 6: Implementation</b>	
<b>Production Environment</b> <b>Live System</b>	<ul style="list-style-type: none"> <li>- Provides users real time experience increasing trust on the system</li> <li>- With more people putting system under stress , bugs and errors are being easily discovered and faster addressed [7–9]</li> </ul>
<b>Phase 7: Maintenance</b>	
<b>Dependencies</b> <b>Tech Debt</b>	<ul style="list-style-type: none"> <li>- Informs users on necessary third-party components, supporting easier implementation.</li> <li>- External feedback leads to updating outdated or risky dependencies.</li> <li>- Increases code security by identifying unused or vulnerable dependencies.</li> <li>- Supports learning and adaptation by helping users manage dependencies better.</li> <li>- Reduces the cost of fixing vulnerabilities related to outdated dependencies [16–18].</li> </ul>
<b>Maintenance Guide</b>	<ul style="list-style-type: none"> <li>- Helps new contributors easily adapt, aiding in long-term project sustainability.</li> <li>- Gains users' trust by providing clear maintenance practices.</li> <li>- Distributes maintenance tasks, reducing the burden on individual maintainers as the project scales.</li> <li>- Enhances the project's longevity by ensuring consistent upkeep and adaptability within the software ecosystem [8,11,15].</li> </ul>
<b>Customer's Review</b> <b>Software metrics</b> <b>Development metrics</b> <b>Usage metrics</b>	<ul style="list-style-type: none"> <li>- Raising awareness on the software's quality.</li> <li>- Product appears appealing due to transparency of metrics.</li> <li>- Better evaluation and validation of the project.</li> <li>- Increasing willingness to use the data.</li> <li>- Builds reputation, leading to more views and increased discoverability.</li> <li>- Increase loyalty [9,19]</li> </ul>

### 3. Who Should Be Considered during the Process of Opening Software Engineering Research Data?

When considering data openness, it is essential to identify key stakeholders who will benefit from or be affected by this openness [43]. There are three main categories of subjects:

The **Research Creators, Software Engineering (SE) Academic Researchers**, are the primary beneficiaries of open sharing data. They demonstrate a commitment to openness,

reproducibility, and the skills of collaboration—all of which are highly appreciated in the academic community. When their research artifacts are made publicly available, one of the most immediate benefits is increased discoverability by a broader audience, including both individuals and organizations. This improved visibility will help increase their reputation and also create new opportunities for collaboration. As the number of citations received for their work increases, the more likely they will be considered for more opportunities to place their articles in prestigious academic journals. Publication leads to recognition, and this means that one is offered consultation opportunities, thus having more influence outside their academic parameters and into industry and beyond.

However, Research Creators should be careful when it comes to choosing which data to release, especially when the study includes human subjects, such as in surveys or empirical research. In these cases **Research Participants** should be notified about the plan for sharing and preserving the research data with their approval being necessary. When required, anonymization techniques should be used to safeguard privacy. Formal data-sharing agreements between partners are necessary in collaborative research to maintain quality while also respecting legal requirements, such as the GDPR. Data access for research participants creates a sense of involvement and trust in the research process.

The **Software Engineering Community** is made up of highly qualified professionals who demand unrestricted access to data in order to reuse or replicate artifacts in any format, to expand their research and development operations [44]. Under this term, we aim to include **Software Engineers in the private sector, Data Scientists, Developers, and Maintainers**. For these users, access to open research data not only lowers the costs and effort associated with software development it also provides exposure to innovative tools and methodologies from academic research, fostering continuous learning and the acquisition of new skills. With access to existing knowledge and products, developers no longer need to build everything from scratch, leading to faster development cycles. The open nature of research data also ensures that these artifacts are constantly tested by a diverse user base, which helps maintain high-quality standards and facilitates the quicker identification and rectification of bugs, reducing integration risks. Offering access to all available resources leads to more efficient and reliable software development practices.

**End users** benefit from the research outputs, together with the necessary system description, in an accessible and understandable format. The need to access extra software artifacts, such as test cases or requirements documentation, is typically unnecessary and might cause confusion. Examples in this category are **Research Funders** who, through data openness, enjoy improved financial returns from their research investments and improved networking. On the business end, transparency drives innovation by accessing a wider pool of ideas and integrating new insights into operations, fostering a culture of continuous improvement. The **Publishers** enjoy benefits such as independent verification and qualification of research, increased reputation, and elevated impact. As their reputation grows, they become more attractive to high-quality research submissions. **Affiliations**, such as universities and research institutions, also benefit greatly from open data in much the same way. Funders would be most likely to fund institutions known for their transparency and quality research.

When considering the benefits of sharing research data for different audiences, it is essential to understand both the advantages and the appropriate platforms for data deposition, as presented in Table 2.



**Table 2.** Why sharing Is beneficial for your Audience and Where is the right place to deposit your Data.

Audience (Who)	Why Is Beneficial	Where		
		Generalist Data Repositories	Discipline Specific Repositories	(Data) Journals
<b>SE Academic Researchers</b>	<ul style="list-style-type: none"> <li>- Additional publications.</li> <li>- Greater citation rate.</li> <li>- Wider recognition among peers.</li> <li>- Invitations to collaborate.</li> <li>- Invitations to provide consultancy [45].</li> </ul>	✓	✓	✓
<b>SE in private sector</b>	<ul style="list-style-type: none"> <li>- Cost and effort reduction due to reuse.</li> <li>- Ready made solutions requiring alterations to fit sector’s need.</li> <li>- Increased availability of verified data.</li> <li>- Acquiring new skills and knowledge [46].</li> </ul>	✓	✓	
<b>Data Scientists</b>	<ul style="list-style-type: none"> <li>- Access to Extended Open Data Sets.</li> <li>- Time and effort reduce.</li> <li>- High quality multi format data.</li> <li>- Reduced data redundancy.</li> </ul>	✓	✓	
<b>Developers</b>	<ul style="list-style-type: none"> <li>- Acquiring new skills and knowledge.</li> <li>- Reduced software redundancy.</li> <li>- Easy access to innovative tools.</li> <li>- Better understanding of code structure.</li> <li>- Faster development.</li> </ul>	✓	✓	
<b>Maintainers</b>	<ul style="list-style-type: none"> <li>- Acquiring new skills and knowledge.</li> <li>- Data is undergoing tests by a diverse pool of users, leading to faster identification and rectification of bugs.</li> <li>- Reduced reliance on outdated sources.</li> <li>- Greater pool of automated test cases.</li> <li>- Reduction of vulnerable code</li> </ul>	✓	✓	✓
<b>Business owners</b>	<ul style="list-style-type: none"> <li>- Businesses strengthen their innovative capabilities.</li> <li>- Better informed employers.</li> <li>- Greater pool of users [46].</li> </ul>	✓		
<b>Research’s Funders</b>	<ul style="list-style-type: none"> <li>- Better financial return from research investment.</li> <li>- Increased reputation.</li> <li>- Building network.</li> <li>- Invitations to collaborate.</li> <li>- Invitations to provide consultancy.</li> <li>- New opportunities for funds.</li> </ul>		✓	✓
<b>Publishers</b>	<ul style="list-style-type: none"> <li>- Independent verification and qualification of research</li> <li>- Increased reputation can elevate the publisher’s impact</li> <li>- Related research is likely to gain attention</li> </ul>			✓
<b>Affiliations</b>	<ul style="list-style-type: none"> <li>- Independent verification and qualification of research.</li> <li>- Increased reputation.</li> <li>- Higher impact.</li> <li>- Attracting better funding opportunities.</li> </ul>		✓	✓
<b>Research Participants</b>	<ul style="list-style-type: none"> <li>- Access to data</li> <li>- Better control over shared content.</li> <li>- Increased Trust.</li> </ul>			✓
<b>Public</b>	<ul style="list-style-type: none"> <li>- Access to knowledge.</li> <li>- Greater Support.</li> <li>- Easier access to resources.</li> <li>- Better understanding.</li> <li>- Reducing communication barrier.</li> <li>- Increased Trust.</li> </ul>	✓		

A checkmark (✓) indicates the associated deposit option.

#### 4. What Type of Software Engineering Research Data Should Be Shared?

The golden rule of data sharing is that “data should be as open as possible and as closed as necessary”. This principle guides engineers to balance openness with necessary restrictions. As open science and FAIR principles gain more attention across all domains, the existing literature thoroughly explains each principle, suggesting changes and improvements. These discussions focus on the ways data is produced and the kinds of relationships that are required across different domains [47–49]. In terms of Software Engineering Research, the FAIR requirements can be described as follows:

- **Findability:** Software Research Data and its associated metadata are easy for both humans and machines to find.
- **Accessibility:** Software Research Data should be understandable and retrievable through standardized protocols, with the original research purpose clearly defined.
- **Interoperability:** Software Research Data should be executable and have the ability to seamlessly be integrated into other software artifacts.
- **Reusability:** Software Research Data should be both executable and modifiable, ensuring ongoing relevance.

According to the FAIR principles, to promote open science, engineers must ensure that their data are classified into all four categories. In the case of software research, the term “data” includes all the artifacts that are used, produced, or might be related to the research process in one or more stages of the research life cycle. The life cycle of research software comprises all essential development stages, from the idea and conception through the development, use, and maintenance, to the archiving and decommissioning. As a result, different types of artifacts (documents, code, test results, etc.) are produced at various stages. In addition to artifacts, all the processing steps of the research data—as the basis of scientific publications—must be available [48]. However, from our experience, sharing everything is not possible throughout all development phases, and several times it is unnecessary.

Effective data sharing can be determined by two factors: (a) the information research engineers are able to share due to either the state of data or disclosure constraints, and (b) the reason or expected end use for the research data [50]. If researchers’ primary concern is reputation and receiving academic credit for their software research, sharing citation metadata becomes their focus. If they want part of their analysis to be replicated, they may worry more about sharing efficient metadata regarding dependencies, Software metrics (e.g., quality metrics like code coverage, etc.), and Development metrics (e.g., pertaining to issues, pull requests, etc.) than authors and titles. If their main focus is for their research to be recreated and their software to be reused, then they need to share the whole range of software research data. Following this categorization, Software researchers need to not only preprocess data into a universally readable format but also choose which metadata needs to be shared for each case [51].

Unlike conventional data, which are static, software development data is unique in that they are static during development and dynamic throughout their lifetime [20,47]. This dual nature of data presents significant challenges in the sharing of research data in software engineering, with each type of artifact requiring specialized handling [52]. Inspired by the research of Druskat et al. [53] where they described metadata as software-specific and general metadata, depending on the scope of use, we categorized the Software research products as follows:

- **Static metadata:** Primarily includes data produced during the planning, designing, and maintaining phases of software development. This type of metadata provides general information necessary to recreate the described system, without including the executables or runtime results.
- **Dynamic-Runtime metadata:** Includes metadata that can be executed to reproduce either the entire system or the research outcomes. These data mostly come from the development, implementation, and testing phases, where the system executes in

real or simulated environments. These types may also include actions focused on reproduction, replication, verification, validation, repeatability, and/or utility [48].

