The Origin and Opportunities of Developers' Perceived Code Accountability in Open Source AI Software Development

Sebastian Clemens Bartsch[1], Moritz Lother[1], Jan-Hendrik Schmidt[1], Martin Adam[2], Alexander Benlian[1]

[1]Technical University of Darmstadt
[2]Georg-August-University of Göttingen

bartsch@ise.tu-darmstadt.de, moritz.lother@stud.tu-darmstadt.de, schmidt@ise.tu-darmstadt.de, martin.adam@uni-goettingen.de, benlian@ise.tu-darmstadt.de

Abstract

Open source (OS) software projects in artificial intelligence (AI), such as TensorFlow and scikit-learn, depend on developers' continuous, voluntary code contributions. However, recent security incidents highlighted substantial risks in such software, requiring examinations of factors motivating developers to continuously contribute high-quality code (i.e., providing secure and reliable code fulfilling its functions). Prior research suggests code accountability (i.e., requirements to explain and justify contributed code) to improve code quality, enforced through external accountability mechanisms such as sanctions and rewards. However, the OS domain often lacks such mechanisms, questioning whether and how code accountability arises in this domain and how it affects code contributions. To address these questions, we conducted 26 semi-structured interviews with developers contributing to OS AI software projects. Our findings reveal that despite the absence of external accountability mechanisms, system-, project-, and individual-related factors evoke developers' perceived code accountability. Notably, we discovered a trade-off as high perceived code accountability is associated with higher code quality but discourages developers from participating in OS AI software projects. Overall, this study contributes to understanding the nuanced roles of perceived code accountability in continuously contributing high-quality code without external accountability mechanisms and highlights the complex trade-offs developers face in OS AI software projects.

Introduction

OS software is indispensable for organizations implementing socio-technical systems (e.g., Bonaccorsi and Rossi 2003; Rolandsson, Bergquist, and Ljungberg 2011), offering significant benefits such as cost and time savings due to free access to continually evolving software with new features (Fitzgerald 2006; Goggins, Lumbard, and Germonprez 2021; Wolter et al. 2023). Highlighting its importance, a study provided by Bitkom reports that over 75% of organizations utilize OS software in their socio-technical systems (Schnaack and Termer 2023). This dependency on OS software is even higher in the implementation of AI systems, with essential AI implementation tools like TensorFlow, PyTorch, and scikit-learn emanating from OS software, underlining the pivotal role of OS AI software, with its flexible and collaborative nature in the implementation of AI systems (Abadi et al. 2015; Paszke et al. 2019; Pedregosa et al. 2011; Widder et al. 2022).

While OS software offers many advantages, it also presents substantial operational and security vulnerabilities for organizations (e.g., Goggins, Lumbard, and Germonprez 2021; Synopsys Incorporation 2023). In 2022, an investigation of over 1,500 OS software projects revealed operational and security vulnerabilities in more than 85% of them (Synopsys Incorporation 2023). Notable examples include the Log4j zero-day vulnerability impacting numerous socio-technical systems worldwide (Perez-Etchegoyen and Forbes Technology Council 2023), libssh enabling unauthorized SSH connections (Red Hat Incorporation 2018), and Apache Struts 2 enabling web-server compromises via malicious uploads (Apache Software Foundation 2017). Despite the risks inherent in OS software, there is no legal foundation to hold developers liable for their code as they use software licenses such as MIT and Apache-2.0 to exclude liability in all cases (Apache Software Foundation 2004; MIT 1987). Consequently, developers contributing to OS software projects may not be legally compelled to provide secure and functioning code nor to rectify bugs after deployment (e.g., Apache Software Foundation 2004; MIT 1987). This absence of liability raises the critical question of how to incentivize developers to contribute secure and functioning code. In response, previous research on socio-technical systems suggests the importance of code accountability for developers as a motivational factor for enhancing the security and integrity of their contributed code (e.g., Martin 2019b; Novelli, Taddeo, and Floridi 2023).

Code accountability describes the inherent accountability to explain and justify developed code to relevant stakeholders. Thus, it necessitates developers to explain and justify the behavior and functioning of their code to others, particularly to third parties utilizing the code to implement socio-technical systems (e.g., Novelli, Taddeo, and Floridi 2023; Wieringa 2020). In this vein, accountability frequently carries significant implications for developers, ranging from sanctions to rewards (Bovens 2007). Such sanctions and rewards can appear, on the one hand, in the form of reputation or even job loss or, on the other hand, as being promoted or receiving bonuses, acting as vital external accountability mechanisms to motivate developers to adhere to guidelines and rules as well as motive them to behave ethically (e.g., Bovens 2007; Finnigan and Gross 2007; Martin 2019b). However, in the OS domain, these external accountability mechanisms are largely inapplicable, as developers are usually volunteers who are not tied to employment contracts nor receive monetary incentives for their code contributions (Dalle et al. 2004; Fitzgerald 2006). This lack of external accountability mechanisms stems from the chosen software licenses within the OS software projects that exclude any consequences from them (e.g., Apache Software Foundation 2004; MIT 1987). Consequently, the formal concept of code accountability, as understood in prior research, faces challenges in the OS domain due to the absence of external accountability mechanisms, making the OS domain different from classical socio-technical systems contexts (e.g., Bartsch and Schmidt 2023; Horneber and Laumer 2023). Nevertheless, it is crucial to consider that developers may still adopt a sense of code accountability driven by personal factors to hold themselves accountable for their code (e.g., Bartsch et al. 2024; Seguel and Vaast 2021). In this vein, developers might recognize and internalize a specific amount of code accountability, which we define and describe as perceived code accountability.

Remarkably, while the use and risk of continuously developed and improved OS software projects are indispensable for organizations, it is surprising that previous research on socio-technical systems did not pay much attention to strategies to motivate developers to consistently contribute new and high-quality code (e.g., Bonaccorsi and Rossi 2003; Rolandsson, Bergquist, and Ljungberg 2011). Instead, existing research has primarily focused on identifying factors that generally motivate developers to participate in OS software projects, often overlooking the impact of these factors on the quality of their contributed code (e.g., Hertel, Niedner, and Herrmann 2003; Sharma et al. 2022; Wu, Gerlach, and Young 2007). However, although past research underscores the need for code accountability among developers to explain and justify their code, it falls short in examining how developers can be made more aware of their accountability obligations and how they respond to such expectations in the absence of external accountability mechanisms (e.g., Ahmed and van den Hoven 2010; Martin 2019b). This absence leaves a critical gap in understanding developers' motivation to contribute high-quality code. Without these insights, socio-technical research may fail to comprehend how developers' perceived code accountability is present within the OS domain - despite the lack of external accountability mechanisms - and how it navigates them to contribute secure and reliable code continuously. Against this background, we raise the following research questions:

RQ1: What shapes developers' perceived code accountability in OS domains?

