

Exploring the Susceptibility to Fraud of Monetary Incentive Mechanisms for Strengthening FOSS Projects

Ben Swierzy^{1,2}[0009–0003–0485–4791], Timo Pohl¹[0009–0002–3760–7976], Marc Ohm^{1,2}[0000–0002–2913–5270], and Michael Meier^{1,2}[0009–0006–8199–5004]

¹ University of Bonn, Germany
`{swierzy,pohl,ohm,mm}@cs.uni-bonn.de`
² Fraunhofer FKIE, Germany

Abstract. Free and open source software (FOSS) is ubiquitous on modern IT systems, accelerating the speed of software engineering over the past decades. With its increasing importance and historical reliance on uncompensated contributions, questions have been raised regarding the continuous maintenance of FOSS and its implications from a security perspective. In recent years, different funding programs have emerged to provide external incentives to reinforce community FOSS’ sustainability. Past research primarily focused on analyses what type of projects have been funded and for what reasons. However, it has neither been considered whether there is a need for such external incentives, nor whether the incentive mechanisms, especially with the development of decentralized approaches, are susceptible to fraud. In this study, we explore the need for funding through a literature review and compare the susceptibility to fraud of centralized and decentralized incentive programs by performing case studies on the Sovereign Tech Fund (STF) and the tea project. We find non-commercial incentives to fill an important gap, ensuring longevity and sustainability of projects. Furthermore, we find the STF to be able to achieve a high resilience against fraud attempts, while tea is highly susceptible to fraud, as evidenced by revelation of an associated sybil attack on npm. Our results imply that special considerations must be taken into account when utilizing quantitative repository metrics regardless whether spoofing is expected.

Keywords: fraud · risk analysis · software metrics · software supply chain · open source · funding.

1 Introduction

During the past decade, free and open source software (FOSS) has established itself as essential in modern digital systems. It is recognized as digital infrastructure, building the foundation for new developments. FOSS is viewed as a digital common good with society benefiting from it. While individuals are able to rapidly bootstrap new projects, developing products has become drastically cheaper for the industry as well. Overall, digital innovation is thriving. [14]

Compared to physical items, companies divest any liability claims for digital goods through mandatory legal agreements and users have no possibility to reliably judge security aspects [21]. In the past, commercial products have been criticized for this. With the majority of systems depending on FOSS, questions are raised on the maintenance and security of these projects. As a solution for the commercial case, areas of transparency are proposed [21] which already apply to FOSS: Openly available code facilitates independent audits, transparent development reveals best practices and open issue trackers disclose problems. While these incentives may work in a commercial context, free-time contributors have only limited capacity for the necessary maintenance. In this scenario, assistance or external incentives can improve the situation: Research has shown that several security practices in FOSS projects benefit from funding [4]. However, this topic is complex and multifaceted. For example, open source stewardship has a crux in finding the balance between private cost and public benefit [17] and fewer users are willing to support stable and established projects as they assume they are healthy and cared for [27].

External incentives can be established in many different ways. Central design questions are the eligibility criteria for funding and the method of allocating budget to eligible projects. As monetary support is the most common type of external incentive, fraud is inevitable if no preventative measures are deployed. The organizational structure (central or decentral) has important implications on possible fraud scenarios. In this work, we explore the susceptibility for fraud of external incentive mechanisms for FOSS projects. We focus on projects which fulfill the open source definition of the Open Source Initiative (OSI) [31] and restrict the scope to community FOSS projects. This confines to projects defined as commercial [6] or single-vendor OSS [39] which are fully backed by a commercial business model. We investigate the following research questions.

- RQ1** Is there a need for external incentive mechanisms for strengthening maintenance of open source projects?
- RQ2** How do processes and properties of centralized and decentralized incentive mechanisms compare?
- RQ3** How susceptible are incentive mechanisms for FOSS projects to fraud?

Accordingly, our contributions are three-fold. We perform a meta study to show the need of FOSS funding without commercial interest, compile practical examples for automatic manipulation of impact metrics, and uncover a sybil attack on npm which could affect future research building on its registry data.

The remainder of the paper is structured as follows. Section 2 discusses whether there is a need for external incentive mechanisms. Section 3 compares centralized and decentralized mechanisms with the help of case studies. The susceptibility to fraud is examined in Section 4. Threats to validity and related work are discussed in Section 5 and 6. We conclude in Section 7.

2 Incentives for Open Source Contributors

There is a plethora of examples of successful and long-living FOSS projects backed by corporate support and even without any external incentives at all. With non-commercial funding gradually moving into the focus of software engineering research, there is little knowledge why such incentives are required. This motivates our first research question. To obtain an answer to this, we performed a keyword-based literature review on the research efforts of the past two decades into understanding the perspectives, motivations and development in FOSS projects. With an initial set of 12 papers obtained through the ACM digital library and DBLP using the keywords *open source maintenance*, we performed backwards snowball sampling to achieve a final collection of 21 papers. As a full systematic literature review is beyond the scope of this paper, we extract the key findings from each paper which contribute arguments to answer RQ1. Finally, all statements are grouped into a coherent order. This approach enables a longitudinal yet nuanced view, better suited to address the statements' diverse nature than quantitative characterizations.