With this categorization in mind and the demand for FAIR data, we have divided the software research artifacts into the corresponding categories in Table 3.

**Table 3.** Static and Dynamic Metadata, with associated Use Cases and Deposit options.

Artifact (What)	Use Case (Why)								Where					
	Increasing FAIR Principles	Static/Dynamic Runtime	Replication of Research	Contribute to Software	Present Funding Results	Use Data for New Project	Evaluate Contribution	Store Software Entry	Informal Sharing among Peers	Formal Sharing to Consortium	Funder's Website	Discipline Specific Repositories	Generalist Data Repositories	(Data) Journals as Part of Research Paper
<b>Phase 1: Project Initiation</b>														
Project Plan	F	S	✓		✓				✓	✓	✓	✓		
<b>Phase 2: Analysis and Detailed Planning</b>														
Business Requirement Documentation (BRD)	I	S	✓	✓	✓		✓				✓	✓		✓
Software Requirement Specifications (SRS)	I	S	✓	✓	✓		✓				✓	✓		✓
User Requirement Specifications	I	S	✓	✓	✓		✓				✓	✓		
Configuration Management (CM) Plan	I	S	✓	✓	✓		✓				✓	✓		
<b>Phase 3: Design</b>														
Detailed Design Specifications (DDS)	A	DR	✓	✓		✓	✓		✓	✓		✓		✓
<b>Phase 4: Software Construction</b>														
Unit Code	R	DR	✓	✓	✓	✓	✓	✓				✓	✓	✓
Software packages Artifacts	R	DR	✓	✓	✓	✓	✓	✓			✓		✓	✓
<b>Phase 5: Testing</b>														
Code Inspection	I	DR	✓			✓	✓		✓			✓	✓	✓
Test Summary Report	I	S	✓			✓	✓		✓	✓		✓	✓	✓
User Acceptance	F	S	✓		✓	✓	✓		✓		✓			
Risk Management	I	S	✓	✓		✓		✓	✓					
<b>Phase 6: Implementation</b>														
Production Environment	F	DR			✓	✓			✓	✓				
Live System	R	DR	✓	✓	✓	✓	✓	✓		✓			✓	✓
<b>Phase 7: Maintenance</b>														
Dependencies	I	S	✓			✓		✓	✓			✓	✓	✓
Tech debt	I	S	✓			✓			✓					
Maintenance Guide	R	S	✓	✓	✓	✓			✓		✓	✓	✓	✓
Customer's Review	F	S			✓		✓		✓					✓
Software metrics	A	S	✓		✓	✓	✓		✓		✓			✓
Development metrics	A	S	✓		✓	✓	✓		✓		✓			
Usage metrics	A	S	✓		✓	✓	✓		✓		✓			✓

Explanation: F = Findability, A = Accessibility, I = Interoperability, R = Reusability. S = Static, DR = Dynamic Runtime. A checkmark (✓) indicates that the specific artifact applies to the associated use case or deposit option.

While this representation provides a clear understanding of which artifacts should be publicly available and under what circumstances, the question of what should be shared within the landscape of the FAIR principles arises. In his research, Katz [20] provides a summary of software as increasingly FAIR research objects. Based on Katz's suggestion, we tried to develop a roadmap for researchers.

It might seem straightforward to suggest that static data should always be accessible and dynamic data reusable upon completion; however, this generalization can be misled-

ing. We have highlighted in Table 3 the artifacts as we consider they fall in the increasingly FAIR principles.

The first step to software research data becoming findable is for the researchers to make basic information available; this includes the name of the software, persistent identifiers, funding plan descriptions, customer reviews, and usage metrics. Moreover, publishing the project plan through many channels can increase awareness and lead to greater discoverability, boosting the researcher's reputation.

The second step towards openness is ensuring that both the software and its metadata are available without financial, legal, or technical barriers. To further improve accessibility, artifacts such as Detailed Design Specifications (DDS) and measurements on software quality and usage must be published. These artifacts provide users with access to the research's key findings, which they may analyze or utilize as a foundation for their work. Licensing of data by researchers at this stage needs to be done clearly so that users understand the citation requirements for different artifacts. Properly given licensing increases trust in data, facilitating its transparent use. Researchers should employ standardized access protocols, such as RESTful APIs or FTP, in order to ensure that data retrieval can be done effectively by users across systems and locations.

Integration with metadata should be seamless, and all information needed to execute software provided to the other researchers. A list of dependencies and requirements on software should be available in a formal, machine-readable format. Users at this stage of development should be able to test the system with supplied test units along with expected results to ensure that it conforms with the initial plan of the project. The opening of the configuration and risk plan will increase trust in research.

The final step of reusability includes not only the open share of the source code and live system, but also the maintenance plan to ensure the longevity of the system. By allowing others to replicate the full software development process, building upon it, and adapting it for diverse applications, the impact of research grows within the scientific community.

There are many compelling reasons for making all the Software Research Artifacts publicly available, yet there are cases where artifacts are no longer available, or the data may be subject to high privacy requirements that prevent public sharing. In such situations, static metadata can serve as surrogates for the described objects. Researchers should therefore ensure that the minimum metadata necessary for project preservation is publicly accessible. Key metadata includes:

- Software name.
- Programming language.
- Version information.
- File system metadata (e.g., file sizes, number of files, etc.).
- All types of Licenses applied to any material (e.g., text, image, multimedia, software) those licenses can be GPL, MIT, LGPL (GNU Lesser Public License), CC creative etc. [54].
- Identifiers refers to unique, persistent, and machine-actionable identifiers (e.g., DOI, ARK, or PURL).
- All Locations where metadata is stored.
- Categorization information (e.g., application category, keywords, etc.).
- Availability information (e.g., release dates, embargo, etc.).
- Relational metadata (e.g., software is part of another work).
- High-level description (e.g., abstract, README, or other text description).
- Dependency information.
- References to work the software is built on or relates to.
- Software metrics (e.g., quality metrics like code coverage, etc.).
- Development metrics (e.g., pertaining to issues, pull requests, etc.).
- Usage metrics (e.g., downloads, stars, citations, etc.).
- Funding information.

These types of metadata will not provide an executable, but rather a general idea of what and how to build a similar system. Cases with a low tolerance for sharing Software research results highlight the need for proper metadata, which is crucial for detailing the use and context of each artifact.

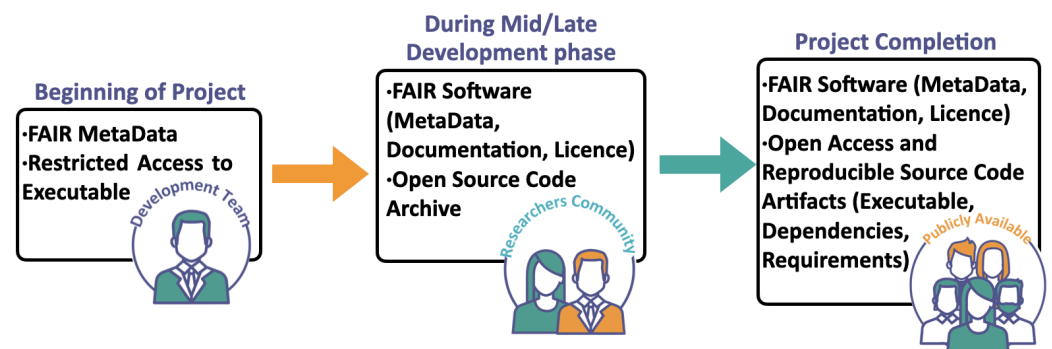
### 5. When Is the Right Time to Share Software Engineering Research Data?

One of the most complex questions related to open science is the timing of data exchange and research output, as research creators need to balance the possible gain (expressed by the value function) against the potential risks of delivering (too) early. Typically, Software Engineers tend to share their results at the end of a project's life-cycle, risking alternatives approaches being published and accepted before theirs. Studies indicate that the level of experience influence the decision of sharing, as more experienced researchers tend to share their data more frequently [55], while non-experienced researcher are waiting for the momentum to deliver discoveries.

Releasing data at the later stages of research offers advantages, particularly in terms of the quality of the final product. However, there are also some serious disadvantages: earlier stages of research remain untested and unreviewed by third-party practitioners. From our experience, the risks of keeping a project completely private until its final stages are greater than the risks of publishing a not-yet-perfect version. In the former scenario, researchers risk having to completely restart development, if the result does not suit the needs of the clients or falls short comparing state-of-the-art competing solutions. On the other hand, sharing early a version allows developers to address issues and obtain market feedback to further refine their solution.

Throughout our research careers, we have learned that a structured approach to data sharing is the most effective strategy. It provides researchers with a sense of security by not exposing the entirety of their work prematurely, while still allowing them to receive valuable feedback and make iterative improvements.

To address the benefits of a scheduled opening process, following the instructions of Katz et al. [10], we propose a minimum three-phase process. This approach, as presented in Figure 3 ensures systematic data sharing, providing engineers with a structured detailed plan to follow in each development phase.



**Figure 3.** Timeline of Opening Software Research Engineering Data.

**Beginning of the Project:** At the beginning of a project, the developers should declare a project repository that will include all resources, intermediary files, supporting documents, and programming scripts used to process the data. This repository will ensure that (a) no work is lost and (b) all materials are accessible from the onset. In this stage, engineers should provide the audience with detailed documentation describing the objectives, methodologies, and anticipated outcomes of the project. Besides general purpose data, in this phase, Software Engineers should include all documents related to system requirements, system architecture, use case scenarios, and the detailed development plan, that way more sophisticated users will have access to all details needed to follow the same procedure and rebuild the system. Opening the data early on reassures users of the trustworthiness of



the developer behind the project. Additionally, it offers a great opportunity for feedback exchange between users and developers, ensuring that any change in the initial design will directly affect the end product.

**Midpoint of the Project:** Engineers are ready to deliver the generated Software Research Data alongside complementary documentation, including operating source code and the instructions necessary for reusing it, at the halfway point of a project's lifespan. At this stage, the data should include complete documentation, extensive metadata, and a clear license in accordance with the FAIR principles (Findable, Accessible, Interoperable, and Reusable). Following best practices in software engineering, researchers must also utilize licenses, version control, coding standards, code documentation, and testing. As an alternative, research software engineers may offer domain scientists expert assistance, guaranteeing compliance with recommended procedures and advancing the program's sustainability.

**Completion of the Project:** Upon completion, all data created and produced during the project's life cycle should be thoroughly documented and shared. In this final stage, the data should be mature, and detailed reasoning for the research results can be provided. Comprehensive documentation should include final reports, detailed methodologies, full datasets, analysis scripts, and any other relevant artifacts.

While this process may require more effort, it maximizes the research's impact and relevance, since results are immediately available to the audience and there is no delay or potential risk of the data becoming no longer representative of the research context.