RQ2: How does developers' perceived code accountability affect their code contributions in OS AI software projects?

Drawing on existing literature on accountability, we conducted 26 semi-structured interviews with developers who actively contribute to OS AI software projects. We decided to focus on the OS AI software development context as this context is an important and representative area within OS software projects due to the particular characteristics of diverse OS AI software projects, as well as the primary use of such projects within the implementation of AI systems (Yang et al. 2023). Our analysis reveals that developers within the OS domain do indeed perceive a sense of code accountability, influenced by a complex interplay of system-, project-, and individual-related factors. Notably, system-related factors often prevent their accountability perception, whereas individual-related factors tend to promote it. Project-related factors play a dual role in affecting developers' perceived code accountability, with the effect varying according to project integration (high vs. low) and the developers' work status (professional vs. free and voluntary). Crucially, our findings highlight a significant trade-off of developers' perceived code accountability in that an increased code accountability perception increases the assumed code quality while simultaneously discouraging developers' intention to participate in OS AI software projects.

Our study contributes to accountability mechanisms in socio-technical research, particularly in the development of OS AI software, in two important ways: First, it addresses a gap in previous research on socio-technical systems, which emphasized the need for developers to adopt code accountability but did not provide clear pathways beyond external accountability mechanisms for enhancing their willingness to adopt it (e.g., Ahmed and van den Hoven 2010; Martin 2019b). Our findings show that system-, project-, and individual-related factors play a critical role in raising developers' perceived code accountability in the absence of external accountability mechanisms. Thus, our findings shift the focus from solely external accountability mechanisms to a deeper understanding of developers' accountability perceptions behind such mechanisms, highlighting the importance

of recognizing why developers perceive themselves as accountable for their developed code. Second, our study diverges from existing research that primarily focused on motivating developers to contribute to OS software projects without considering potential downsides (e.g., Oreg and Nov 2008; Sharma et al. 2022), as we uncover a nuanced trade-off between the number of code contributions and the contribution of high-quality code. This trade-off arises from developers' perceived code accountability, suggesting that the downsides of accountability perceptions within OS AI software projects should not be overlooked. With these findings, we advocate for careful consideration by maintainers of OS AI software and organizations of developers' perceived code accountability to navigate this trade-off, balancing the encouragement of abundant code contributions with the necessity for high-quality code.

Theoretical Background

Open Source Software in Socio-Technical Systems' Implementation

OS software is defined as software "that allows for the modification of source code, is freely distributed, is technologically neutral, and grants free subsidiary licensing rights" (Aksulu and Wade 2010, p. 577). Moreover, OS software is characterized by not discriminating against any individual persons or societal groups and predominantly relies on developers' voluntary participation (Dalle et al. 2004; Fitzgerald 2006; Perens 1999). Organizations benefit from using OS software as it obviates the need for significant time and financial investments in developing or buying proprietary software, thus acquiring it free of charge for integration into their socio-technical systems (e.g., Fitzgerald 2006). Despite its accessibility and cost-free usage, utilizing OS software within socio-technical systems' implementation (i.e., using OS software to create socio-technical systems) carries inherent risks for organizations due to possible bugs (e.g., Fitzgerald 2006; Pistilli et al. 2023; Synopsys Incorporation 2023). Since licenses typically associated with OS software explicitly exclude liability for any occurring bugs, the legal responsibility transfers to organizations utilizing OS software (e.g., Apache Software Foundation 2004; MIT 1987). Consequently, organizations bear the legal liability for any bugs in the socio-technical systems they implement and are advised to rigorously check the OS software project prior to usage (e.g., Martin 2019b; Novelli, Taddeo, and Floridi 2023). This checking process is crucial for organizations to identify and rectify bugs, thereby preventing the implementation of flawed socio-technical systems that could potentially harm their systems' users. However, the challenge for organizations lies in the complexity of OS software projects, which often comprise thousands of lines of code, making it challenging to fully comprehend and identify bugs (e.g., Allamanis and Sutton 2013; Mockus, Fielding, and Herbsleb 2000). As a result, organizations benefit when developers already contribute high-quality code to OS software projects. This not only enhances the reliability of OS software projects but also potentially reduces the need for extensive and challenging double-checking of the code by organizations.

In the realm of OS software, organizations primarily assume the role of testing these projects and identifying bugs, thereby creating feedback loops in which developers are informed of, but not obligated to rectify, reported issues. On the contrary, the primary focus of developers usually lies in the development and contribution of new features, which are then integrated into the OS software project (Aberdour 2007). Furthermore, the voluntary nature of contributions in OS software projects necessitates motivating factors to encourage developers' participation. Such motivations include enhanced social status, personal alignment with OS software projects' values, and hedonistic motives like enjoying the coding process (e.g., Aberdour 2007; Hertel, Niedner, and Herrmann 2003; Sharma et al. 2022). However, while these factors mainly focus on how developers can be motivated to make continual contributions to OS software projects, they do not directly investigate and address how to make developers develop high-quality code. In this vein, previous research posits the concept of holding developers accountable for their code as a promising approach, aiming to encourage developers to scrutinize their code more rigorously to proactively identify and rectify bugs (e.g., Bartsch et al. 2024; Martin 2019a; Seguel and Vaast 2021). Therefore, the concept of code accountability suggests a shift towards a more conscientious and quality-focused approach in OS software development, potentially leading to more secure and reliable OS software projects.

Developers' Code Accountability in Open Source Software Projects

Accountability is defined as the need to explain and justify one's behavior (Bovens 2007) and is contextualized as algorithmic accountability, implying the "obligation to explain and justify their use, design, and/or decisions of/concerning the system and the subsequent effects of that conduct" (Wieringa 2020, p. 10). This concept is a cornerstone in the governance of socio-technical systems, requiring developers to explain and justify their implemented socio-technical systems to other stakeholders, including managers and users (Novelli, Taddeo, and Floridi 2023). In this vein, algorithmic accountability suggests that developers also need to explain and justify the behavior and functionality of their developed code, which we define as code accountability.

In this context, consequences such as sanctions and rewards act as crucial external accountability mechanisms,

compelling developers to adhere to guidelines and rules (e.g., Bovens 2007; Novelli, Taddeo, and Floridi 2023). These external accountability mechanisms appear in various forms like fines, job losses, and bonus payments, and play a significant role in shaping developers' behavior throughout the implementation of socio-technical systems (e.g., Bartsch and Schmidt 2023; Bovens 2007; Novelli, Taddeo, and Floridi 2023). Accordingly, it is likely that the presence of external accountability mechanisms often results in developers adopting a more cautious approach to their work, encouraging them to engage in more frequent and thorough code reviews during the development of OS software.