Behind the success stories in FOSS, there is also a significant amount of failed projects. Through an interview study among impactful FOSS project maintainers from GitHub, Coelho et al. [11] compiled reasons why modern FOSS projects fail. Among the top 5 reasons — usurped by a competitor, obsolete, lack of time, lack of interest, and outdated technologies — external incentives can help in the majority of cases. Clearly, funding helps maintainers to invest more time on a project by not requiring them to work a full-time job in parallel. This also allows for necessary refactoring and for overcoming outdated technologies. Furthermore, it can be argued, that external incentives may help core maintainers see perspectives for a project in case commercial competition emerges. These results are in line with further interviews [22], identifying professional activities and financial aspects to be the major reasons why contributors switch from an active to a sleeping or even to a dead state. According to their own views, funding does not only compensate them for their time, but also helps them to dedicate more professional time towards their project [27]. Geer and Sieniawski find long-term commitment to open source stewardship to be essential for project success [17]. However, many projects have only few core developers, making them susceptible to dangers of a low bus factor [54], i.e., the minimum amount of contributors needing to stop working on a project until development halts. External incentives can be targeted towards community building, positively influencing the longevity and maintenance to sustain and grow the group of core developers. In this context, the subgroup within core developers which have administrative rights (called elite developers), show special correlations: While their investment into non-technical activities seems to negatively affect the productivity of a project [51], their time is invaluable for increasing the amount of (core) contributors [13]. Therefore, it is critical that elite developers have as few other professional obligations as possible with external incentives through funding a clear way of achieving this. However, it must be noted that lowering the barriers for participation may invite lots of low quality contributions [27]

which would work against the intended target. Finally, several funding sources have been shown to improve the IT security, measured by increases in many categories on Open Secure Software Foundation (OpenSSF) scorecards [4]. The literature review results only in a single example of negative consequences of funding. Within an experimental initiative, parts of the Debian project tried to shorten release cycles through obtaining short-term sponsorships which created a complex conflict between contributors [18]. Overall, the reviewed research shows clear indications that funding increases quality, maintenance and, accordingly, the reliability and success chances for FOSS projects.

External incentives for FOSS projects, especially through funding, are prevalent for a long time. Already more than 10 years ago, half of the contributors of a project sample make at least 95% of their commits during regular working hours, suggesting they are paid for their contributions [40]. Most existing sponsorships may primarily be classified into individual and corporate sponsoring which exhibit different characteristics. For most projects, individual donors are shown to be more important than corporate donors in the long run [60]. At the same time, corporate donations are more significant than individual ones, which are considered to be a symbol of gratitude, and with neither being considered a sustainable source of income [27]. While these statements seem to partially contradict each other, GitHub’s sponsor mechanism is shown to attract only individual donors, to scale with developer reputation and to only have a short-term effect on the project [56]. Therefore, individual donations seem to be an indicator for a strong project community with corporate sponsorship not achieving a comparable level of self-sustainability. This is backed by the observation that corporate domination shows a negative relationship with survival probability [59]. In addition, corporate involvement is always based on intrinsic motivation which can be subdivided into economic, technological and social dimensions [26]. This leads to prejudices from volunteers towards company-funded contributors which may cause frustration and conflicts [58]. Rust is an example where such an issue occurred, with several core developers leaving because of Amazon’s participation [58]. To work against this, it is suggested that companies are highly transparent with their motivations and contributions [58]. Still, a FOSS project’s sustainability is not in the focus of companies as seen on the example of OpenStack, where many companies withdrew as soon as their goals were achieved or have failed [57]. Further, Gonzalez-Barahona et al. suggest that conflicts may arise when multiple companies contribute to the same project which can lead to unfairness [20], although neither proven nor considered in depth. Tight corporate coupling can lead to highly restrictive contributor license agreements (CLAs), which may ask contributors to surrender rights to their contributions [2]. In addition, the copyleft license GPL sees less prevalence [3]. Overall, the literature review reveals numerous problematic aspects that corporate sponsorship may have on FOSS projects and their sustainability. In contrast, the arguments towards corporate funding are sparse. Capiluppi et al. find project management by a commercial entity to be a major success factor in FOSS [5]. Furthermore, company involvement positively influences project popularity [6]. Though, in the same work indicators

for negative effects on software design quality and changes in governance are found [6].

In conclusion, we can answer RQ1 positively. There are clear indicators that funding is important for the maintenance and longevity of FOSS projects. However, the common scenario of corporate sponsorship may have diverse negative effects on projects, especially, on their sustainability after the sponsorship ends. At the same time, it is rare that donations are able to provide an income above poverty thresholds and maintainers often save them instead of spending them [37]. Consequently, maintenance and sustainability funding driven without commercial motivations needs to be addressed to ensure optimal functioning of the FOSS ecosystem and its dependents.

3 Comparison of Incentive Mechanisms

There exists a variety of financial and organizational support opportunities for FOSS projects. The results of the previous section indicate the need for the special class of non-commercial and structured funding programs for community FOSS. In this section, we take a detailed look at two sub-classes of such funding programs, namely, centralized and decentralized approaches. With respect to the risk of fraud, these face distinct challenges. We compare four essential components for these classes which each program requires: source of budget, eligibility to funding, the application and assessment process and the allocation of budget to applicants. Moreover, the Sovereign Tech Fund (STF) and the tea project are considered as case studies for the purpose of identifying exploitation possibilities by fraudulent actors.

3.1 Centralized Approaches

A funding program shows centralized characteristics if a single instance is in full control of at least one essential component. The component descriptions presented in this section are based on the STF [44], Open Technology Fund (OTF)’s FOSS sustainability fund [33], the Open Source Technology Improvement Fund (OSTIF) [32], and OpenSSF’s Alpha-Omega [49]. All of these are well-known representatives for centralized FOSS funding agencies and transparently document their economics and methods. In all cases, the centrality property exhibits by the programs’ full control over the allocation of budget to interested projects.

Source of Budget Centralized approaches have heterogeneous budget sources which can be split into governmental budget and sponsoring by companies or foundations. In most cases, the approaches are almost exclusively built on a single budget source, with OSTIF being the only exception where each category contributes a significant share. The financial resources of this approach are substantial, with a budget range of 1 to 100 million USD. Notably, this scope is beyond those of individual donors which are at most a negligible source of budget for these approaches.

Eligibility to Funding Before FOSS projects may apply for funding, they must fulfill eligibility criteria. The exact requirements vary based on the political focus of the funding agency. In most cases, projects need to be established and impactful by some definition.