## 6. Where Should Software Engineers Deposit Their Software Engineering Research Data?

Digital preservation is an umbrella term for a variety of services with a single goal: maintaining data so that it may be reused in the future. Beyond the duration of a grant or research project, researchers must guarantee that their data is preserved for future generations [56]. There are a plethora of options regarding sharing software research data, from informal sharing among peers and hosting on project or institutional websites to depositing in specialized data centers and submitting to data journals for publication. The decision of where to place data highly relies on the means of support provided by each repository. The completeness of metadata records and recommended formats vary across repositories [57]. Depositing data in open archives is generally viewed as the gold standard, as data becomes more discoverable, linkable, and citable [36]. However, as more developers share their data in generalist repositories, discoverability decreases, with users describing the process of finding the desired data as "going down a rabbit hole" [9].

To overcome this added discoverability challenge, when choosing the best option for publishing data, researchers should consider five factors: (a) the availability of supplementary data; (b) the format of the data, and (c) the desired audience, (d) the phase of the project's life cycle and (e) the maintenance plan. Those requirements can be translated in terms of principles to TRUST (Transparency, Responsibility, User Focus, Sustainability, and Technology).

TRUST principles referring to transparency, responsibility, user focus, sustainability, and technology act complementary to FAIR. When considering options for sharing data, researchers should choose the best option for the data to be trustworthy.

- **Transparency:** To be transparent about specific repository services and data holdings that are verifiable by publicly accessible evidence. When choosing the method of deposition, researchers need to make sure that there are no limitations or restrictions regarding the use of the deposit data. Being able to easily assess whether a repository can handle sensitive data responsibly would also inform their decision on whether to utilize the available data services.
- **Responsibility:** To be responsible for ensuring the authenticity and integrity of data holdings and for the reliability and persistence of its service. Before sharing your data, you need to make sure that it aligns with the practice restrictions of the means of sharing. In several cases, the verification of the data and metadata integrity might

require time. Both depositors and users must have confidence that the data will remain accessible over time and thus can be cited and referenced in scholarly publications.

- **User Focus:** To ensure that the data management norms and expectations of target user communities are met. Before sharing data, researchers need to address the project's target group. It might need to monitor and identify the expectations of users before depositing data. Moreover, it is important to remember that different types of users reach data from different routes [9].
- **Sustainability:** To sustain services and preserve data holdings for the long term. The software's lifetime can be long for well-maintained projects, or end quickly if the task it was supposed to do is not needed anymore or if another software does it in a better way. While the software is easily replaceable, metadata, on the other hand, is an important digital artifact that should be preserved [58] along with datasets in order to properly verify or reproduce [59] published findings. Metadata might become less interesting with time, but it cannot be replaced as it is connected to one particular experiment at that particular time.
- **Technology:** To provide infrastructure and capabilities to support secure, persistent, and reliable services. Without active maintenance, data deteriorates and disappears at an alarming rate, making digital data far more disposable than traditional information sources [56].

Generalist repositories such as GitHub, Dataverse, Dryad, and Zenodo, where researchers from any field can store and share their data and metadata without the need for extensive pre-editing or reconstruction, are the most popular among the options. One of the main advantages of generalist repositories is that they enable a combination of discovery of digital objects through their metadata and direct access to the object artifacts themselves [53]. GitHub is one of the largest code repositories to date [60], with a wide diversity in documentation maturity, software purpose, and programming languages. Dryad is designed to preserve the underlying data reported in a paper at the time of publication, when there is the greatest incentive and the ability for authors to share their data, while Dataverse is an example of an initiative offering data management support throughout the research life cycle [37]. Zenodo was seen as an easy-to-use solution, open to all, and provided by a recognized organization. However, it only serves as the final destination for the data, as data cannot be removed, and it is always public, at least after an embargo period [61]. Commercial generalist repositories such as Figshare, on the other hand, provide immediate access but not necessarily long-term sustainability.

While generalist repositories can adequately fulfill the needs of small to medium-sized projects, projects without sensitive data, university projects, or projects that do not require significant computational power, more advanced projects necessitate the use of discipline- or subject-specific data repositories. For research software, it is crucial to ensure that it is both archived for reproducibility and actively maintained for reusability [62]. Depositing scientific software in archives like Software Heritage or Research Software Directory establishes long-lasting, resilient links between research data and the software. Those repositories are specifically designed for software research, and so they store not only a software artifact but also its full development history, requiring a structured workflow, as those repositories have formatting standards [63].

Another option that has been gaining popularity over the past few years is the distribution through data journals. Although the number of data journals has been increasing over the past few years, many researchers are still unaware of them. Among the most notable data journals in terms of volume are Data, Data in Brief, and Scientific Data [64]. Data journals primarily publish articles that describe the details surrounding datasets, such as the reasoning behind their collection, the methodology used, and the timing involved. These articles often highlight the potential future value of the data for analysis, rather than presenting current findings or conclusions. From the developer's point of view, writing a software article can involve a great deal of extra work. The mindful software research process includes documentation for both users and developers that is sufficient to make it

understandable [20,51]. Different journals implement diverse approaches to the management of data, causing great heterogeneity in the format of the metadata associated with the components, each reflecting the strategies of the corresponding managing entity and the technical solutions adopted.

Lastly, another possible yet not very common solution that combines static and dynamic metadata to provide a simplistic version of the software, is the use of a runnable sample deposit (i.e., GitHub). Researchers after agreeing with peers can share a black box demonstration of the system, alongside some sample data and their expected outcome. A runnable deposit should allow another researcher to build, install, configure, and run software on the sample data.

After conducting extensive research on open data organizations and reviewing studies in which engineers were asked about their preferred platforms for data sharing, we created the Table 4 This table provides a comprehensive overview of repositories that may be able to meet the diverse needs of engineers, drawing on related work and our own experiences.

**Table 4.** Where, What and When to distribute Software Engineering Research Data.

	Name (Where)	Link	Supported Artifacts (What)	When		
Access	Name (Where)		Supported Artifacts (What)	Beginning of Project	Midpoint of Project	Completion of Project
	<b>Generalist Data Repository</b>					
Open Access	Kaggle	<a href="https://www.kaggle.com/">https://www.kaggle.com/</a> (accessed on 15 September 2024)	Datasets, Notebooks (Jupyter notebooks, R or Python scripts), pre-trained ML models	✓	✓	✓
	GitHub	<a href="https://github.com/">https://github.com/</a> (accessed on 15 September 2024)	Static and Dynamic Metadata, raw Source code, test cases, and documents in any format	✓	✓	✓
	EUDAT	<a href="https://www.eudat.eu/">https://www.eudat.eu/</a> (accessed on 15 September 2024)	Dynamic Metadata, Software, workflows		✓	✓
	Zenodo	<a href="https://zenodo.org/">https://zenodo.org/</a> (accessed on 15 September 2024)	Static and Dynamic Metadata, Software, Pre-publications		✓	✓
	ISBSG	<a href="https://www.isbsg.org/about-isbsg/">https://www.isbsg.org/about-isbsg/</a> (accessed on 15 September 2024)	Software projects with documentation			
	OSF	<a href="https://osf.io/">https://osf.io/</a>	Files, research data, code, protocols		✓	✓
	OLOS	<a href="https://access.olos.swiss/portal/#/home">https://access.olos.swiss/portal/#/home</a> (accessed on 15 September 2024)	Research data, Static Metadata, Source code		✓	✓
	SIR	<a href="https://sir.csc.ncsu.edu/portal/index.php">https://sir.csc.ncsu.edu/portal/index.php</a> (accessed on 15 September 2024)	Software systems, artifacts, test suites, scripts		✓	✓
	Dataverse	<a href="https://dataverse.org/">https://dataverse.org/</a> (accessed on 15 September 2024)	Static and Dynamic Metadata, Software, Research data, Publications			✓
	SourceForge	<a href="https://sourceforge.net/">https://sourceforge.net/</a> (accessed on 15 September 2024)	Static Metadata, Open source and paid software projects			✓
Proprietary	Dryad	<a href="https://datadryad.org/stash">https://datadryad.org/stash</a> (accessed on 15 September 2024)	Any form of Research Files, including compressed archives		✓	✓
	Figshare	<a href="https://figshare.com/">https://figshare.com/</a> (accessed on 15 September 2024)	Any form of Static or multimedia Research Output, excluding the source code		✓	✓
<b>Discipline-/Subject-Specific Data Repository</b>						
Open Access	PROMISE	<a href="http://promise.site.uottawa.ca/SERepository/">http://promise.site.uottawa.ca/SERepository/</a> (accessed on 15 September 2024)	Datasets and tools to serve researchers in building predictive software models		✓	✓
	Software Heritage Dataset	<a href="https://docs.softwareheritage.org/index.html">https://docs.softwareheritage.org/index.html</a> (accessed on 15 September 2024)	Source code from software projects and development forges		✓	✓

Table 4. Cont.

	Name (Where)	Link	Supported Artifacts (What)	When		
Access	Name (Where)		Supported Artifacts (What)	Beginning of Project	Midpoint of Project	Completion of Project
Only for data use not no contribution	Qualitas Corpus	<a href="http://www.qualitascorpus.com/">http://www.qualitascorpus.com/</a> (accessed on 15 September 2024)	Collection of software systems intended for empirical studies of code artifacts	✓	✓	✓
	Bug Prediction Dataset	<a href="https://bug.inf.usi.ch/index.php">https://bug.inf.usi.ch/index.php</a> (accessed on 15 September 2024)	Collection of models and metrics of software systems and their histories	✓	✓	✓
	Ultimate Debian Database (UDD)	<a href="http://udd.debian.org/">http://udd.debian.org/</a> (accessed on 15 September 2024)	Aspects of Debian in the same SQL database	✓	✓	✓
	CiteSeerx	<a href="https://csxstatic.ist.psu.edu/home">https://csxstatic.ist.psu.edu/home</a> (accessed on 15 September 2024)	Digital library, includes scientific papers, algorithms, data, metadata, services, techniques, and software	✓	✓	✓
	Software Engineering Data Repository for Research and Education	<a href="http://analytics.jpn.org/SEdata/">http://analytics.jpn.org/SEdata/</a> (accessed on 15 September 2024)	Datasets and tools to serve researchers in building predictive software models	✓	✓	✓
<b>Data Journals</b>						
Open Access	Data in Brief	<a href="https://www.sciencedirect.com/journal/data-in-brief">https://www.sciencedirect.com/journal/data-in-brief</a> (accessed on 15 September 2024)	Short articles, Static and Dynamic Metadata, Software	✓	✓	✓
	SoftwareX	<a href="https://www.sciencedirect.com/journal/softwarex">https://www.sciencedirect.com/journal/softwarex</a> (accessed on 15 September 2024)	Scientific paper, Software	✓	✓	✓
	Data	<a href="https://www.mdpi.com/journal/data">https://www.mdpi.com/journal/data</a> (accessed on 15 September 2024)	Static and Dynamic Metadata, Scientific paper	✓	✓	✓
	Journal of Open Research Software (JORS)	<a href="https://openresearchsoftware.metajnl.com/">https://openresearchsoftware.metajnl.com/</a> (accessed on 15 September 2024)	Software Metapapers, Research Software		✓	✓
	Journal of Open Source Software (JOSS)	<a href="https://joss.theoj.org/">https://joss.theoj.org/</a> (accessed on 15 September 2024)	Scientific papers, research software packages		✓	✓
	Scientific Data	<a href="https://www.nature.com/sdata/">https://www.nature.com/sdata/</a> (accessed on 15 September 2024)	Short articles, Static Metadata		✓	✓
	Software Impacts	<a href="https://www.sciencedirect.com/journal/software-impacts">https://www.sciencedirect.com/journal/software-impacts</a> (accessed on 15 September 2024)	Static and Dynamic Metadata, Software, Short description (Software research Impact, Use cases)			✓
SIAM Journal on Scientific Computing (SISC)	<a href="https://epubs.siam.org/journal/sisc/">https://epubs.siam.org/journal/sisc/</a> (accessed on 15 September 2024)	Scientific papers, documented Research Impact and Use cases			✓	