Since OS software projects often rely on software licenses that explicitly exclude legal liability for developers, external accountability mechanisms are often absent in OS software projects (e.g., Apache Software Foundation 2004; MIT 1987). The absence of external accountability mechanisms is due to the predominantly voluntary participation in OS software projects, coupled with the lack of financial compensation for time and code contributions (e.g., Aberdour 2007; Fitzgerald 2006). Consequently, unlike in other contexts of socio-technical systems implementation where external accountability mechanisms like fines and job losses are prevalent, code accountability may not inherently influence developers' behavior in the OS domain without these external accountability mechanisms (e.g., Vance, Lowry, and Eggett 2015). Therefore, developers contributing to OS software projects are not encouraged to adhere to specific guidelines and rules. However, qualitative observations suggest that developers' ethical and moral considerations affect their perceptions of accountability for their developed code (Bartsch et al. 2024; Seguel and Vaast 2021). These observations point to the existence of personal drivers that may motivate developers to perceive themselves as accountable for their contributed code, even in the absence of external accountability mechanisms. Consequently, perceived code accountability expresses the extent to which developers recognize, perceive, and internalize code accountability, regardless of whether or not code accountability is enforceable through external accountability mechanisms.

Previous research on socio-technical systems has predominantly focused on motivating developers to participate in OS software projects, emphasizing aspects like human capital enhancement and personal fulfillment to bolster ongoing participation (e.g., Hertel, Niedner, and Herrmann 2003; Sharma et al. 2022). More specifically, prior research has identified various motivational factors, including social and hedonistic motivations, as well as alignment with the OS software projects' values and monetary incentives, enhancing code contribution frequency. However, prior research has largely overlooked the role of code accountability despite its recognized benefits in socio-technical systems development projects (e.g., Bartsch et al. 2024; Hall, Frink, and Buckley 2017). Additionally, previous research has investigated how developers' perceived accountability for AI systems' outcomes can be increased without exploring potential implications, leaving uncertainty in how perceived accountability affects OS software development projects (Schmidt, Bartsch, and Adam 2023). Furthermore, while the importance of developers assuming accountability for their code has been underscored, there is limited knowledge in understanding how to heighten developers' awareness and commitment thereof (e.g., Ahmed and van den Hoven 2010; Schmidt et al. 2023). Therefore, previous research provides limited insights into how code accountability appears and behaves in contexts lacking external accountability mechanisms, making it challenging to understand the potential effects of code accountability within the OS domain.

Research Methodology

Open Source AI Software Development Context

To explore the potential effects of code accountability in the OS domain, we relied on developers' perceived code accountability, as perceived code accountability provides us information about the extent to which developers recognize and internalize the formal obligations of code accountability, making code accountability tangible and measurable.

Moreover, since the OS domain encompasses many different types of projects, such as the development of whole operation systems and programming languages, we specifically focus on the development of OS AI software. This context is particularly relevant given the rising prominence and uses of AI systems in both organizations and society (e.g., Adam, Roethke, and Benlian 2023; Berente et al. 2021). AI systems, while powerful and innovative, are not immune to algorithmic flaws and ethical concerns, posing significant challenges for developers tasked with their implementation (Asatiani et al. 2020; Berente et al. 2021; Martin 2019a). Examples of AI systems containing bugs and behaving not in the intended way are AI systems leading to arrests resulting from inaccurate facial detections and biases in automated decision-making processes, like discrimination in loan application AI systems (Gal, Hansen, and Lee 2022; Hill 2023). Not least, these incidents have convinced legislators, organizations, and researchers to think about accountable AI systems (e.g., High-Level Expert Group on AI 2019; IBM 2022; Martin 2019a). As a result, legislators passed the EU AI Act to regulate AI systems, and organizations created accountability guidelines for their employees (e.g., European Parliament 2024; IBM 2022). In this evolving landscape, developers' accountability extends beyond just adhering to guidelines and rules as it encompasses developers possessing appropriate and functional code to implement AI systems.

Another reason for focusing on the OS AI software development context was because AI systems are highly dependent on OS AI software projects (Yang et al. 2023). Therefore, OS AI software projects are paramount to the implementation of the variety of AI systems in use today and allow us to gain many different perspectives on OS software development. This helps us to provide a more generalized view of the effects of developers' perceived code accountability within the OS domain. Additionally to the opportunity of gaining a diverse perspective, we chose the OS AI software development context because in the realm of AI systems' implementation, the majority of available OS AI software projects, such as TensorFlow, PyTorch, and scikit-learn, are characterized by continuous development, free availability, and non-commercial distribution (e.g., Abadi et al. 2015; Paszke et al. 2019; Pedregosa et al. 2011; Widder et al. 2022). Thus, the implementation of AI systems contrasts with software for traditional socio-technical systems, where developers can often rely on organization-distributed software for which organizations are obligated to explain, justify, and correct any occurring bugs. Since organization-distributed software often does not exist when implementing AI systems, developers in organizations face limited choices and must either accept available OS AI software 'as is' or undertake the laborious task of understanding the OS AI software and, thus, scrutinizing and correcting bugs within it, which is nearly a challenging task alongside their primary goal of implementing AI systems (e.g., Khan 2013; Nidhra et al. 2013). Given these circumstances, developers must be motivated not only to innovate and add new features to OS AI software projects but also to ensure the development of high-quality code, making this context ideal for our study.

Interviews

We conducted 26 semi-structured interviews with developers actively contributing to the most advanced and prominent OS AI software projects like TensorFlow, PyTorch, and scikit-learn (Abadi et al. 2015; Paszke et al. 2019; Pedregosa et al. 2011). After conducting the 26th interview, we only gained a few additional insights. Therefore, we followed previous research and stopped conducting additional interviews as we observed indicators for having a comprehensive and extensive data set for our subsequent analysis (Keutel, Michalik, and Richter 2014). To identify relevant projects, we utilized GitHub as a search platform for OS AI software projects, employing keywords such as *artificial intelligence*, *machine learning*, and *deep learning* (GitHub 2007). OS AI software projects were then selected based on their popularity, as indicated by the number of stars they received, and repositories with more than 10,000 stars were selected. We contacted developers involved in these OS AI software projects, inviting them to participate in voluntary online interviews. Our interviews lasted, on average, 33 minutes (std = 10.7) and were recorded digitally and transcribed for further analysis with the developers' consent. The interviews explored developers' perceived code accountability in the domain of their OS AI software projects, particularly focusing on how this perception affects their code development in the absence of external accountability mechanisms. In this vein, we created and followed an interview guide, which Table 1 in the supplementary materials shows (see *doi.org/10.13140/RG.2.2.28402.11206*).