Application and Assessment The eligibility needs to be self-assessed and part of an online application which is connected to a relevant amount of bureaucratic effort. Furthermore, the online application needs to propose activities to be funded. The degree of formality and the amount and design of application stages varies slightly. In special cases, the central party selects funded projects or a subset thereof directly and skips an open application process. Applications are reviewed in a single or multiple stages behind closed doors and their appropriateness is assessed with respect to predefined criteria.

Allocation of Budget In central approaches, an application is either accepted or declined, i.e., either the cost estimate is fully allocated to the applicant or no funding is provided. Partial acceptance is not supported. While the decision is shared with the applicant, feedback is usually sparse [52]. Based on an internal ranking created during assessment, the program funds all projects in order until the budget is exhausted.

Case Study: Sovereign Tech Fund The Sovereign Tech Agency (STA) was established in 2022 by the German government and offers the STF, in addition to other funding programs. Since its fund allocation and its impact are already outlined in prior work [36,41], this case study focuses on the process until funding is approved. If not clearly indicated as presumed, all information in this section is taken from the STA’s website [44].

Source of Budget The STF is completely publicly funded and has an annual budget of 17 million euros in 2024 at its disposal. Sponsorships by companies or foundations are not designed to be part of its budget.

Eligibility to Funding The STF funds established FOSS digital base technologies, i.e., neither prototypes nor user-facing applications. According to own statements, development and maintenance work is primarily funded, but other activities such as security audits may also be encompassed. This aligns with an analysis of STF’s funded projects [41].

Application and Assessment The application itself is a questionnaire where the criticality, use cases, challenges, and planned activities including a cost estimate need to be comprehensively described. The 12 central questions are formulated rather openly which allows an applicant to choose how they want to prove impact, criticality and relevance themselves. After submission, the applications needs to pass three official stages taking up to 6 months in total until a funding contract is finalized. Each of the stages is further subdivided into multiple internal steps with

few details published. In the first step the eligibility is judged, while afterwards the application is rated based on the internal criteria. This is handled by employed *technologists* with one of their tasks explicitly being technology assessment [45]. If this stage is passed, the STF team helps refining the application and concretizing the planned activities. Then, external experts provide additional reviews which are taken into account for the final decision. Additionally, the STA actively scouts FOSS projects for funding.

Allocation of Budget For budget to be allocated to an applicant, all application stages must be passed. While there is no explicit information, it can be inferred from other German funding programs, that STF applications are either completely accepted or declined.

3.2 Decentralized Approaches

A decentralized system is characterized by the absence of a central instance of power. In the context of incentive mechanisms, this manifests either as unstructured sponsorships or the transparent specification of a protocol. While the first lacks governance, the latter is able to provide decision-making capabilities by specifying a domain-specific algorithm or a employing consensus mechanism. Nonetheless, both are characterized by their low barriers for participation.

Source of Budget Compared to central funding systems, the budget sources for decentral systems are more diffuse. In governance-less mechanisms, individuals or companies directly donate to a FOSS project. Therefore, this scenario does not have immediate budget at its disposal. If the construction relies on a protocol, the decentralized funding mechanism may be constructed similarly to crypto currencies. In detail, tokens managed on a distributed ledger are recognized by the community to have a value. As open-source maintainers should monetarily profit from the system, the monetary investments backing the token needs to be sponsored or crowd-sourced. These investments can be incentivized with some sort of power obtainable within the system. While the perspective of profit could also be used to attract investments, this contradicts the concept of moving funds away from investors and towards open-source maintainers.

Eligibility to Funding All packages within supported ecosystems or hosted on supported platforms are deemed eligible. Typically, there is no verification process in place to ascertain the existence of a FOSS license or the availability of the source code.

Application and Assessment The low barriers for participation are particularly evident in the application process. Usually, a simple online registration is the only required step. In addition, decentralized approaches may require a proof of ownership. Although similar to before, this may be as easy as uploading an identifying file to your repository or connecting using an external identity provider such as GitHub.

Allocation of Budget There are two primary ways of budget allocation in decentralized funding: impact metrics and reputation. The first are typically used to split a general budget among participants. Conversely, the latter builds the foundation and motivates funding in direct peer-to-peer scenarios.

Case Study: tea tea is a novel project, aiming to provide incentive mechanisms for FOSS contributions. It builds upon a distributed ledger managing a custom-tailored crypto currency (TEA tokens). Its first implementation called *testnet* was launched in the beginning of 2024. tea consists of multiple processes such as staking tokens on a project, filing bug reports or participating in the decentral autonomous organization (DAO). In this case study, we focus on the subset of these processes which provides monetary incentives for FOSS maintainers. Other incentives share traits with gamification, which is not a type of incentive working towards the need identified in Section 2. If not stated otherwise, all information from this case study is extracted from the tea’s documentation [47].

Source of Budget While TEA token generation and deletion influences the budget within the ecosystem, only the exchange into widely accepted currencies will give the token its value. As tea is not yet in production mode, there is no value and therefore budget yet. For the future, it is planned that monetary value flows into the system through investors, public interest and parties interested in sponsorships [23].

Eligibility to Funding All packages managed by "crates, npm, pkgx, hombrew, pypi, apt-get, and rubygems" [48] are eligible to join. There are no requirements that a package must be FOSS or even source-available.

Application and Assessment For registering a package, its metadata must refer to a GitHub repository URL which must contain a `tea.yaml` constitution file. It works as proof of ownership and connects the project to participants in tea. It must be noted that this repository does not need to contain the sources of the project.

Allocation of Budget Besides providing the possibility for immediate donations of tokens, tea passively allocates budget to maintainers of impactful projects estimated through a metric called teaRank. It is based on the PageRank algorithm, but has been enhanced by additional parameters to combat some conceptual shortcomings. While there is a brief technical description available [48], the implementation is not openly available and details remain unclear. Therefore, we reverse engineer the teaRank calculation to be able to reliably assess the susceptibility to fraud. As the results are technical and do not contribute to answering the research questions, we refrain from presenting them here. Instead, they can be found in Appendix A to facilitate future work on this.