## 7. How Should Software Engineers Share the Software Engineering Research Data?

To effectively manage and share software research data, it is essential to develop a realistic Holistic Research Management Plan (HRMP) that integrates both a Data Management Plan (DMP) and a Software Management Plan (SMP) [65]. The former outlines how data will be captured, managed, stored, shared, and published, and how the integrity, security, and confidentiality of data will be maintained, during and after the research project. On the other hand, the SMP focuses on the software aspect of a research project, formalizing a set of structures and objectives to ensure the software's reproducibility, reusability, and publicity in the short, medium, and long term [66,67]. Both DMP and SMP are evolving documents that need updates; therefore, new versions of the HRMP should be created whenever important changes to the project occur due to the inclusion of new data [65]. Ideally, the HRMP should be drafted at the beginning of the Software research project. However, even for existing projects, it is important to create an HRMP, as it is a valuable tool to summarize established practices and stimulate reflection and evaluation in software research [50]. Good data management and stewardship is not a goal in itself, but rather a precondition for supporting knowledge discovery and innovation. Research data management requires effort related to navigating legal aspects, presenting data clearly, deciding where to upload, and making the data usable by adding metadata. Currently, a number of groups and organizations, including [ELIXIR](#), [Australian Research Data Commons \(ARDC\)](#), and

Netherlands eScience Center (NLeSC), are proposing structured management approaches that adopt the FAIR4RS Principles. Still, many researchers do not consistently manage their data [50], as they are getting overwhelmed by the number of different requirements they have to fulfill depending on the funders, research domain, or institution. The following list provides a clear understanding of the requirements that cover most of the aspects that research software needs in order to fulfill its purpose:

- **Purpose:** Provide a concise description of the research and the produced software, outlining its purpose and the intended audience. Specify the primary function it serves and who would benefit most from using it.
- **Types of Data Description:** Clearly describe the types and formats of data collected or generated in the project, including any existing data being reused and its origin.
- **Contextual Details (Metadata):** Provide the contextual details and metadata necessary to make the data meaningful to others, ensuring documentation and data quality.
- **Storage, Backup, and Security:** Outline the storage infrastructure, backup procedures, and security measures to protect data against loss, corruption, or unauthorized access.
- **Provisions for Protection/Privacy, Legal, and Ethical Aspects:** Address privacy and legal considerations, including provisions for data protection, compliance with ethical guidelines, and compliance with relevant laws and regulations. While choosing licensing, Engineers need to keep in mind that data and software should be licensed under different licenses [68]. For scientific data content, the most used licenses are Creative Commons licenses. The CC licenses are a good option for works such as articles, books, working papers, and reports while a dedication to the public domain using CC0 is recommended for datasets and databases. The list of possible choices CC licenses, Open Data Commons Licenses, CC0 and CC Public Domain Mark. On the software licensing side the licenses are divided into three groups: permissive licenses (MIT License, Free BSD License, the New BSD License, Apache License 2.0 and Artistic License 2.0), weak copyleft licenses (GNU LGPL, Mozilla Public License 2.0, Eclipse Public License 1.0, Common Development and Distribution License 1.0), strong copyleft licenses (GNU GPL) and an additional group of network copyleft licenses (Affero GPL). The choice of licenses is therefore relatively broad [54]. Engineers need to ensure that it is compatible with the licenses of any libraries or dependencies used.
- **Policies for Reuse:** Define policies for data reuse, including access requirements, sharing agreements, and licensing terms to facilitate responsible data sharing.
- **Policies for Access and Sharing:** Establish policies and procedures for accessing and sharing research data with collaborators, institutions, or the broader scientific community. Explain how users can cite your research in their work. Provide links to citation guidelines that users should follow.
- **Plan for Archiving and Preservation:** Define a plan for the long-term archiving and preservation of data, specifying repositories or archives to deposit the data and ensuring its accessibility over time. Special attention must be paid when a reusable software component nears the end of its life. Discuss partnerships, sponsorships, or funding sources that will help support the software over time. Describe the level of support that will be provided to users of the software. This includes specifying the types of support (e.g., bug fixes, updates, technical assistance) and how it will be organized.

Although we do not claim to be able to present an exhaustive list here, the following provides some useful resources, as a starting point to help Software Engineers to form a high-quality HMP:

- **Argos:** Developed by OpenAIRE to facilitate Research Data Management activities. The service is an open platform designed to manage, validate, monitor, and maintain data management plans (DMPs) and data sets. It allows users to view publicly released DMPs and Datasets, and build DMPs that can be either private or publicly released and traded between infrastructures.



- **DMPRoadmap:** The tool is jointly provided by the [Digital Curation Center \(DCC\)](#), and the [University of California Curation Center \(UC3\)](#). DMPRoadmap assists the collaborative creation and maintenance of different versions of DMPs, provides guidance on management issues, and supports various formats exportation. The service is free at the point of use for researchers to develop DMPs.
- **DMPonline:** DMPonline is a web-based open tool that relies on the DMPRoadmap code-base. The tool offers a number of templates that represent the requirements of different funders and institutions. Users are asked three questions at the outset to determine the appropriate template. Guidance by researcher funders, universities, and disciplines is provided to help researchers interpret and answer the questions.
- **DMPTool:** DMP Tool provides a click-through wizard to create a DMP that complies with the requirements of the funders, with direct links to the funder websites, help text to answer questions and resources for best practices in data management.
- **Data Stewardship Wizard:** Developed by the [Elixir](#) is one of the recommended tools for DMPs by the European Commission. Brings together data stewards and researchers to compose DMPs efficiently. Researchers are guided through composing the DMP, which can then be exported using a selected template and format, including machine-actionable.
- **easyDMP:** EasyDMP has been developed by [Sigma2](#) in collaboration with [EU-DAT2020](#). Simplifies the task of creating data management plans by guiding researchers through a set of easy-to-answer questions. Provides guidelines into a series of easy-to-answer questions containing canned answers.
- **RDMO:** Offering an API, the RDMO supports researchers with the systematic planning, organization, and implementation of data management throughout the course of a research project.
- **DAMAP:** The tool supports managing both data and code along the research data life-cycle by generating DMPs using a guided ten-step questionnaire.

All the aforementioned tools, even though they can support writing both DMPs and SMPs, are focused mostly on DMPs. Besides [Software Management Wizard](#), which extends the capabilities of Data Stewardship Wizard, in the field of life sciences, we are not aware of any SMP-specialized tool. The eScience formulated a [Practical Guide to Software Management Plans](#) while other organizations to cover the gap developed templates including [Research Data Management Organizer \(RDMO\) SMP](#), [Practical guide to SMPs](#), [Template for SMPs](#) and [SOMEF](#). To further support the effort, the Research Data Management Librarian Academy (RDMLA), in partnership with financial support from Elsevier, is offering an [RDMLA](#) curriculum that focuses on the essential knowledge and skills needed to collaborate effectively with researchers on data management.

## 8. Use Case Scenario: Opening Software Research

In this section, we are going to present a scenario of Software Research Data Opening following our guidelines. Through this use case, our aim is to present the process of opening an already completed software research project.

### 8.1. Use Case Description

The scenario is based on our previous work, "Using Code from ChatGPT: Finding Patterns in the Developers' Interaction with ChatGPT" [32]. The research identifies patterns that describe the interaction of developers with ChatGPT, with respect to the characteristics of the prompts and the actual use of the code provided by the developer. For the study, we used the method "Adopt, adapt, develop", which calls for the use and adaptation of publicly available already existing resources.

### 8.2. Description of Research Study

This study investigated how software developers advised ChatGPT for coding tasks, such as bug fixing, feature addition, refactoring, code reuse, and security improvements. Up till the point the research was undertaken, there was little knowledge on how the developers use ChatGPT and whether the responses provided are relevant. The research aimed to highlight the primary concerns developers raised in ChatGPT conversations, with respect to code acquisition, and the context under which the code provided is actually used. This involves analyzing the nature of concerns (programming language prompted, tokens provided) and potential challenges (types of development tasks prompted), along with the developers' code base (GitHub repositories).

The first goal was to identify common concerns that developers presented to ChatGPT through the perspective of the programming languages and development tasks. This analysis uncovered patterns in the prompts, reflecting the community's key needs and areas where ChatGPT could provide assistance.

The second goal examined how developers interacted with ChatGPT and how these interaction styles affected the use of generated code. By exploring whether the developer is usually instructive or adopting a conversational approach to explore whether there is a correlation between styles of interactions and utilization of code, offering recommendations on how developers should tailor their interactions to maximize the effectiveness of ChatGPT's contribution.