Analysis

We conducted an inductive qualitative content analysis method to analyze our data (Elo and Kyngäs 2008). First, keeping our research question in mind, we created comprehensive reviews of each interview. Subsequently, we established a set of evaluation categories. These categories were designed to classify and code the relevant statements from the interviews systematically. By sorting these coded segments based on their relevance, we identified distinct characteristics within each evaluation category. This process of sorting and identifying was iterative, involving continuous refinement and additional coding as new characteristics emerged from the data. Inconsistencies in coding were meticulously discussed among the authors' team until we achieved consensus. As an example, Figure 1 illustrates the resulting data structure of project-related factors affecting developers' perceived code accountability.

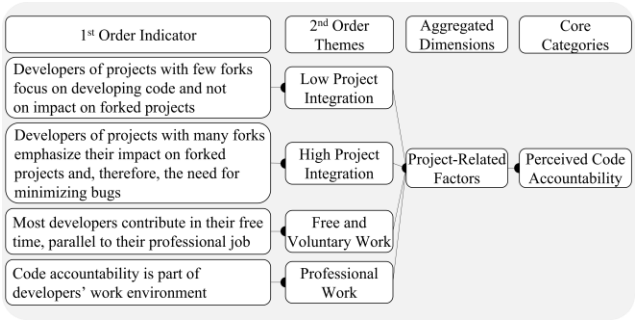

Figure 1: Data Structure of Project-Related Factors Affecting Developers' Perceived Code Accountability

Participant Demographics

96.15% of the interviewed developers identified themselves as male, 3.85% as female, and 0.00% as non-binary, and they have an average of 9.6 years (std = 5.2) of OS AI software development experience. They have participated, on average, in 66.62 (std = 88.13) OS AI software projects with an average number of 7.55k (std = 8.30k) code commits. The sample included 21 developers (coded as OSD1-OSD21). Sixteen developers contributed code to OS AI software projects in their free time only, and five were full-time contrib-

utors to these projects, working at organizations such as Alphabet and Meta. Moreover, we interviewed five developers who, besides contributing code, additionally maintain the OS AI software project (coded as OSM1-OSM5). This varied group of interviewees provided us with a wide range of perspectives on the topic of perceptions of code accountability in the OS AI software development domain.

Ethical Considerations

Although the interviews did not delve into personally sensitive areas, we exercised particular care in handling ethical concerns. In doing so, we provided full disclosure to the developers regarding the interview process and ensured their anonymity in the analysis. To confirm that the developers agreed to this, we obtained verbal declarations of consent. In addition, we also emphasized that there were no right and wrong answers to prevent developers from steering in a predefined direction and reduce the social desirability bias. In this way, we provided developers with a safe and open space to tell us about their OS AI software development experiences without directing them into ethically questionable or unpleasant situations. Finally, to protect the anonymity of the developers, we decided to describe only a minimum of socio-demographic data to avoid any conclusions about specific developers or OS AI software projects.

Findings

The conducted interviews provided nuanced insights into developers' perceived code accountability and its influence on their activities during OS AI software development. Specifically, we identified system-, project-, and individual-related factors as preventing or promoting factors shaping developers' code accountability. Furthermore, developers' perceived code accountability affects their behavior within OS AI software development, in which higher code accountability perceptions reduce the intention to participate in OS AI software projects while improving the code quality.

Perceived Code Accountability

System-Related Factors

Software Licenses: Our analysis of the most frequently used OS AI software projects revealed the prevalent use of MIT and Apache-2.0 software licenses (Apache Software Foundation 2004; Golubev et al. 2020; MIT 1987). For instance, among the top 70 OS AI software projects, which we ranked according to the number of stars, we found the use of MIT or Apache-2.0 software license 56 times (Table 2 in the supplementary materials lists the analyzed OS AI software projects and Table 3 lists the distribution of software licenses of these projects). These software licenses release developers from their code accountability, explicitly stating they provide their code 'as is,' requiring end-users (i.e., developers utilizing OS AI software, for example, in organizations implementing AI systems) to scrutinize the code before their usage. Thus, software licenses make external accountability mechanisms ineffective from a legal perspective (e.g., Apache Software Foundation 2004; Bernelin 2020; MIT 1987). Against this background, one developer stated the following:

> "Currently, if you're doing open source work, you're not going to get any kind of consequences. (...) Even people who have injected malicious code in popular packages, I think they got away with nothing since there isn't really a legal structure there to be had." (OSM5)

Some developers, albeit a minority, are aware of these legal regulations. However, for many, software licenses are an overlooked aspect during their contributions to OS AI software projects. Therefore, despite the exclusion of code accountability through legal regulations caused by such software licenses, it only impacts developers engaging actively with legal regulations. Many developers perceive software licenses merely as legal safeguards and do not deal with them in detail, stating that their primary task is code development and that software licenses are less relevant. In particular, developers unanimously emphasize that end-users are accountable for the use of OS AI software code and thus transfer both the risk and the responsibility for usage to them when implementing socio-technical systems. In this vein, one developer said:

> "To be honest, only in certain cases, except if I have to use an open source software project as part of my work, then I have actually to care about the license, but in general, I do not excessively care about the licensing as long as it is open source." (OSD10)

Project-Related Factors

Project Integration: Developers' perceived code accountability varies with the integration of OS AI software projects into other projects, which we determined by measuring the number of forks on GitHub. OS AI software projects with more integrations into other projects tend to evoke higher perceived code accountability among developers compared to OS AI software projects with fewer integrations.

OS AI software projects with many integrations into other projects are typically well-known within the community of end-users, leading to increased dependencies on other projects, as indicated by the high number of forks on GitHub. Developers acknowledge this interest in their OS AI software projects and realize the repercussions of bugs in their code contributions, affecting multiple dependent projects. Consequently, they perceive themselves as accountable for contributing high-quality code, which they think end-

users expect from and attribute to them. One developer articulated this perception by the following:

> "Because I feel that everything I modify, every bug I introduce, every mistake I make would affect so many people (users). I may accidentally break their systems and waste hundreds of hours of other developers just figuring out how to fix this. So that's why I feel more accountable." (OSD16)

Conversely, developers contributing to **OS AI software projects with few integrations into other projects** perceive a lower impact from their OS AI software projects and, thus, a reduced impact from their code contributions. Small projects usually have fewer integrations into other projects since they are often initiated for personal use. As a result, developers perceive less code accountability for such projects, as they are primarily intended for their private use and are less concerned about end-users utilizing their code. Additionally, developers contributing to small OS AI software projects think that end-users have lower expectations, making them assume less code accountability; thus, they do not perceive the need to assume accountability for their code contributions. Moreover, they rarely receive feedback on bugs due to the infrequent usage of end-users, reinforcing their denied view towards assuming code accountability, as one developer shared with us:

> "There are other small things (projects) that I have that are just mainly for myself and that no one else uses. So, I feel more relaxed, just like changing things there. And like, I guess like the accountability is not so much in like, 'Oh, I'm going to make a mistake that is going to break (something or) is going to leak their data' or something like that or something super terrible." (OSM3)

Thus, project integration is important in shaping developers' perceived code accountability, exerting both promoting and preventing effects on their code accountability perceptions.