Answering RQ2, we find two major differences between the approaches. First, central programs require more bureaucracy than decentral programs with the latter benefiting from the absence of manual assessment. Second, the funding policies contrast each other. Decentral programs fund either impact or reputation, while central programs focus on funding maintenance work.

4 Incentive Mechanisms' Susceptibility to Fraud

In this section, we critically reflect on the presented incentive mechanisms and identify theoretical fraud scenarios. Afterwards, the spoofability of the underlying metrics are considered before these results are being transferred back to the incentive mechanisms to assess the risk of the fraud scenarios.

4.1 Identification of Fraud Scenarios

The presented fraud scenarios are derived from three models for the fraudster:

- F1** does not maintain a project.
- F2** maintains a semi-impactful project.
- F3** maintains an impactful project.

Centralized Approaches To pass the eligibility stage, F1 is mandated to reference a FOSS project. For this, they can create a new project (C1.1), copy an existing project (C1.2) or impersonate a project by referencing a repository they does not have access to (C1.3). In comparison, F2 honestly passes the eligibility stage but would fall short during assessment. Therefore, they seek to deceive their reviewers (C2). F3 is able to honestly acquire funding but deliberately submits a cost estimate higher than the actual cost (C3). All fraudster models may try to place an insider in the central instance to manipulate internal processes (C-I).

Decentralized Approaches Analog to the centralized approaches, F1 can pursue three scenarios: creating (D1.1), copying (D1.2) or impersonating (D1.3) a project. In contrast, there is no application to be assessed and, thus, F2 and F3 share a joint scenario, in which they try to illegitimately maximize the allocated budget (D2). A large advantage of decentralized approaches is the absence of a central party to be compromised. In practice, decentralized systems are often accessed through a uniform platform which could be susceptible to insider threats. However, as this scenario leans more towards technical IT security instead of fraudulent activities, we disregard it for the remainder of this work.

4.2 Reliability of Impact and Popularity Metrics for Software

Rating the criticality of a project and measuring its impact is an essential step for most centralized and decentralized approaches. It is important to consider

the reliability of impact and popularity metrics for FOSS to be able to assess the likelihood of the presented fraud scenarios. In this section, we analyze 4 existing combined metrics, namely OpenSSF’s criticality score [1], npm’s quality/popularity/maintenance scores, teaRank, and CHAOSS’s project popularity metric [9], by dissecting them into their components, referred to as *atomic metrics*. To cope with subtle differences between some atomic metrics, we categorize them by their underlying data source. All categories are visualized in Figure 1. We discuss their reliability with respect to a project maintainer with write access to a project’s repository contents and its presence on social coding platforms. This covers all fraud scenarios except impersonation (C1.3, D1.3) and insiders (C-I).

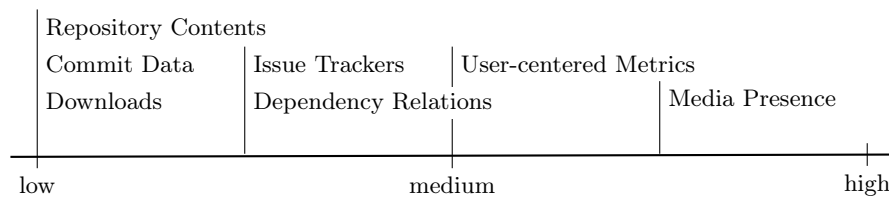


Fig. 1. Spoofing effort for categories of atomic metrics

Commit Data A commit is the essential unit of a version control system (VCS). Active project maintenance implies regular commits. Therefore, atomic metrics derived from commit timestamps are commonly used. However, this type of data is highly unreliable and easily spoofed. Social coding platforms display commit data unaltered. This allows maintainers to create an arbitrary history and push that onto the platform to fulfill all desired metrics. While commits may be signed to verify the identity of the commit author, it does not help in the attacker model of a project maintainer. There are tools exploiting this behavior, for example, the `github-activity-generator` creating a commit history to obtain a custom activity graph in GitHub. It should be noted that the tool explicitly discourages its use to "misrepresent professional contributions or coding activity" [43].

Issue Trackers Many FOSS projects facilitate the issue trackers provided on social coding platforms. While lots of open issues do not imply meaningful statements, an active usage of the issue tracker suggests interest and maintenance of a project. The lifetime of issues offers multiple atomic features to be extracted. Among the associated metrics, the ratio of open to closed issues and the average duration until an issue is closed are prime examples. All popular issue trackers offer APIs for automatic interactions and integration of external clients. A plethora of well-known large projects utilizes bots to assist with the issue management.

Examples are Tensorflow³ which automatically assigns developers, and Go⁴ which has an AI assistant automatically tagging the issue and searching links for related information. React⁵ utilizes an officially provided GitHub Actions workflow to automatically close stale open issues. While we do not assume any malicious motives in this case, this directly boosts an atomic issue metric determining the maintenance score on npm. Other platform features such as publishing explicit releases offer comparable data for atomic metrics. For maintainers, these are as easily spoofable as issue-related data due to public APIs. Therefore, legitimate tools like `semantic-release`⁶ increase the score in associated metrics.

User-centered Metrics Atomic metrics are considered to be user-centered if every user can increase this metric at most by one. They are platform-specific with examples being stars, forks, subscriptions or contributors. Of all categories with implemented measures, user-centered atomic metrics are the most difficult metric to spoof, requiring the attacker to automate account creation and violating the terms of service. Nonetheless, this can be outsourced to paid services. A report [15] describes stars to be successfully purchasable for 0.08 EUR each. However, the automatically created accounts for these are detected and removed along with the stars within a month. In contrast, "quality" stars cost 0.8 EUR each and are backed by accounts not as prone to bot detection. Services to buy forks and subscribers are found as easily. Therefore, given enough monetary or technical resources it seems to be possible to spoof user-centered metrics.