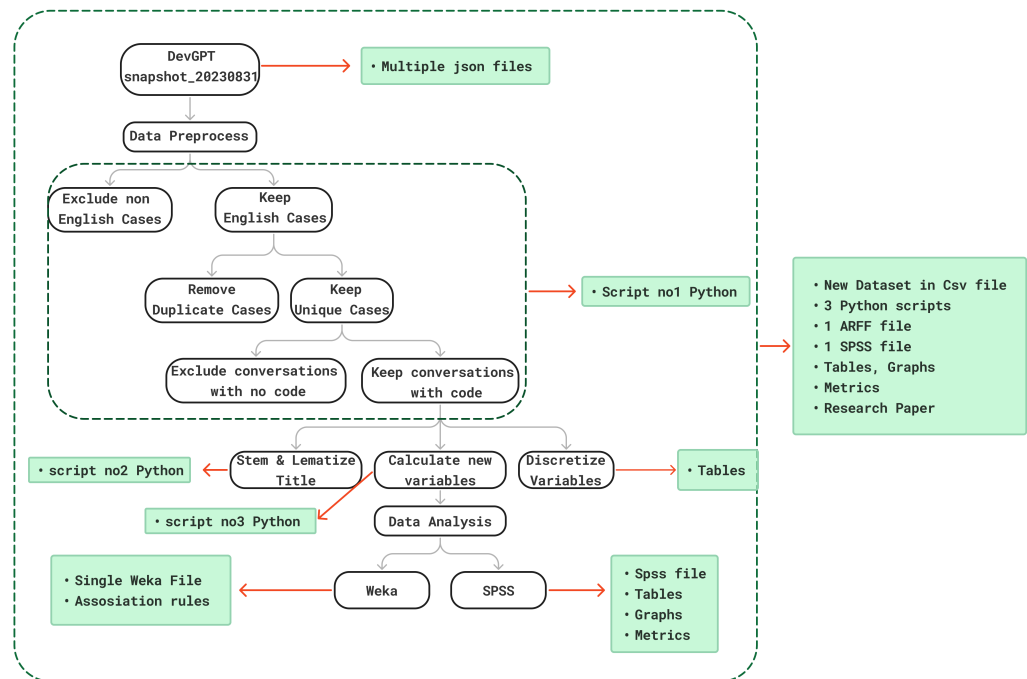
### 8.3. Description of Data

During the study, we have **adopted** the DevGPT [69] dataset from GitHub; it comprises all conversations between developers and ChatGPT—both prompts and responses. The snapshot studied, snapshot\_202308311 [70], contains 2714 conversations between ChatGPT and users. The DevGPT dataset is launched as a collection of six sets of JSON files. Each file includes the URL to the ChatGPT conversation [71], associated HTTP response status codes, access date of the URL, and the HTML response content and a link to the developer's GitHub repository. Additionally, each conversation contains a list of prompts/answers, inclusive of any code snippets. The dataset provided details including the date of the conversation, the count of prompts/answers, their token information, and the model version involved in the chat.

From the enclosed information **adapted** the prompts in an attempt to provide an overview that includes their frequency, characteristics, and possible outcomes. Finally, to extract our results, we had **developed** scripts to isolate repetitive keywords from the titles, that are used to describe the prompts and identify patterns that are indicative of the co-existence of certain tasks and the level of implementation based on this categorization. The analysis provides six different categories of tasks, which are: **BUG FIX**, **ENERGY**, **NEW FEATURE**, **REFACTOR**, **SECURITY** and **OTHER** tasks.

The outlined approach consists of four phases that are (i) data collection, (ii) data preprocessing, (iii) discretization of variables with continuous values, (iv) data analysis, and (v) extraction of results. The process accompanied by the data created in each phase is presented in Figure 4.

Now that we have made a general outline, we aim to describe each step of the opening process with respect to the 5Ws+1H. We revisited the questions to be more specific to the use case.



**Figure 4.** Artifacts created during Research, available for open sharing.

#### 8.4. Why Should We Share Our Software Engineering Research Data?

With the rise of GenAI and the introduction of ChatGPT, more and more individual researchers and organizations are turning their attention to the use of those models as supplementary assistants for their organization units. Currently, ChatGPT has emerged as a game changer in the developer landscape, leveraging the power of Natural Language Processing to generate text that aids in multipurpose tasks. ChatGPT can advise developers on how to fix bugs, add new features, improve already existing code, or even suggest tools and libraries to be reused for a specific purpose [72]. However, there is little knowledge of whether developers trust ChatGPT's responses and actually utilize the provided solutions. This research identifies the primary issues that developers present to ChatGPT and provides an overall overview of the prompts fed to ChatGPT by the developers regarding their frequency and characteristics and their possible consequences. Focusing on prompts, questions, and inherent characteristics of the interaction process patterns that help towards receiving a useful answer from ChatGPT, are noted.

By opening our data, we contribute not only to the broader discussion of AI use in software engineering, we are providing a resource that others can build on. Our data allows researchers to develop better prompt templates, and figure out what works and what does not in AI-assisted coding. Prompt templates and good practices when prompting LLMs could increase the percentages of code utilization. An important aspect highlighted in our research was the originality and ownership related to AI-generated code. During the research, we gathered numerous ChatGPT's answers that were replicated or even copied from the same resource as the user's prompt. Sharing these findings means opening up further ways the community can debate and discuss these ethical challenges to create an innovative, yet responsible, role for AI in development.

#### 8.5. Who Should Be Considered during OUR Process of Opening Software Engineering Research Data?

The discussions about the adoption of Generative AI models in software engineering have focused mainly on potential security risks and the impact on code quality. In contrast, our research highlighted the trends associated with AI-generated code usage and the specific needs of engineers as they interact with models like GPT. The primary audience for our findings includes **software researchers** interested in understanding the dynamics and trends of human-AI collaboration that can be used to develop more effective strategies

for interacting with AI so that better results are achieved. The findings from our study could inspire **businesses and the private sector** to develop software that best meets the specific needs expressed by engineers in interacting with GPT. Moreover, the research highlights a gap in prompt engineering and human-AI collaboration that suggests the need to rethink the way curriculums are delivered within **educational institutions** to better prepare learners for new demands within the software industry. On an individual level, **learners** can gain insight into the kind of competencies and knowledge that will be increasingly demanding within the employment market. Finally, our findings act like a stimulus for further research in this area, setting the stage for opportunities for **financial sponsors** to support continuing investigation and for **publishers** to issue special editions that focus on the interrelationship between artificial intelligence and software engineering.

#### *8.6. What Type of Software Engineering Research Data Should WE Share?*

The primary goal of this research was not only to conduct an experimental study, but also to produce a scientific paper. To ensure the credibility and reproducibility of our work, it's crucial that our paper provides all the necessary information for others to verify and replicate the study. Due to that, all data regarding the dataset, data analysis, and methodology need to be publicly shared. As shown in Figure 4, our study has generated a diverse set of artifacts that support the research process; however, all these artifacts are not suited for publication in their raw form. In line with the categorization in Section 4, we classified the data into two categories: **Static Metadata and Dynamic Runtime Metadata**.

**Static metadata**, crucial for understanding and verifying the study, are being shared as complementary data to the publication. It includes a complex description of methodology, procedures for data analysis, preprocessing actions, and the derivation of results. Those artifacts are well explained and expressed in the different sections of our research paper, acting as supporting data to the publication.

On the other hand, **Dynamic Runtime Metadata** includes assembled dataset, Python scripts and files prepared for Weka and SPSS analyses are being publicly shared with the community as-is in the repository, to support the research reproduction.

By publishing dynamic metadata in their original form, other researchers can easily obtain the necessary resources to recreate the research, while the scientific manuscript, including the static metadata, serves both as a specific how-to guide and source to cross validate their findings.

#### *8.7. When Is the Right Time to Share OUR Software Engineering Research Data?*

For the sharing process, we did use the three-stage process described above.

##### **Beginning of the Project:**

At the beginning of the project, the first and second authors worked independently on local machines for about one month. During that early period, the two preprocessed raw data formed the methodologies and drafted some preliminary results. To guarantee consistency in the approach, both often communicated by exchanging progress updates, providing input on issues encountered, and sharing source code. The goal during this stage was to establish a solid foundation, ensuring that the data was accurate and that the initial results were good enough to justify the next stage.

##### **Midpoint of the Project:**

Once the preliminary results were satisfying, it was then time for team collaboration. At this stage, a private repository was created allowing for structured and centralized data management, making it easier for the entire team to access and contribute to the project. Each member handled different activities within the same database, such as lemmatization, code comparison, data combining, association rule extraction, and statistical analysis. Working in the same instance of the database meant that interrelated tasks kept the updates in real time and allowed for internal data sharing, showing the progress and changes.

### Completion of the Project:

Once everyone was happy with the data, we had to make a decision whether to make the repository publicly accessible immediately or to wait until the associated research paper was published. The team focused on writing the research paper with the inclusion of data as proof of credibility. Once the paper was accepted for publication, the repository was made openly available, accompanied by a preprint of the paper to provide context and scientific backing for the data. Finally, following the publication of the final version, a link to the publisher's website was added to support citations.

#### 8.8. Where Should WE Deposit Their Software Engineering Research Data?

As already described, during our project lifetime we decided to share our research through two generalist repositories, in addition to the publisher's website, where the data complement the published scientific paper. Throughout the entire research, our data were stored in GitHub [73] either in a private or in a public manner, Figure 5 both to keep track of our changes and manage better the parts where collaborative work was required. This is a common practice among researchers as it keeps the data in a safe place assuring there will be no data loss and provides consistency in formatting, as researchers need to follow standards. Later on, having data shared in that manner helps to easily transform the raw information to scientific paper. Furthermore, this approach allows reviewers and readers to validate the data and replicate the research. Although common, this practice plays a crucial role in maintaining transparency and accessibility throughout the research process

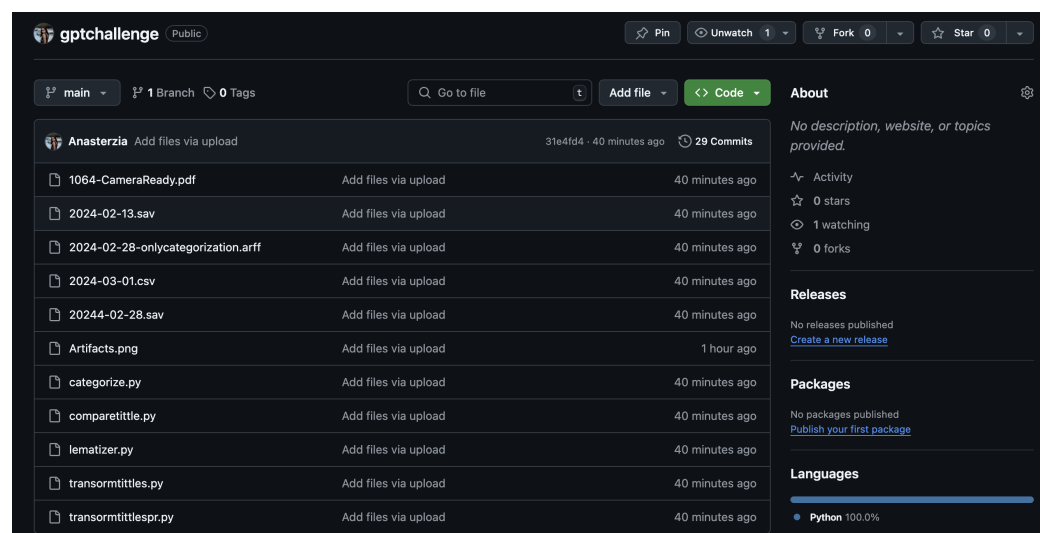



Figure 5. GitHub Repository.