Work Status: Developers' work status, which describes whether they contribute voluntarily in their free time or as employees within organizations receiving payments, affects their perceived code accountability. In this vein, contributing code voluntarily is associated with lower code accountability perceptions, while being employed is associated with higher code accountability perceptions.

Developers who contribute code **voluntarily and in their free time** are not willing to be held accountable for their code. They justify this negligence by stating that they invested time without any compensation and that they contribute to OS AI software projects available to others out of their benevolence. In this vein, developers perceive code accountability as an additional burden, which they deny since they are convinced that it is their decision to do whatever they want in their free time. On the contrary, developers believe that end-users should be pleased that they take the time to contribute code voluntarily. As a compromise, one developer told us that end-users should either pay them for their code contributions and hold the developers accountable or not pay them and not hold them accountable for their code:

> "It's like you buy something to get (your job) faster (done), but it's free. So, you can't have everything. You can't find something free and then ask for accountability to others (...) it's like a trade-off." (OSD19)

On the other hand, developers employed by organizations **working on OS AI software projects professionally** claimed higher perceived code accountability. Examples of OS AI software projects that organizations maintain and, thus, employ developers within their organization are TensorFlow (Abadi et al. 2015), maintained by Alphabet, and PyTorch (Paszke et al. 2019), maintained by Meta. These employed developers perceive themselves as accountable for their code contributions, as they get monetary compensation. Therefore, their employment status exposes them to external accountability mechanisms such as sanctions and rewards like job loss or bonus payments. This exposure leads these developers to consider their contributed code more carefully and perceive code accountability more compared to OS AI software projects they run in their free time. In this context, developers explained that they test their contributed code more closely before publishing compared to their free time projects:

> "(...) we are aware that some of our clients are using the tools, so (...) we are always testing things before updating the project. So probably, I think more about accountability when I do things for a company or when it's not a personal spare time project." (OSD21)

Consequently, work status shapes developers' perceived code accountability, with professional developers exhibiting higher levels of perceived code accountability compared to those contributing voluntarily in their free time.

Individual-Related Factors

Reputation: Developers' perceived reputation, both within and outside the OS AI software community, affects their perceived code accountability. Beyond community recognition, developers perceive code contributions as a means to showcase their development expertise, thus enhancing their career prospects. Accordingly, they highlight the need to prove, for example, by collecting badges or showing certificates, that they have contributed to one or more OS AI software projects. In this vein, developers are confronted with having to explain and justify their code to others, like future employers, thus perceiving code accountability, as one developer explained:

> "I really care about my career, and doing a good job in open source software projects is a good way to showcase my capability in software engineering. And I can always showcase the software I did, I contributed to, I

developed to the interviewer while I try to find a job. So that's why I try to do my best in these open source software projects." (OSD17)

As fulfilling expectations from potential future employers could require developers to closely scrutinize and review their developed code before contributing it, developers find additional motivation for such tasks as their names are referenced to the OS AI software projects. Through this motivation, developers invest more time and effort to identify and correct bugs. In this context, one developer stated:

"The motivation is that you are putting your name on it. So, other people will see how you work. Other people will see how you deal." (OSD8)

Effects of Perceived Code Accountability in Open Source Software Projects

Intention to Participate in OS AI Software Projects

When developers perceive themselves accountable for their code contributions, their inclination to participate in OS AI software projects tends to decrease. Developers view code accountability as a deviation from their main objective of developing code and pursuing personal interests, such as covering their specific use cases within OS AI software projects. In doing so, they perceive code accountability as an additional burden that they think they have deliberately excluded legally through software licenses. This perceived burden can be seen as detracting from the freedom inherent in voluntary work, as it introduces obligations to adhere to certain guidelines and rules. As a result, developers feel as if they are in an uncompensated, quasi-employment situation, diminishing their motivation to contribute code. In contrast, when developers do not perceive this burden, they are more inclined to contribute to OS AI software projects. This freedom allows them to pursue their interests and code contributions without the obligation to explain and justify their code to others. As one developer explained, the absence of accountability perceptions for their code provides a sense of autonomy and aligns more closely with their original motivations for engaging in OS AI software development:

"I mean, if there was such a big barrier to entry that you would have to be accountable for the entire work that you're putting out there (the contributed code), nobody would make open source work, definitely. So, this kind of freedom from consequences is necessary for developers to be able to share their work openly and freely like this." (OSM5)

Assumed Increased Code Quality

The perception of code accountability among developers, while dampening their willingness to engage in OS AI software projects, simultaneously fosters a commitment to contribute high-quality code, resulting in secure and reliable code fulfilling its functions. This emphasis on high code quality primarily arises from high project integrations and the pursuit of reputation. Motivated by these factors, developers are more likely to diligently address and rectify reported bugs as well as to scrutinize and carefully think about the impact of their code contributions. Scrutinizing code is a vital aspect of enhancing the overall quality and reliability of OS AI software projects since end-users encounter fewer bugs and experience more secure and sophisticated software. One developer succinctly encapsulated this perspective, highlighting the dual nature of code accountability:

"Because right now, if I'm doing an update and I'm doing some code, I'll push it, I won't think if this is a big security risk. But if you put in the legal obligations for that, that would actually improve the quality of the code." (OSD13)

Furthermore, developers' perceived code accountability strengthens their motivation to address reported bugs and, thus, improve the code quality: Since OS AI software projects can be misused, for example, to deliberately deceive humans by creating deep fakes, developers highlight the importance of high code quality to prevent such misuse. In this vein, developers emphasize code accountability to assess whether their code creates more good than bad. Based on this assessment, developers decide whether their code possesses high quality and can be contributed to the OS AI software project, as one developer told us:

"I won't say it (misuse) is not possible, but I don't think that possibility would stop me or other open source software developers from contributing towards a project that can have more good than bad." (OSD13)

Synthesized Model

The findings elucidate how system-, project-, and individual-related factors affect developers' perceived code accountability, leading to a trade-off between participation in OS AI software projects and high-quality code, as depicted in Figure 2.