Downloads Impactful projects are utilized by many users and, thus, deployed on many systems. Analyzing recent download counts is a proxy metric suggesting to approximate the deployment count. However, there is no clear connection and a widely deployed project does not need to have many recent downloads. More severely, the process of downloading is easy to automate. For npm, there even exist tools such as `npm-increaser-downloads`⁷ offering an optimized implementation. Though it considerably wastes resources of the package registry, we did not find information on restricted counting, e.g., only once per IP address and day. While this does not defend against such an attack, it significantly increases the effort required by an attacker with only minor overhead for the registry.

Repository Contents The quality metric of npm solely focuses on atomic metrics derived from the contents of a repository such as the existence of a license or the use of linters. The repository contents are trivial to adjust for a maintainer and, thus, it requires low effort to achieve a high score in this metric. Though in this case, we refrain from classifying such adjustments as spoofing, since no information is illegitimately represented.

³ <https://github.com/tensorflow/tensorflow>

⁴ <https://github.com/golang/go>

⁵ <https://github.com/facebook/react>

⁶ <https://github.com/semantic-release/semantic-release>

⁷ <https://github.com/MinhOmega/npm-increaser-downloads>

Media Presence CHAOSS’ project popularity metric [9] enhances its score by incorporating data from sources beyond social coding platforms and package registries. The atomic metrics range from social media mentions over job postings requesting project skills to event participation. Most are challenging to determine automatically and as such, their applicability is likely limited to the largest FOSS projects. Accordingly, these atomic metrics are difficult to spoof as their methods of measurement is not well specified.

Dependency Relations Packages support an effective development process by providing functionality in a re-usable manner. Besides customer-facing end products, this also backs the development of new packages, resulting in a dependency network. This offers a unique view on the impact of projects and is integrated into all considered impact metrics. More specific, either the number of dependent projects or the more complicated and holistic view of teaRank [48] are employed. These metrics can be spoofed by creating bogus packages, referencing the maintainer’s project either directly or transitively as dependency. For uncured package repositories, we deem this attack to require low to medium effort, depending on the efficacy of its automatic spam detection mechanisms.

Sybil Attacks on npm A passive rewarding system such as teaRank has the potential to incentivize maintainers to enhance their impact through dishonest methods. Its technical description [48] acknowledges two types of attacks. Width attacks introduce lots of dependents pointing to a single package. Tree attacks create long dependency chains. It is stated that both attacks are prevented by tracking the width and tree limit of a package and flagging it as potential spam if (secret) thresholds are surpassed. During our work, we manually inspected packages on the tea testnet and found the majority of the projects denoted as most impactful, i.e., having the highest teaRanks, to have either been unpublished or replaced with a security holding package on npm. When metadata was still available, the projects showed thousands of dependents with a record of less than 10 weekly downloads. This is a clear indicator for a sybil attack on teaRank.

To analyze this phenomenon, we examine package metadata from npm. As we try to approximate the order of magnitude for this attack, we employ the following heuristic-based methodology: Initially, all npm packages registered in the tea testnet are classified as sybil if they are published after 2024, have less than 10 published versions and fulfill one of the following three criteria:

- More than 95% of transitive dependencies must have been created within 4 weeks of the creation of the package under consideration.
- More than 80% of dependents must have more than 100 dependencies.
- The package was unpublished or marked as security holding.

In addition, we consider all transitive dependents of a sybil package to be sybil. This methodology is conservative and it can be assumed to be unlikely that a legitimate package is flagged as sybil. Overall, this results in the detection of 71,710 sybil packages on npm (2% of all listed). To confirm the reliability of the

heuristic, we sample 100 potentially sybil packages uniformly at random and manually classify them. We do not find any erroneous classification. By calculating the one-tailed confidence interval for sample proportions, we can be 95% certain that the population contains at most 3% false positives. The sample reveals multiple classes of automatically generated packages (see Figure 2). Partially, the generation scripts can still be found in the artifacts of the packages. Most commonly, boilerplate created by `create-next-app`, a bogus library detectable through `wallet.js` and `chains.js`, an artifact without JavaScript inside or a function returning a string are found. Comparing these with unpublished packages, we assume that these contents prevailed against npm’s spam detection.

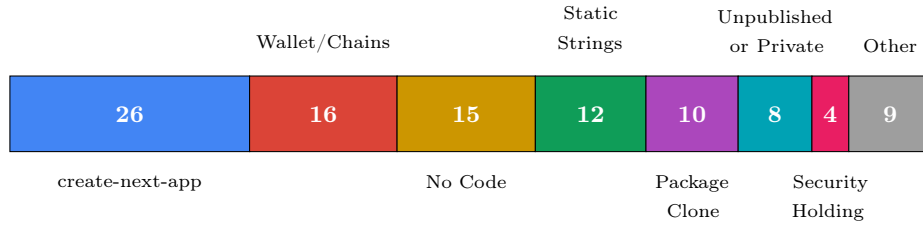


Fig. 2. Classes of sybil packages in a sample of 100 packages

These sybil attacks do not only increase the teaRank of projects but also affect other impact metrics. Past research has considered the top $N \leq 1000$ most depended upon packages as benign packages for evaluating malware protection [16,38], evaluating the adoption of security best practices [24] and others [7]. We find that, at the time of writing, 532 packages of the top 1000 most directly depended upon are in our sybil set. Though when taking transitive relationships into account, no package of the top 1000 most depended upon is marked as sybil by our approach. As a significant fraction of sybil packages was unpublished, the actual figures were likely to be higher in mid 2024. Since tea was introduced at the start of 2024, it is unlikely that the results of the referenced papers are affected by these sybil attacks. Still, this raises questions on the validity of using this or similar impact metrics for software package focused research.