The second repository we have chosen to store our data is the Generalist repository B2SHARE(EUDAT) [74]. Although B2SHARE is a generalist repository, it differs significantly from GitHub in its requirements for metadata management. B2SHARE operates with a structure and policies that are between those of a generalist repository and a discipline-oriented one. This platform emphasizes the importance of including complementary data alongside research results and provides step-by-step guidance on data sharing.

At the end of the project, we had uploaded in EUDAT the same artifacts as found on GitHub. However, as can be seen in Figure 6, the requirements of this platform are strict, and before the data could be made openly available, we had to first ensure that both the quality and security of the research were provided through metadata. The system first requires basic details such as the title, a short description, and the names of the creators of the project. In addition, users can control access to the files, making them openly public or restricting them to the record's owner and community administrators. However, even in these restricted access cases, the metadata remains open, which is useful when, as described in Section 4, not all research data can be shared.



**Community \*** 

**Titles \*** Using code from ChatGPT: Finding patterns in the developers' interaction with ChatGPT

**Descriptions** ChatGPT can advise developers and provide code on how to fix bugs, add new features, refactor, reuse, and secure their code but currently, there is little knowledge about whether the developers trust ChatGPT's responses and actually use the provided code. In this context, this study aims to identify patterns that describe the interaction of developers with ChatGPT with respect to the characteristics of the prompts and the actual use of the provided code by the developer. We performed a case study on 267,098 lines of code provided by ChatGPT related to commits, pull requests, files of code, and discussions between ChatGPT and developers. Our findings show that developers are more likely to integrate the given code snapshot in their code base when they have provided information to ChatGPT through several rounds of brief prompts that include problem-related specific words instead of using large textual or code prompts. Results also highlight the ability of ChatGPT to handle efficiently different types of problems across different programming languages.

**Type \*** Abstract

**Creators**

Terzi Anastasia

Bibi Stamatia

Tsitsimiklis Nikolaos

Angelidis Pantelis

**Open access \***  True

**Embargo date**

**License**

**Disciplines** 4.1.13 → Computer sciences → Software engineering

**Identifier \*** 4.1.13 → Computer sciences → Software engineering

**Keywords** ChatGPT, bugs, features, reusing, developer interaction, association rules, case study, prompts, pattern concepts

**Contact email** a.terzi@uowm.gr

**Publication date**

[Show more details >](#)

This record is already published. Any changes you make will be directly visible to other people.

**Figure 6.** EUDAT Repository.

B2SHARE also offers options on embargo policies, allowing researchers to delay open access to their data. Depending on the type of data, for example, data sets or software, the system offers various licensing options that ensure users interact with the data in a respectful and legal manner. The discipline applicable to the data will determine the appropriate persistent identifier for the project, making the citations accurate.

Once the metadata upload is complete, as shown in Figure 7, the record is ready to be published, respecting all the necessary attributes to become FAIR.

[GO TO EUDAT WEBSITE](#)

[HELP](#)
[COMMUNITIES](#)
[UPLOAD](#)
[CONTACT](#)

RECORDS > DB2EF5890FA44C7A85AF366A50DE73B9

## Using code from ChatGPT: Finding patterns in the developers' interaction with ChatGPT

by Terzi Anastasia; Bibi Stamatia; Tsitsimiklis Nikolaos; Angelidis Pantelis;

Aug 30, 2024

Last updated at Sep 26, 2024

**Abstract:** ChatGPT can advise developers and provide code on how to fix bugs, add new features, refactor, reuse, and secure their code but currently, there is little knowledge about whether the developers trust ChatGPT's responses and actually use the provided code. In this context, this study aims to identify patterns that describe the interaction of developers with ChatGPT with respect to the characteristics of the prompts and the actual use of the provided code by the developer. We performed a case study on 267,098 lines of code provided by ChatGPT related to commits, pull requests, files of code, and discussions between ChatGPT and developers. Our findings show that developers are more likely to integrate the given code snapshot in their code base when they have provided information to ChatGPT through several rounds of brief prompts that include problem-related specific words instead of using large textual or code prompts. Results also highlight the ability of ChatGPT to handle efficiently different types of problems across different programming languages.

**Disciplines:** 4.1.13 → Computer sciences → Software engineering

**Keywords:** ChatGPT, bugs, features, reusing, developer interaction, association rules, case study, prompts, pattern concepts;

**DOI:** [10.23728/b2share.db2ef5890fa44c7a85af366a50de73b9](https://doi.org/10.23728/b2share.db2ef5890fa44c7a85af366a50de73b9)

**PID:** [11304/0afef08b-8fa9-4058-841a-ab984581c75d](https://nbn-resolving.org/urn:nbn:de:hbz:5:1-11304-0afef08b-8fa9-4058-841a-ab984581c75d)

Views

13

---

File Downloads

26

---

Files

11

---

Total Size

3.2 MB

EUDAT

**Share**

**Cite as**

Terzi Anastasia, Bibi Stamatia, Tsitsimiklis Nikolaos, & Angelidis Pantelis. (2024). Using code from ChatGPT: Finding patterns in the developers' interaction with ChatGPT [Data set]. <https://b2share.eudat.eu>. <https://doi.org/10.23728/b2share.db2ef5890fa44c7a85af366a50de73b9>

**Copy BibTeX**

[More citation choices](#)

**Basic metadata**

Open access True ✓

Contact email a.terzi@uowm.gr

**Files**

Name	Size	
1064-CameraReady.pdf <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">8 downloads</span>	784.95 KB	* 🔗
2024-02-13.sav <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">2 downloads</span>	340.61 KB	* 🔗
2024-02-28-onlycategorization.arff <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">2 downloads</span>	335.06 KB	* 🔗
2024-03-01.csv <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">1 download</span>	257.75 KB	* 🔗
20244-02-28.sav <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">1 download</span>	1.22 MB	* 🔗
Artifacts.png	412.67 KB	* 🔗
categorize.py <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">3 downloads</span>	1.84 KB	* 🔗
comparetittle.py <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">3 downloads</span>	1.84 KB	* 🔗
lematizer.py <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">2 downloads</span>	2.27 KB	* 🔗
transormittlespr.py <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">2 downloads</span>	7.32 KB	* 🔗
transormittles.py <span style="font-size: x-small; background-color: #004a99; color: white; padding: 2px;">2 downloads</span>	7.34 KB	* 🔗

Edit metadata
 Create New Version

**Figure 7.** EUDAT Repository.

### 8.9. How Should WE Share the Software Engineering Research Data?

One of the requirements when writing a scientific paper is the structured research framework, which includes detailed description of acts and methods led to resulting research. Even if this plan is not meticulously detailed from the start, software researchers—the authors in this case—should carefully document everything involved in the process to ensure that the study is not only easy to understand but also reproducible by others. Within this context, the paper has to include a Case Study Design section. This section should include all the relevant details required in regard to a Data Management Plan (DMP). Such documentation prevents the need for researchers to redundantly record actions and procedures. While the common elements of this section are well understood, we will highlight the minimum required information as included in our case.

This section through the research questions narrows down the objective of the study and identify its intended audience. The data analysis plan states the various types and formats of data that are being collected or generated during the project. Additionally, this section, along with the results sections, outlines the storage structure, methods for backup, and long-term preservation process for the data. For long-term access, specific repositories or archives proposed for data deposition are specified.

### 8.10. Use Case Conclusion

The following table (Table 5) gives an overview of the opening process followed in the aforementioned use case, regarding the 5W+1H.

**Table 5.** Use Case Data Sharing Timeline: Stakeholders, Purposes, and Platforms.

When	Who	Why	What	Where
Beginning of Project	Authors	Collaboration, Disclosure, Establishing a foundation for data accuracy	Raw Data and Preprocessed Data (original JSON files, Python scripts, data analysis procedures, preprocessing actions)	Natively on machines and in a GitHub private repository
Midpoint of Project	Research Team, Software Researchers	Cross-Validation, Reproducibility, Verification of research, Proofreading, Ensuring real-time collaboration	Processed Data (assembled dataset, Python scripts, Weka files, SPSS files, complete data analysis procedures and methodologies)	GitHub public repository
Completion of Project	Software Researchers, Businesses, Educational Institutions, Learners, Financial Sponsors, Publishers	Knowledge Dissemination, Enhancing transparency, Supporting future research, Informing curriculum development	Final Data (new dataset in CSV file, Python scripts, ARFF file, SPSS file) and Research Paper (tables, graphs, metrics, and analytic research methodology)	Journal and EUDAT repository

## 9. Discussion

Although data sharing seems to have a global benefit, cultural and national factors pose a significant challenge to a one-size-fits-all approach. Regardless of the benefits, deciding what data can be shared, how it should be shared, and making it usable by others requires additional effort, training, and resources.

Open Software Research Data offers benefits highly related to transparency, reproducibility, and greater trust in society. The availability of data increases their reuse by researchers and practitioners, saving time and effort while enhancing the credibility and generalizability of findings through the use of diverse data sets. Even though the idea of open shared knowledge is seen as ideal, and we have already discussed the benefits of openly sharing Software Engineering Research Data, significant challenges persist in fully embracing data openness.

Despite growing demand within the community, guidelines and dedicated ICT tools for data openness are still lacking. Engineers often spend valuable time developing custom data-sharing methods or navigating multiple tools to automate processes and ensure successful data release, both in uploading the data and in the requests for assistance that may follow after the release of the data. Time may also be spent rebutting future re-analysis of the data and defending the original research findings. Data must be presented in an understandable and usable format to ensure maximum transferability between projects, adding additional difficulty to engineers who should create suitable technological infrastructure to support broad international interoperability and enforce robust data quality controls for optimal utilization.

Another challenge that prevents researchers from employing openness in their research is the area of conflict between anonymity and confidentiality on the one hand and openness on the other. The lack of consistent guidelines around Software Research Data sharing makes navigating privacy and ethical concerns difficult, leading practitioners to keep their data within a trusted circle of colleagues, Borgman [36]. Researchers tend to stick with the close culture because of their fear of losing control over their research, possibly through unethical data use (including misinterpretation and misuse), data commercialization, and potential harm to themselves. There is also concern about the level of trust in other researchers' data and the knowledge about who is using the data, since standards for citing another researcher's data are not universally understood. Researchers may intend to share their data, yet this intention often does not translate into actual data-sharing behavior.

Practitioners periodically use open data principles and are not fully committed to the practices due to their perceptions and the work environment [21,75]. This inconsistency creates an environment of unreliability for the research community and the public. Not only does this inconsistent behavior make users skeptical of engineers' intentions but also creates gaps in research data, as several practices remain private or only partially shared, which makes it difficult to analyze and reuse the data.

To overcome all the aforementioned challenges, engineers need to prioritize improving harmonization in interoperability and operational practices. Developing standardized processes would allow engineers to focus on their core competencies, reducing time costs associated with data sharing, and enhancing efficiency. By establishing a consistent practice, it becomes easier to verify and replicate findings, promoting transparency and reliability in scientific output.