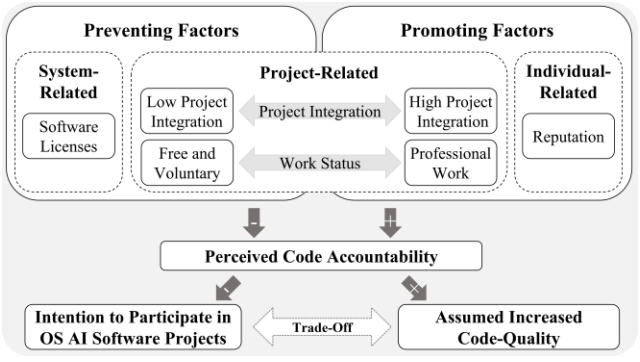

Figure 2: The Effects of Developers' Perceived Code Accountability within OS AI Software Development

Discussion

The vital role of OS AI software in underpinning AI systems underscores the need for secure and reliable OS AI software projects. Against this backdrop, this paper investigates factors shaping developers' perceived code accountability in OS domains and how developers' perceived code accountability affects their continuous contribution of high-quality code. Our findings indicate that system-, project-, and individual-related factors shape developers' perceptions. System-related factors tend to impede, whereas individual-related factors encourage it. Project-related factors have a mixed effect, varying with the project integration and developers' work status. Additionally, we observed a trade-off promoted by developers' perceived code accountability since, while it enhances code quality, it reduces their willingness to participate in OS AI software projects. This trade-off highlights the complexity of motivating developers in OS AI software projects, balancing the quality of their contributed code with promoting their active engagement.

Contributions to Research

This paper contributes novel insights into mechanisms of accountability in socio-technical research, particularly in the development of OS AI software, in two significant ways: First, we reveal that perceived code accountability emerges even in the OS domain, a context in which external accountability mechanisms are usually not applicable. Our findings reveal that developers can and do perceive themselves as accountable for their code even in the absence of such external accountability mechanisms. Their perceptions of code accountability emerge from a complex interplay of system-, project-, and individual-related factors that both promote and inhibit them. Previous research on socio-technical systems emphasized the need for code accountability among developers but insufficiently suggested pathways and addressed how to foster this awareness beyond external accountability mechanisms (e.g., Ahmed and van den Hoven 2010; Martin 2019b). Our study addresses this gap by identifying project- and individual-related factors in driving developers' code accountability perceptions, suggesting that, while external accountability mechanisms may be beneficial, they are not necessary to foster a sense of code accountability. This finding is important for informing future research on ways to investigate the perceptions and effects of code accountability by looking at surrounding project-related factors as well as developers' individual-related factors as drivers of code accountability in addition to focusing on external accountability mechanisms.

Second, developers' perceived code accountability is not only beneficial because of increased code quality, but it also has downsides presented by reduced intentions to participate in OS AI software development. This finding is a significant extension of previous research on socio-technical systems, which largely focused on the alignment of developers' values with OS software projects and their code contributions that support positive effects without considering potential downsides (e.g., Oreg and Nov 2008; Sharma et al. 2022). By revealing the trade-off between participation and contributing high-quality code, our findings underscore the complexity of leveraging developers' perceived code accountability, balancing the need for new feature development with the imperative of secure and reliable code. Understanding this shift in potential consequences is critical in advancing and implementing more sophisticated and reliable AI systems, thereby enriching our understanding of developers' behaviors and motivations in OS AI software projects.

Implications for Practice

Our paper also offers practical implications for maintainers of OS AI software projects and organizations that cannot rely on external accountability mechanisms like job losses and fines to enforce code accountability but also seek to improve and better manage their OS AI software projects: First, maintainers of OS AI software projects and organizations can leverage developers' perceived code accountability to enhance the code quality within their OS AI software projects. This can be done by incentivizing developers through recognition, such as issuing certificates for job applications, or by appealing to their sense of voluntary code accountability towards end-users. Such strategies not only motivate developers to identify and rectify bugs but also improve the usability and reliability of OS AI software projects. This finding is particularly advantageous for organizations, given their obligation to explain and justify their AI systems to others, regardless of whether they utilize OS AI software or not, since they experience bugs less frequently (e.g., Martin 2019b; Novelli, Taddeo, and Floridi 2023). Furthermore, maintainers of OS AI software projects also reap benefits from improved code quality as their OS AI software projects become more appealing and usable for third parties through reduced bugs. However, as higher code quality often signifies increased effort and time investment that results in slower code development and OS AI software project progress, we recommend finding a balance between encouraging high-quality code and the rapid development of new features.

Second, maintainers of OS AI software projects are, in particular, tasked with navigating the critical balance between continuous code contributions and ensuring high-quality code when drawing on developers' perceived code accountability. Finding a good balance is especially important as the goals of OS AI software projects vary depending on the specific development phase of the project: Maintainers of OS AI software projects may need to prioritize

rapid project progression at certain stages while, at other times, enhancing existing code for improved quality becomes more important. One opportunity for maintainers of OS AI software projects to influence developers' perceived code accountability might be through communication within the OS AI software project. By explicitly stating that developers are not required to explain and justify their code as well as emphasizing the lack of external accountability mechanisms, maintainers of OS AI software projects might reduce developers' perceived code accountability perceptions. Conversely, by implementing internal project policies that emphasize code peer reviews and extensive testing, maintainers of OS AI software projects might enhance the focus on high code quality. These strategies enable maintainers of OS AI software projects to respond adaptively to the evolving needs of their OS AI software project, whether that be accelerating project development or enhancing code quality. Consequently, maintainers of OS AI software projects must strategically evaluate and address the trade-off of project progress versus code quality based on the current circumstances of their OS AI software projects. However, as it gets increasingly difficult to fix bugs at a later stage of the project, the development of high-quality code should not be neglected at any stage of the OS AI software project.

Limitations and Directions for Future Research

Our study, while contributing valuable insights into mechanisms of accountability in socio-technical research, is subject to certain limitations that pave multiple ways for future research: First, the qualitative nature of our research approach offers a broad understanding of the effects of absent external accountability mechanisms but lacks the statistical robustness of quantitative studies. Future research could adopt methodologies like designing comparable quantitative experiments to enhance statistical validity. Specifically, experiments manipulating developers' perceived code accountability could provide a clearer understanding of the impacts of absent or present external accountability mechanisms. In this vein, future research could conduct case studies or action research to uncover deeper insights into the accountability dynamics of specific OS software development environments. This approach would yield a more detailed perspective on the implications of lacking external accountability mechanisms regarding developers' code accountability perceptions.

Second, given the diversity of OS software projects, we decided to focus on OS AI software development projects as specific instances of OS software projects. While this approach allowed us to investigate developers' perceived code accountability using concrete examples, our paper is limited to one important of many different types of OS software projects. While our insights are likely to translate to other types of OS software projects because such projects often share common principles and practices like software licenses and voluntary contributing, the generalizability of our results needs to be empirically tested and supported. Empirical tests are important as boundary conditions in other OS software projects might exist, leading to different or contrary results. Future research should, therefore, explore developers' perceived code accountability within the variety of OS software projects.