4.3 Assessment of Fraud Scenarios

In this section, we assess the previously defined fraud scenarios by weighing the required effort and risks against the potential gains for the fraudster through a discussion. The assessment focuses on both case studies, since generalization abstracts essential details for an insightful fraud analysis. Nonetheless, we refrain from quantitative characterizations as risk is highly individual.

Sovereign Tech Fund With respect to fraud, the most critical step within the application process of the STF is the assessment of applications. Although the

details of this process are not disclosed, it can be expected that the technologist at least partially confirms the claims from the application while additionally confirming the criticality through a common methodology. Therefore, a successful case of fraud in C1.1 must spoof all considered impact metrics in a consistent way. In C1.2 and C2, this holds only for the subset of metrics which do not attest the necessary criticality. Slight inconsistencies may be fatal and cause a more detailed assessment. It is likely that technologists explicitly try to find dependents deemed to be trustworthy based on their prior experience. In the field of sybil detection, it is a standard assumption that links pointing from a benign to a sybil project are difficult to achieve [19]. Though most severely, the potential analysis of historic development of some quantitative metrics and qualitative metrics requires years of adversarial preparation. Consequently, we expect it to be unlikely that a fraudulent application withstands a thorough manual assessment of both, technologists and external experts.

An easier fraud attempt could be reached through impersonation (C1.3) of an existing impactful project. In this scenario, it is conceivable that all application stages may be passed. However, the funding is paid for achieving milestones which is unrealistic as an impersonator does not have the required access rights for this. While such long payment intervals have been criticized [34], they are an effective mechanism against fraud.

For the other potential weak points, the setup for successful fraud is much more complicated. In C3, the fraudulent applicant must already be responsible for an impactful core technology, overestimate the costs in bad faith and defend the argumentation in the later stages of application. In doing so, they risk being declined and act detrimental towards their own FOSS project. This does not fit the typical motivation and views of core maintainers [27]. In C-I, the fraudster places an insider threat in the STA. This provides a significant advantage to all other scenarios as an inconsistent presentation could be defalcated by the reviewer. However, we claim that the risks and efforts in this scenario clearly outweigh the potential gains, given that the allocation of budget is subject to the perspectives of several individuals.

tea We start by assessing the susceptibility for impersonation (D1.3). Generally, proof of ownership is an effective concept for preventing impersonation. Still, it has two downsides. First, the project’s artifact must be associated with an URL referencing a GitHub repository. Hosting sources on a different or custom platform as well as using a different VCS is not supported. Second, the first months after tea’s testnet launch, hundreds of pull request in popular projects from unrelated users trying to push a `tea.yaml` containing themselves as maintainer have been created [50]. Since then, proof of ownership has been improved by requiring a direct commit of the file, reducing but not eliminating the chance of social engineering. Overall, we estimate this fraud scenario to have low chances of success but it must be noted that effort and risk for trying are low as well.

The fraud scenarios D1.1, D1.2 and D2 share the same success condition, i.e., dishonestly increasing the teaRank as impact metric. While the teaRank can be

increased by becoming a dependency of impactful packages, the fraudster controls none of them. Instead, it requires much less effort to create a lot of new dependent packages as it was done in the sybil attacks on the tea testnet. Most importantly, some persistence is required to keep a project’s growth just below tea’s spam detection thresholds and to deceive malware and spam detection mechanisms of the package registry. Both are achievable with low risk and medium effort through automation and generative AI. Essentially, this is a conceptual flaw of all quantitative impact metrics in this scenario as automatic abuse detection is weaker than automatic abuse, especially, if the detection algorithms are known. Here, Goodhart’s law applies: When a measure becomes a target, it ceases to be a good measure [46]. Overall, the fraud scenarios D1.1, D1.2 and D2 have high chances of success with moderate effort and no risk for the fraudster.

Increasing Resilience against Fraud After identifying the largest risks for fraud in the STF and tea, we propose improvements to increase the resilience against fraud attempts. In the STF, the largest risk is the deception of the reviewing technologist leading to an incorrect assessment of the applicant’s criticality. Therefore, we suggest 4 steps to minimize chances for successful fraud.

1. Target the focus towards the own impact assessment, not on the applicants’ self-assessment.
2. Historic developments of impact metrics require the highest effort for spoofing and should be part of the analysis.
3. Maintain a high awareness for inconsistencies, e.g., many dependents but few downloads.
4. Acquire a set of trust anchors, i.e., confirmed benign projects, and try to obtain as many (transitive) references from the trust anchors to the applicant.

In tea, manipulation of the teaRank bears the largest risk for fraud. Unfortunately, we do not see a possibility to prevent fraud if any of the investigated impact metrics immediately decide the budget allocation. Nonetheless, there are several ideas how a similar incentive mechanism with increased fraud resilience could be designed. First, a distinct impact metric could be designed. All considered impact metrics are, at most, tied to digital identities which can be arbitrarily created. In contrast, binding an impact metric to physical identities does not fully fix the reliability but drastically increases the effort and cost required for spoofing. Second, an equivalent concept to trust-on-first-use can be employed to create a central invariant: At any point in time, benign projects have a voting power surpassing the power of sybil projects. If the growth of nodes is bounded in each time step, the benign projects can actively defend their majority by excluding sybil packages. Moreover, this concept allows a shift of the allocation policy from impact-based to work-based seen in centralized systems by facilitating votes on proposals. Although it must be noted, that such an approach places a burden of work on FOSS maintainers and, in this basic form, will likely fail to strengthen the maintenance level in FOSS.

Answering RQ3, the STF as a representative for central approaches has a low susceptibility to fraud if a thorough assessment of applications is performed. In contrast, our decentral case study tea appears to be susceptible to fraud since the underlying impact metrics are easily spoofable. All ideas to inherently improve its resilience of the approach require conceptual adjustments.