By systematically addressing the 5Ws+1H (Why, Who, What, When, Where, and How) framework, we provide a structured approach that aims to simplify the process of data sharing, offering researchers and practitioners clear guidelines on navigating open science blurred boundaries in software engineering. Timing, selection of the right repository, identification of stakeholders, and data management are some other important aspects ignored by most of the current literature, which often addresses the opening process of single software artifacts. Most existing research focuses on the sharing of deployable software artifacts like source code [7–9], testing code [11–14], and supporting documentation, including maintenance guides [11,15], technical specifications [16–18], and quality metrics [19]. These efforts, while valuable, do not address the broader lifecycle of data sharing in software development. Our work fills this gap by providing a clear step-by-step pathway on how data can be shared during a software development life cycle. Unlike previous works that primarily focus on the dissemination of knowledge, our paper indicates clear benefits practical for various stakeholders. This expanded scope makes our work of particular interest to industry practitioners seeking to streamline development workflows, reduce duplication of effort, and increase innovativeness. It also has societal impacts in that the public will better understand, and participate in, software research by having open, usable data to guide policy and drive citizen science. It covers the academic and practical issues, providing the software engineering community with the tools to adopt open science practices in an away that facilitates its needs.

### *9.1. Implications to Researchers And Practitioners*

This chapter describes implications of our research. In this section, first we describe the practical impact of our findings, and then we detail what are the implications of this research to the different stakeholders in the field.

The main research impact is that we provide a guidance for practitioners on the complete process of opening Software Research Data. The study is designed to guide practitioners step-by-step, through the main research concerns, enabling them to achieve the necessary goals for effectively sharing research data in a structured and impactful way.

Regarding the stakeholders, some take away messages are :

For **Research Creators**, can use the guidelines reported in this work to structure their ongoing software development research, ensuring data openness in all levels of the life cycle of software development. The detailed breakdown of the framework into specific stages—such as when, how, and where to share data—ensures that practitioners have a clear, actionable plan tailored to different phases of software development.

The **Software Engineering Community** benefits from this research by gaining insights into available resources and best practices related to open software data. This paper can act as a guideline showing them where to find the relevant research outputs, when to look for them, and what specific research data is most useful to meet their needs.

**End users** will be benefited from adoption of the research guidelines by the creators, as they will have access to well-documented software accompanied by clear explanations and a strong support system. Implementation of our guidelines ensures that end users

receive the information needed in an accessible and understandable format improving their overall experience with the software.

### 9.2. Future Work

As future work, we intend to first evaluate the implementation of these guidelines in wide-ranging cases. We then aim to update our strategy to better fit the needs of researchers. The updated guidelines will be after circulated within the community for feedback and validation. The adoption strategy will include steps on engaging with key stakeholders, such as those who will:

- Support and promote the guidelines: Identifying key organizations and institutions that will actively endorse and promote the guidelines within their networks.
- Provide training on the guidelines: Providing resources for researchers, data managers, and developers to put the principles into practice.
- Use the guidelines: Encouraging researchers and developers to apply these recommendations into their research process.

## 10. Conclusions

In this paper, we analyzed the challenges and opportunities of open data in software engineering using the 5Ws+1H framework. We pointed out that it is important to foster an open, collaborative culture within the software engineering community by specifying clear guidelines for data sharing throughout the entire software development cycle. By identifying key stakeholders, relevant types of data, appropriate timing, and suitable repositories, we aim to facilitate the process of opening Software Engineering Research Data. The study underlines the need to create and share a Holistic Data Plan, which includes a Data Management Plan (DMP) and a Software Management Plan (SMP). The practical applicability of these guidelines was demonstrated on a use case based on our past work by sharing all data created, used, and published in that research highlighting the significant opportunities for further innovation, transparency, and collaboration that open data practices bring. This paper aims to encourage engineers and researchers to embrace open data practices fostering innovation, transparency, and collaboration in the field. We hope these efforts increase the use of open science principles in software engineering in order to build a more open and interconnected research environment.

**Author Contributions:** All authors contributed to the conception and design of the study. Material preparation, data collection, and analysis were performed by A.T. The first draft of the manuscript was written by A.T. S.B. conceptualized the paper and commented on all versions of the manuscript. All authors have read and agreed to the published version of the manuscript.

**Funding:** No funding was obtained for this study.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** The authors have no conflicts of interest to declare that are relevant to the content of this article.

## References

1. Russell, A.L. *Open Standards and the Digital Age: History, Ideology, and Networks*; Cambridge Studies in the Emergence of Global Enterprise; Cambridge University Press: Cambridge, UK, 2014.
2. Critchlow, T.; Kleese, K. *Data-Intensive Science*; CRC Press: Boca Raton, FL, USA, 2016.
3. Woelfle, M.; Olliaro, P.; Todd, M. Open science is a research accelerator. *Nat. Chem.* **2011**, *3*, 745–748. [CrossRef] [PubMed]
4. Open Science Definition | FOSTER. Available online: <https://www.fosteropenscience.eu/foster-taxonomy/open-science-definition> (accessed on 15 September 2024).
5. Budapest Open Access Initiative. Available online: <https://www.budapestopenaccessinitiative.org/read/> (accessed on 15 September 2024).



6. Kelley, A.; Garijo, D. A framework for creating knowledge graphs of scientific software metadata. *Quant. Sci. Stud.* **2021**, *2*, 1423–1446. [[CrossRef](#)]
7. Anzt, H.; Kuehn, E.; Flegar, G. Crediting pull requests to open source research software as an academic contribution. *J. Comput. Sci.* **2021**, *49*, 101278. [[CrossRef](#)]
8. Li, Z.; Yu, Y.; Zhou, M.; Wang, T.; Yin, G.; Lan, L.; Wang, H. Redundancy, context, and preference: An empirical study of duplicate pull requests in OSS projects. *IEEE Trans. Softw. Eng.* **2020**, *48*, 1309–1335. [[CrossRef](#)]
9. Hucka, M.; Graham, M. Software search is not a science, even among scientists: A survey of how scientists and engineers find software. *J. Syst. Softw.* **2018**, *141*, 171–191. [[CrossRef](#)]
10. Katz, D.S.; Gruenpeter, M.; Honeyman, T. Taking a fresh look at FAIR for research software. *Patterns* **2021**, *2*, 100222. [[CrossRef](#)]
11. Ojala, M.; Cohn, M.L. Software Maintenance as Materialization of Common Knowledge. *Engag. Sci. Technol. Soc.* **2023**, *9*, 165–185. [[CrossRef](#)]
12. Zaragozaí, B.M.; Trilles, S.; Navarro-Carrión, J.T. Leveraging Container Technologies in a GIScience Project: A Perspective from Open Reproducible Research. *ISPRS Int. J. Geo-Inf.* **2020**, *9*, 138. [[CrossRef](#)]
13. Herala, A.; Kasurinen, J.; Vanhala, E. Views on Open Data Business from Software Development Companies. *J. Theor. Appl. Electron. Commer. Res.* **2018**, *13*, 91–105. [[CrossRef](#)]
14. Krishnamurthy, S. A managerial overview of open source software. *Bus. Horizons* **2003**, *45*, 47–56. [[CrossRef](#)]
15. Geiger, R.S.; Howard, D.; Irani, L. The Labor of Maintaining and Scaling Free and Open-Source Software Projects. *Proc. ACM Hum.-Comput. Interact.* **2021**, *5*, 175. [[CrossRef](#)]
16. Terzi, A.; Christou, O.; Bibi, S.; Angelidis, P. Software Reuse and Evolution in JavaScript Applications. In Proceedings of the 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain, 31 August–2 September 2022; pp. 263–269. [[CrossRef](#)]
17. Jackson, M. Software Deposit: What to Deposit. 2018. Available online: <https://doi.org/10.5281/zenodo.1327325> (accessed on 15 September 2024).
18. Pashchenko, I.; Plate, H.; Ponta, S.E.; Sabetta, A.; Massacci, F. Vulnerable open source dependencies: Counting those that matter. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, New York, NY, USA, 11–12 October 2018. [[CrossRef](#)]
19. Alarcon, G.M.; Gibson, A.M.; Walter, C.; Gamble, R.F.; Ryan, T.J.; Jessup, S.A.; Boyd, B.E.; Capiola, A. Trust Perceptions of Metadata in Open-Source Software: The Role of Performance and Reputation. *Systems* **2020**, *8*, 28. [[CrossRef](#)]
20. Katz, D.; Niemeyer, K.; Smith, A.; Anderson, W.; Boettiger, C.; Hinsén, K.; Hooft, R.; Hucka, M.; Lee, A.; Löffler, F.; et al. Software vs. data in the context of citation. *PeerJ Prepr.* **2016**, *4*, e2630v1.
21. Tenopir, C.; Allard, S.; Douglass, K.; Aydinoglu, A.U.; Wu, L.; Read, E.; Manoff, M.; Frame, M. Data Sharing by Scientists: Practices and Perceptions. *PLoS ONE* **2011**, *6*, e21101. [[CrossRef](#)]
22. Wnuk, K.; Pfahl, D.; Callele, D.; Karlsson, E.A. How can open source software development help requirements management gain the potential of open innovation: An exploratory study. In Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, New York, NY, USA, 19–20 September 2012; pp. 271–280. [[CrossRef](#)]
23. Ho-Quang, T.; Hebig, R.; Robles, G.; Chaudron, M.R.; Fernandez, M.A. Practices and Perceptions of UML Use in Open Source Projects. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), Buenos Aires, Argentina, 20–28 May 2017; pp. 203–212. [[CrossRef](#)]
24. Hebig, R.; Quang, T.H.; Chaudron, M.R.V.; Robles, G.; Fernandez, M.A. The quest for open source projects that use UML: Mining GitHub. In Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, New York, NY, USA, 2–7 October 2016; pp. 173–183. [[CrossRef](#)]
25. Di Gangi, P.M.; Wasko, M. Steal my idea! Organizational adoption of user innovations from a user innovation community: A case study of Dell IdeaStorm. *Decis. Support Syst.* **2009**, *48*, 303–312. [[CrossRef](#)]
26. Open Data Report | Elsevier. Available online: <https://www.elsevier.com/about/open-science/research-data/open-data-report> (accessed on 15 September 2024).
27. Runeson, P.; Soderberg, E.; Host, M. A conceptual framework and recommendations for open data and artifacts in empirical software engineering. In Proceedings of the 1st IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering, Lisbon, Portugal, 16 August 2024; pp. 68–75.
28. Kipling, R. *Just So Stories*; Macmillan & Co.: London, UK, 1902.
29. Imarah, T.S.; Jaelani, R. ABC Analysis, Forecasting And Economic Order Quantity (Eoq) Implementation to Improve Smooth Operation Process. *Dinasti Int. J. Educ. Manag. Soc. Sci.* **2020**, *1*, 319–325.
30. Hart, G. 'The five W's: An old tool for the new task of task analysis. *Tech. Commun.* **1996**, *43*, 139–145.
31. Abdulkadir, S.; Aliyu, H.O. *ReQueclass: A Framework for Classifying Requirement Elicitation Questions Based on Kipling's Technique and Zachman's Enterprise Framework—A Guide for Software Requirement Engineers*; i-manager Publications: Tamil Nadu, India, 2018.
32. Terzi, A.; Bibi, S.; Tsitsimiklis, N.; Angelidis, P. Using Code from ChatGPT: Finding Patterns in the Developers' Interaction with ChatGPT. In *Proceedings of the International Conference on Software and Software Reuse*; Springer: Cham, Switzerland, 2024; pp. 137–152.
33. Schmidt, B.; Gemeinholzer, B.; Treloar, A. Open Data in Global Environmental Research: The Belmont Forum's Open Data Survey. *PLoS ONE* **2016**, *11*, e0146695. [[CrossRef](#)]