Third, when asking developers about their behavior and activities affected by their perceived code accountability, we obtained information about activities such as code peer reviews, which are usually effective in increasing code quality. However, since code quality is subjective to everyone and conducting code peer reviews might not always result in higher code quality, we treated such information as developers' assumed increased code quality. To objectify the measurement of code quality, future research could conduct longitudinal experiments focusing on specific OS software projects to measure the effects of perceived code accountability on code quality by evaluating the developed code based on quantitative, measurable indicators like counting occurring bugs or feature requests.

Conclusion

In the dynamic world of today's AI systems landscape, the integration of continuously evolving OS AI software projects has become indispensable. However, this reliance brings the risk of introducing flawed code, potentially leading to the deployment of compromised AI systems that could pose significant risks to AI systems' users. Our paper highlights the critical role of developers' code accountability in shaping OS AI software projects, particularly how their accountability perceptions for their code arise in these contexts that are usually free of external accountability mechanisms and how it relates to a trade-off between continuous code contributions and the contribution of high-quality code. Therefore, this paper calls to action for continued exploration of developers' perceived code accountability to determine development practices that meet the evolving requirements of both feature-rich and secure AI systems by utilizing secure and reliable OS AI software projects.

Acknowledgments

The authors gratefully acknowledge the funding support by the German Research Foundation (DFG) as part of the project "Accountable Artificial Intelligence-based Systems" (project number 471168026). In addition, the authors would like to thank the anonymous reviewers and editors for their valuable comments and suggestions for improving this paper.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., & Devin, M. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. doi.org/10.48550/arXiv.1603.04467

Aberdour, M. 2007. Achieving quality in open-source software. *IEEE Software*, *24*(1), 58-64. doi.org/10.1109/MS.2007.2

Adam, M., Roethke, K., & Benlian, A. 2023. Human versus automated sales agents: How and why customer responses shift across sales stages. *Information Systems Research*, *34*(3), 1148-1168. doi.org/10.1287/isre.2022.1171

Ahmed, M. A., & van den Hoven, J. 2010. Agents of responsibility - freelance web developers in web applications development. *Information Systems Frontiers*, *12*, 415-424. doi.org/10.1007/s10796-009-9201-0

Aksulu, A., & Wade, M. R. 2010. A comprehensive review and synthesis of open source research. *Journal of the Association for Information Systems*, *11*(11), 576-656. doi.org/10.17705/1jais.00245

Allamanis, M., & Sutton, C. 2013. Mining source code repositories at massive scale using language modeling. 10th Working Conference on Mining Software Repositories (MSR), San Francisco, USA. doi.org/10.1109/MSR.2013.6624029

Apache License, Version 2.0, (2004). http://www.apache.org/licenses/

Apache Software Foundation. 2017. *CVE-2017-5638*. CVE. https://www.cve.org/CVERecord?id=CVE-2017-5638. Accessed: 11/26/2023

Asatiani, A., Malo, P., Nagbøl, P. R., Penttinen, E., Rinta-Kahila, T., & Salovaara, A. 2020. Challenges of explaining the behavior of black-box AI systems. *MIS Quarterly Executive*, *19*(4), 259-278. doi.org/10.17705/2msqe.00037

Bartsch, S. C., Milani, V., Adam, M., & Benlian, A. 2024. Algorithmic Accountability: What Does it Mean for AI Developers and How Does it Affect AI Development Projects. Hawaii International Conference on System Sciences, O'ahu, USA. www.hdl.handle.net/10125/107087

Bartsch, S. C., & Schmidt, J.-H. 2023. How AI Developers' Perceived Accountability Shapes Their AI Design Decisions. International Conference on Information Systems, Hyderabad, India. www.aisel.aisnet.org/icis2023/aiinbus/aiinbus/111

Berente, N., Gu, B., Recker, J., & Santhanam, R. 2021. Managing artificial intelligence. *MIS Quarterly*, *45*(3), 1433-1450. doi.org/10.25300/MISQ/2021/16274

Bernelin, M. 2020. The compatibility of open/free licences: a legal imbroglio. *International Journal of Law and Information Technology*, *28*(2), 93-111. doi.org/10.1093/ijlit/eaaa010

Bonaccorsi, A., & Rossi, C. 2003. Why open source software can succeed. *Research Policy*, *32*(7), 1243-1258. doi.org/10.1016/S0048-7333(03)00051-9

Bovens, M. 2007. Analysing and assessing accountability: A conceptual framework. *European Law Journal*, *13*, 447-468. doi.org/10.1111/j.1468-0386.2007.00378.x

Dalle, J.-M., David, P. A., Ghosh, R. A., & Wolak, F. A. 2004. Free & Open Source Software Developers and 'the Economy of Regard': Participation and Code-Signing in the Modules of the Linux Kernel. Oxford Workshop on "Libre Source" convened at the Oxford Internet Institute, Pisa, Italy.

Elo, S., & Kyngäs, H. 2008. The qualitative content analysis process. *Journal of Advanced Nursing*, *62*(1), 107-115. doi.org/10.1111/j.1365-2648.2007.04569.x

European Parliament. 2024. *Artificial Intelligence Act: MEPs adopt landmark law*. https://www.europarl.europa.eu/news/en/press-room/20240308IPR19015/artificial-intelligence-act-meps-adopt-landmark-law. Accessed: 04/16/2024

Finnigan, K. S., & Gross, B. 2007. Do accountability policy sanctions influence teacher motivation? Lessons from Chicago's low-performing schools. *American Educational Research Journal*, *44*(3), 594-630. doi.org/10.3102/0002831207306767

Fitzgerald, B. 2006. The transformation of open source software. *MIS Quarterly*, *30*(3), 587-598. doi.org/10.2307/25148740

Gal, U., Hansen, S., & Lee, A. S. 2022. Research Perspectives: Toward Theoretical Rigor in Ethical Analysis: The Case of Algorithmic Decision-Making Systems. *Journal of the Association for Information Systems*, *23*(6), 1634-1661. doi.org/10.17705/1jais.00784

GitHub. 2007. *GitHub*. www.github.com

Goggins, S., Lumbard, K., & Germonprez, M. 2021. Open source community health: Analytical metrics and their corresponding narratives. IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal), Madrid, Spain. doi.org/10.1109/SoHeal52568.2021.00010

Golubev, Y., Eliseeva, M., Povarov, N., & Bryksin, T. 2020. A study of potential code borrowing and license violations in java projects on github. IEEE/ACM 17th International Conference on Mining Software Repositories (MSR), Seoul, Republic of Korea. doi.org/10.1145/3379597.3387455

Hall, A. T., Frink, D. D., & Buckley, M. R. 2017. An accountability account: A review and synthesis of the theoretical and empirical research on felt accountability. *Journal of Organizational Behavior*, *38*(2), 204-224. doi.org/10.1002/job.2052