5 Threats to Validity

The highly empirical nature of research presented in this paper requires a discussion of its validity. RQ1 is answered based on a literature review. The results are one-sided and raise concerns on the internal validity. While the literature review does not follow all steps necessary for a systematic literature review, we argue that the most important methodological steps are employed and the methodology does not imply a bias. Instead, the results are explained by the absence of research on the benefits of commercial involvement for FOSS projects or, indeed, by the limited benefits that commercial involvement does offer. To a large extent, the analyses for RQ2 and RQ3 are based on case studies which is a potential threat to external validity. For the centralized approach, we argue that the STF is a good representative sharing traits with other instances. In the decentralized approach, tea is a unique concept and does not represent other decentral instances. However, to the best of our knowledge only reputation-based mechanisms do also qualify as decentral with their susceptibility to fraud being mainly prone to psychological attacks such as phishing. We minimize associated threats to validity by meticulously examining tea-specific statements before generalizing them. Still, it is imperative to carefully reflect case study related results when employing them in future work.

6 Related Work

Fraud facilitates similar techniques as attacks on IT systems. These techniques can be divided into social engineering, architectural and technical attacks. Zaoui et al. [55] compile a taxonomy for attacks and countermeasures of social engineering attacks. These are complementary to our fraud scenarios and proposed resilience strategies. More fundamentally, Longtchi et al. [28] analyze the underlying psychological factors of these attacks. We similarly include the motivations driving different groups in our argumentations.

For related architectural attacks, there is research on sybil attacks and sybil-resistant architectures. Similar to us, Müller et al. [30] propose binding digital to physical identities to reduce the susceptibility to sybil attacks. Cheng and Friedman [10] proof that it is not possible to construct a symmetric sybilproofness reputation function for decentral networks, i.e., only considering the edges. This result is in line with our observations for sybil attacks on teaRank. Additionally, they show sybilproofness to be achievable for an asymmetric reputation function,

e.g., with respect to a given node. In a related way, we suggest to increase the resilience in central approaches by following packages back to trust anchors through dependency relationships.

In this study, we have opted to exclude commercial open source products from our scope, given their distinct funding models and governance structures. Still, related work performs in-depth analysis of their motivations to explain their seemingly contradictory business model. West and Gallagher [53] reveal commercial FOSS to be driven strategically and provide possibilities to maximize the returns of internal innovation. For example, software components may be donated to provide an extensible platform for external contributors. Osborne [35] considers this strategy in detail for FOSS from the realm of artificial intelligence. They find the most important factor to be the governance democratisation for technological and economic advantages. While outside our scope, such motivations are also responsible for companies and investors supporting FOSS maintenance initiatives beyond their sphere of authority.

Last, the design and analysis of impact and popularity metrics is related. Mujahid et al. [29] observe limitations of current popularity metrics such as stars and downloads and instead suggest to employ package centrality. They successfully apply this approach to identify npm packages in decline which is a criterion to prefer alternative dependencies. Coelho et al. [12] present a machine learning model as a metric for measuring the maintenance status of GitHub software projects. As the input features are covered by the atomic metrics analyzed in this work, it is susceptible to the same degree of spoofing. In Section 4.2, the CHAOSS project popularity metric is considered. Besides that, CHAOSS [8] recently initiated a working group to develop methods for measuring funding impact. Similarly, Osborne [35] develops a toolkit for measuring the impact of public funding on FOSS. They argue that quantitative data has a "risk of creating perverse incentives through metric selection/optimisation" and suggests qualitative and mixed-methods to capture social, economic and technological impact. This is in line with our results that budget allocation based on quantitative impact metrics is highly susceptible to fraud.

7 Conclusion

In this study, we explored external incentive mechanisms for FOSS projects and their susceptibility to fraud. A comprehensive literature review reveals a clear necessity for incentive mechanisms to fund the maintenance of FOSS. Moreover, commercial funding can negatively affect the long-term self-sustainability FOSS projects, stressing the importance of alternative funds. We analyzed two structurally distinct approaches aiming to fill this gap. Central incentive mechanisms are backed by significant financial resources which are allocated on work packages proposed by applicants. On a case study of the STF, it is observed that applications are thoroughly reviewed and assessed before funding is approved.

Conversely, the sources of budget for decentral incentive mechanisms are more diverse and opaque, typically relying on individual or commercial investments.

Though, the largest difference is observed for the funding policy with either impact or reputation being funded but not maintenance work. The case study on tea reveals a high risk of fraud due to the automatic allocation of budget through teaRank. Intuitively, it might be clear that all considered atomic metrics are theoretically spoofable. However, empirical evidence demonstrates the occurrence of spoofing also in practice. This renders it very complex if not impossible to create an impact metric suitable for this use case. Possible solutions require architectural changes such as impact metrics linked with physical identities or a voting-based system. Furthermore, there are also implications for academic research utilizing these metrics for dataset creation as shown on the example of a sybil attack on npm. We advocate to consider perils of mining software repositories [25] and discuss the implications of metrics [42] more thoroughly to increase the general significance of research results. In contrast, the STF shows more potential for resilience against fraud. If the impact assessment during the application process is thorough, there is a low risk of successful fraud.

We identify two major directions for future work based on our results. First, a compilation of quantitative data measured in regular intervals could be utilized to detect sybil attacks on package repositories. While, at first, this may sound contradictory to our result that all considered atomic metrics can be spoofed, we do not expect this to happen for no reason. Sybil attacks on repositories as observed in this work are usually targeted and only focus on the subset of relevant metrics. This enhances the probability of detection by an untargeted approach. Second, package repositories are popular data sources for academic research due to their size and (semi-)structured data. With the occurrence of sybil attacks on this dataset, the results' robustness in the presence of an attack is a novel and relevant research direction. For example, it is intuitively unclear how significant the performance of malware detection systems on npm degrades if they are partially trained on bogus packages.