34. Data, S.; Astell, M. Benefits of Open Research Data Infographic. 2017. Available online: <https://doi.org/10.6084/m9.figshare.5179006.v3> (accessed on 15 September 2024).
35. Jackson, M. Software Deposit: Why Deposit Software. 2018. Available online: <https://doi.org/10.5281/zenodo.1327333> (accessed on 15 September 2024).
36. Pasquetto, I.V.; Sands, A.E.; Borgman, C.L. Exploring openness in data and science: What is “open”, to whom, when, and why? *Proc. Proc. Assoc. Inf. Sci. Technol.* **2015**, *52*, 1–2. [[CrossRef](#)]
37. Reilly, S.; Schallier, W.; Schrimpf, S.; Smit, E.; Wilkinson, M. Report on Integration of Data and Publications. 2011. Available online: <https://doi.org/10.5281/zenodo.8307> (accessed on 15 September 2024).
38. Pasquetto, I.; Randles, B.; Borgman, C. On the Reuse of Scientific Data. *Data Sci. J.* **2017**, *16*, 8. [[CrossRef](#)]
39. Sitek, D.; Bertelmann, R. Open Access: A State of the Art. In *Opening Science: The Evolving Guide on How the Internet is Changing Research, Collaboration and Scholarly Publishing*; Bartling, S., Friesike, S., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 139–153. [[CrossRef](#)]
40. Kazman, R.; Goldenson, D.; Monarch, I.; Nichols, W.; Valetto, G. Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project. *IEEE Trans. Softw. Eng.* **2016**, *42*, 220–260. [[CrossRef](#)]
41. Ding, W.; Liang, P.; Tang, A.; Van Vliet, H.; Shahin, M. How Do Open Source Communities Document Software Architecture: An Exploratory Survey. In Proceedings of the 2014 19th International Conference on Engineering of Complex Computer Systems, Tianjin, China, 4–7 August 2014; pp. 136–145. [[CrossRef](#)]
42. Gandhi, R.; Germonprez, M.; Link, G.J. Open Data Standards for Open Source Software Risk Management Routines: An Examination of SPDX. In Proceedings of the 2018 ACM International Conference on Supporting Group Work, New York, NY, USA, 7–10 January 2018; pp. 219–229. [[CrossRef](#)]
43. Open Science: Purpose, Benefits, and What It Means for You. Available online: <https://blog.theopenscholar.com/en/open-science-purposebenefits> (accessed on 15 September 2024).
44. McKiernan, E.; Bourne, P.; Brown, C.T.; Buck, S.; Kenall, A.; Lin, J.; McDougall, D.; Nosek, B.; Ram, K.; Soderberg, C.; et al. How open science helps researchers succeed. *eLife* **2016**, *5*, e16800. [[CrossRef](#)] [[PubMed](#)]
45. Costello, M.J. Motivating Online Publication of Data. *BioScience* **2009**, *59*, 418–427. [[CrossRef](#)]
46. Enders, T.; Satzger, G.; Fassnacht, M.; Wolff, C. Why should I share? Exploring benefits of open data for private sector organizations. In Proceedings of the Pacific Asia Conference on Information Systems, Taipei, Taiwan, 5–9 July 2022; Volume 1.
47. Barker, M.; Chue Hong, N.; Katz, D.; Lamprecht, A.; Martinez-Ortiz, C.; Psomopoulos, F.; Harrow, J.; Castro, L.; Gruenpeter, M.; Martinez, P.; et al. Introducing the FAIR Principles for research software. *Sci. Data* **2022**, *9*, 622. [[CrossRef](#)]
48. Hasselbring, W.; Carr, L.; Hettrick, S.; Packer, H.; Tiropanis, T. From FAIR research data toward FAIR and open research software. *Inf. Technol.* **2020**, *62*, 39–47. [[CrossRef](#)]
49. Gil, Y.; Ratnakar, V.; Garijo, D. OntoSoft: Capturing Scientific Software Metadata. In Proceedings of the 8th International Conference on Knowledge Capture, New York, NY, USA, 7–10 October 2015. [[CrossRef](#)]
50. Martinez-Ortiz, C.; Martinez Lavanchy, P.; Sesink, L.; Olivier, B.G.; Meakin, J.; de Jong, M.; Cruz, M. Practical Guide to Software Management Plans. 2023. Available online: <https://doi.org/10.5281/zenodo.7589725> (accessed on 15 September 2024).
51. Smith, A.M.; Katz, D.S.; Niemeyer, K.E. Software citation principles. *PeerJ Comput. Sci.* **2016**, *2*, e86. [[CrossRef](#)]
52. Lamprecht, A.L.; Garcia, L.; Kuzak, M.; Martinez, C.; Arcila, R.; Martin Del Pico, E.; Dominguez Del Angel, V.; Sandt, S.; Ison, J.; Martinez, P.; et al. Towards FAIR principles for research software. *Data Sci.* **2020**, *3*, 37–59. [[CrossRef](#)]
53. Druskat, S.; Bertuch, O.; Juckeland, G.; Knodel, O.; Schlauch, T. Software publications with rich metadata: State of the art, automated workflows and HERMES concept. *arXiv* **2022**, arXiv:2201.09015.
54. Kapitsaki, G.M.; Tselikas, N.D.; Foukarakis, I.E. An insight into license tools for open source software systems. *J. Syst. Softw.* **2015**, *102*, 72–87. [[CrossRef](#)]
55. Dorta-González, P.; González-Betancor, S.M.; Dorta-González, M.I. To what extent is researchers’ data-sharing motivated by formal mechanisms of recognition and credit? *Scientometrics* **2021**, *126*, 2209–2225. [[CrossRef](#)]
56. Shah, U.A.; Hussain, M.; Saddiqa, M.; Yar, M.S. Problems and Challenges in the Preservation of Digital Contents: An Analytical Study. *Libr. Philos. Pract.* **2021**, *2021*, 5628.
57. Strecker, D. Quantitative Assessment of Metadata Collections of Research Data Repositories. Master’s Thesis, Humboldt-Universität zu Berlin, Philosophische Fakultät, Berlin, Germany, 2021. [[CrossRef](#)]
58. Rollins, N.D.; Barton, C.M.; Bergin, S.; Janssen, M.A.; Lee, A. A computational model library for publishing model documentation and code. *Environ. Model. Softw.* **2014**, *61*, 59–64. [[CrossRef](#)]
59. Peng, R.D. Reproducible research in computational science. *Science* **2011**, *334*, 1226–1227. [[CrossRef](#)] [[PubMed](#)]
60. Gousios, G.; Vasilescu, B.; Serebrenik, A.; Zaidman, A. Lean GHTorrent: GitHub data on demand. In Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India, 31 May–1 June 2014; pp. 384–387.
61. Hyppölä, J.; Essen von, J.; Keskitalo, E.P. Beyond Open Access—Tools and methods for open research. In Proceedings of the AcademicMindTrek’15, Tampere, Finland, 22–24 September 2015; pp. 206–209. [[CrossRef](#)]
62. Hasselbring, W.; Carr, L.; Hettrick, S.; Packer, H.; Tiropanis, T. Open source research software. *Computer* **2020**, *53*, 84–88. [[CrossRef](#)]
63. Di Cosmo, R. Archiving and referencing source code with Software Heritage. In Proceedings of the Mathematical Software–ICMS 2020: 7th International Conference, Braunschweig, Germany, 13–16 July 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 362–373.

64. Walters, W.H. Data journals: Incentivizing data access and documentation within the scholarly communication system. *Insights* **2020**, *33*, 18. [[CrossRef](#)]
65. von Suchdoletz, D.; Brettschneider, P.; Axtmann, A.; Heber, M.; Oberländer, L.; Leendertse, J.; Schumm, I.; Brandt, O.; Schmidt, K.; Gertis, L.; et al. Sicherstellung der Reproduzierbarkeit von Forschungsergebnissen durch Bewahrung des Zugriffs auf Forschungssoftware. *Bausteine Forschungsdatenmanagement* **2023**, *5*, 1–13. [[CrossRef](#)]
66. Chue Hong, N.P.; Crouch, S. What Is a Software Management Plan and How Can It Help Your Project? 2021. Available online: <https://doi.org/10.5281/zenodo.5648418> (accessed on 15 September 2024).
67. Gomez-Diaz, T.; Romier, G. Research Software Management Plan Template, V3.2. Bilingual Document (FR/EN). Available online: <https://hal.science/hal-01802565/document> (accessed on 15 September 2024).
68. Kamocki, P.; Straňák, P.; Sedlák, M. The Public License Selector: Making Open Licensing Easier. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*; Chair, N.C.C., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., et al., Eds.; European Language Resources Association (ELRA): Paris, France, 2016.
69. Xiao, T.; Treude, C.; Hata, H.; Matsumoto, K. Devgpt: Studying developer-chatgpt conversations. In *Proceedings of the 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*; IEEE: New York, NY, USA, 2024; pp. 227–230.
70. NAIST. Available online: <https://github.com/NAIST> (accessed on 15 September 2024).
71. ChatGPT Shared Links FAQ | OpenAI Help Center. Available online: <https://help.openai.com/en/articles/7925741-chatgpt-shared-links-faq> (accessed on 15 September 2024).
72. White, J.; Hays, S.; Fu, Q.; Spencer-Smith, J.; Schmidt, D.C. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In *Generative AI for Effective Software Development*; Springer: Berlin/Heidelberg, Germany, 2024; pp. 71–108.
73. GitHub-Anasterzia/gptchallenge. Available online: <https://github.com/Anasterzia/gptchallenge> (accessed on 15 September 2024).
74. B2SHARE. Available online: <https://b2share.eudat.eu/records/db2ef5890fa44c7a85af366a50de73b9> (accessed on 15 September 2024).
75. Meijer, I.; Costas, R.; Zahedi, Z.; Wouters, P. *The Value of Research Data—Metrics for Datasets from a Cultural and Technical Point of View. A Knowledge Exchange Report*; Knowledge Exchange: Copenhagen, Denmark, 2016.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.