Hertel, G., Niedner, S., & Herrmann, S. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research*

Policy, *32*(7), 1159-1177. doi.org/10.1016/S0048-7333(03)00047-7

High-Level Expert Group on AI. 2019. *Ethics Guidelines for Trustworthy AI*. European Commission. www.digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

Hill, K. 2023. Eight Months Pregnant and Arrested After False Facial Recognition Match. *The New York Times*. https://www.nytimes.com/2023/08/06/business/facial-recognition-false-arrest.html. Accessed: 04/17/2023

Horneber, D., & Laumer, S. 2023. Algorithmic Accountability. *Business & Information Systems Engineering*, 1-8. doi.org/10.1007/s12599-023-00817-8

IBM. 2022. *Accountability*. https://www.ibm.com/design/ai/ethics/accountability/. Accessed: 07/01/2023

Keutel, M., Michalik, B., & Richter, J. 2014. Towards mindful case study research in IS: A critical analysis of the past ten years. *European Journal of Information Systems*, *23*, 256-272. doi.org/10.1057/ejis.2013.26

Khan, A. S. (2013). *A framework for software system handover*. KTH Royal Institute of Technology

Martin, K. 2019a. Designing ethical algorithms. *MIS Quarterly Executive*, *18*(2), 129-142. doi.org/10.2139/ssrn.3056692

Martin, K. 2019b. Ethical implications and accountability of algorithms. *Journal of Business Ethics*, *160*, 835-850. doi.org/10.1007/s10551-018-3921-3

The MIT License, (1987). www.opensource.org/license/mit/

Mockus, A., Fielding, R. T., & Herbsleb, J. 2000. *A case study of open source software development: the Apache server* Proceedings of the 22nd International Conference on Software Engineering, Ottawa, Canada. doi.org/10.1145/337180.337209

Nidhra, S., Yanamadala, M., Afzal, W., & Torkar, R. 2013. Knowledge transfer challenges and mitigation strategies in global software development - A systematic literature review and industrial validation. *International Journal of Information Management*, *33*(2), 333-355. doi.org/10.1016/j.ijinfomgt.2012.11.004

Novelli, C., Taddeo, M., & Floridi, L. 2023. Accountability in artificial intelligence: what it is and how it works. *AI & Society*, *1*, 1-12. doi.org/10.1007/s00146-023-01635-y

Oreg, S., & Nov, O. 2008. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in Human Behavior*, *24*(5), 2055-2073. doi.org/10.1016/j.chb.2007.09.007

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., & Antiga, L. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*, 1-12. doi.org/10.48550/arXiv.1912.01703

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., & Dubourg, V. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825-2830.

Perens, B. 1999. *The open source definition* (1 ed.). O'Reilly.

Perez-Etchegoyen, J., & Forbes Technology Council. 2023. *Application Security Predictions For 2023*. Forbes. https://www.forbes.com/sites/forbestechcouncil/2023/03/21/application-security-predictions-for-2023/. Accessed: 11/26/2023

Pistilli, G., Muñoz Ferrandis, C., Jernite, Y., & Mitchell, M. 2023. Stronger Together: on the Articulation of Ethical Charters, Legal Tools, and Technical Documentation in ML. In Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency. doi.org/10.1145/3593013.3594002

Red Hat Incorporation. 2018. *CVE-2018-10933*. CVE. https://www.cve.org/CVERecord?id=CVE-2018-10933. Accessed: 11/26/2023

Rolandsson, B., Bergquist, M., & Ljungberg, J. 2011. Open source in the firm: Opening up professional practices of software development. *Research Policy*, *40*(4), 576-587. doi.org/10.1016/j.respol.2010.11.003

Schmidt, J.-H., Bartsch, S. C., & Adam, M. 2023. The Role of Process and Outcome Accountability Claims for Shaping AI Developers' Perceived Accountability. International Conference on Information Systems, Hyderabad, India. www.aisel.aisnet.org/icis2023/isdesign/isdesign/2

Schmidt, J.-H., Bartsch, S. C., Adam, M., & Benlian, A. 2023. *Accountability Incongruence and Its Effects on AI Developers' Job Satisfaction* European Conference on Information Systems, Kristiansand, Norway. aisel.aisnet.org/ecis2023_rp/284. aisel.aisnet.org/ecis2023_rp/284

Schnaack, G., & Termer, F. 2023. *Open-Source-Monitor*. https://www.pwc.de/de/digitale-transformation/bitkom-open-source-monitor-2023.pdf

Seguel, P., & Vaast, E. 2021. Repertories of evaluation in AI ethics: Plurality in professional responsibility and accountability. International Conference on Information Systems, Austin, USA. www.aisel.aisnet.org/icis2021/ai_business/ai_business/8

Sharma, P. N., Daniel, S. L., Chung, T. R., & Grover, V. 2022. A motivation-hygiene model of open source software code contribution and growth. *Journal of the Association for Information Systems*, *23*(1), 165-195. doi.org/10.17705/1jais.00712

Synopsys Incorporation. 2023. *Open Source Security and Risk Analysis Report*. https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2023.pdf

Vance, A., Lowry, P. B., & Eggett, D. 2015. Increasing Accountability Through User-Interface Design Artifacts. *MIS Quarterly*, *39*(2), 345-366. doi.org/10.25300/MISQ/2015/39.2.04

Widder, D. G., Nafus, D., Dabbish, L., & Herbsleb, J. 2022. Limits and possibilities for "Ethical AI" in open source: A study of deepfakes. In Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency, Seoul, South Korea. doi.org/10.1145/3531146.3533779

Wieringa, M. 2020. What to account for when accounting for algorithms: a systematic literature review on algorithmic accountability. In Proceedings of the 2020 ACM Conference on Fairness, Accountability, and Transparency, Barcelona, Spain. doi.org/10.1145/3351095.3372833

Wolter, T., Barcomb, A., Riehle, D., & Harutyunyan, N. 2023. Open source license inconsistencies on github. *ACM Transactions on Software Engineering and Methodology*, *32*(5), 1-23. doi.org/10.1145/3571852

Wu, C.-G., Gerlach, J. H., & Young, C. E. 2007. An empirical analysis of open source software developers' motivations and continuance intentions. *Information & Management*, *44*(3), 253-262. doi.org/10.1016/j.im.2006.12.006

Yang, Z., Wang, C., Shi, J., Hoang, T., Kochhar, P., Lu, Q., Xing, Z., & Lo, D. 2023. What do users ask in open-source AI repositories? An empirical study of GitHub issues. 20th International Conference on Mining Software Repositories (MSR), Melbourne, Australia. doi.org/10.48550/arXiv.2303.09795