A teaRank

Inheriting the fundamental concept from PageRank, teaRank is built upon random walks in a Markov chain. The resulting scores represent the probability that a walk ends in a given state. For this, the dependency network is considered to be a graph $G = (V, E)$ with packages $V = \{p_1, p_2, \dots, p_n\}$ as vertices and directed edges $E = \{(p_i, p_j), \dots\}$ induced between packages if p_i depends on p_j . The translated adjacency matrix of this graph $A = (a_{ji})$ with $a_{ji} = \frac{1}{|E(i)|}$, $E(i) = \{e \in E | e = (p_i, *)\}$ if $(p_i, p_j) \in E$ is the central element of the calculation. Naturally speaking, all outgoing edges of a package have the same weight with each column summing to one if at least a single outgoing edge exists. teaRank utilizes a modified version of this matrix by introducing a parameter $0 \leq \kappa \leq 1$:

$$T = (1 - \kappa)A + \kappa \mathbf{1}_n$$

This adds κ -weighted self-edges to every vertex. Furthermore, PageRank has a decay factor $0 \leq d \leq 1$ which can be interpreted as a restart probability of the

random walk. This factor is essential for the algorithm as G may be partitioned. In the algorithm, d represents a scaling factor for the uniform distribution $E = (\frac{1}{n}, \dots, \frac{1}{n})^\top$.

Given a probability distribution vector v , the multiplication Tv denotes the transition in the Markov chain. A distribution vector v is considered stable if it is not altered by such a transition, i.e., $(1 - d)Tv + dE = v$. To find such an eigenvector, the power iteration method is often used: Starting with any vector v_0 , the series $v_k = (1 - d)Tv_{k-1} + dE$ converges to a solution v . Despite a potentially large n , the algorithm is efficient since T is sparse and an iteration count in the order of 50 usually suffices to achieve a stable result. The resulting eigenvector v contains entries for all packages in the dependency graph. The web interface for the tea ecosystem does not display the raw teaRank, but applies the following function obtained through reverse engineering the frontend:

$$f(t) = 100 \cdot \left(\frac{\log_{10} t}{9} + 1 \right)$$

To check how well the described algorithm's output matches the real teaRank, we calculate the mean multiplicative error. As the concrete values for κ and d are not disclosed, we perform a grid search with a granularity of 0.05. We observe combinations that roughly fulfill the equation $d \approx 0.6 + \frac{\kappa}{3}$ to minimize the error with a value of 2.4. The minor deviation of our results to the project's scores are explainable by deviations in the underlying graph. tea's testnet supports multiple registries besides npm, although with 95% the clear majority of packages on tea is covered by our analysis.

References

1. Arya, A., Brown, C., Pike, R., The Open Source Security Foundation: Open Source Project Criticality Score, https://github.com/ossf/criticality_score
2. Birkinbine, B.J.: Incorporating the Digital Commons: Corporate Involvement in Free and Open Source Software. University of Westminster Press. <https://doi.org/10.16997/book39>
3. Bonaccorsi, A., Lorenzi, D., Merito, M., Rossi, C.: Business Firms' Engagement in Community Projects. Empirical Evidence and Further Developments of the Research. In: First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007). IEEE. <https://doi.org/10.1109/floss.2007.3>
4. Brackett, S.A., Meyers, J.S., Scott, S.: O\$\$ security: Does more money for open source software mean better security? A proof of concept
5. Capiluppi, A., Stol, K.J., Boldyreff, C.: Exploring the Role of Commercial Stakeholders in Open Source Software Evolution, pp. 178–200. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33442-9_12
6. Capra, E., Francalanci, C., Merlo, F., Rossi-Lamastra, C.: Firms' involvement in open source projects: A trade-off between software structural quality and popularity. *Journal of Systems and Software* **84**(1), 144–161 (2011). <https://doi.org/https://doi.org/10.1016/j.jss.2010.09.004>, information Networking and Software Services

7. Chao, J., Tao, S., Ribbink, A.: Evaluating the evaluators: On package scores and their underlying metrics
8. CHAOSS: Funding Impact Measurement Working Group, <https://github.com/chaoss/wg-funding-impact>
9. CHAOSS: Project Popularity, <https://chaoss.community/kb/metric-project-popularity/>
10. Cheng, A., Friedman, E.: Sybilproof reputation mechanisms. In: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems - P2PECON '05. p. 128. P2PECON '05, ACM Press. <https://doi.org/10.1145/1080192.1080202>
11. Coelho, J., Valente, M.T.: Why modern open source projects fail. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 186–196. ESEC/FSE'17, ACM. <https://doi.org/10.1145/3106237.3106246>
12. Coelho, J., Valente, M.T., Milen, L., Silva, L.L.: Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects **122**, 106274. <https://doi.org/10.1016/J.INFSOF.2020.106274>, <https://doi.org/10.1016/j.infsf.2020.106274>
13. Coelho, J., Valente, M.T., Silva, L.L., Hora, A.: Why we engage in FLOSS: answers from core developers. In: Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering. ICSE '18, ACM. <https://doi.org/10.1145/3195836.3195848>
14. Eghbal, N.: Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure, <https://www.fordfoundation.org/work/learning/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/>
15. Eldeeb, Y., Sikora, A.: How Much Are GitHub Stars Worth to You?, <https://the-guild.dev/blog/judging-open-source-by-github-stars>
16. Ferreira, G., Jia, L., Sunshine, J., Kastner, C.: Containing malicious package updates in npm with a lightweight permission system. In: IEEE/ACM 43rd International Conference on Software Engineering. pp. 1334–1346. IEEE. <https://doi.org/10.1109/icse43902.2021.00121>
17. Geer, D., Sieniawski, G.P.: Who Will Pay the Piper for Open Source Software Maintenance? Can We Increase Reliability as We Increase Reliance? **45**(2), <https://www.usenix.org/publications/login/summer2020/geer>
18. Gerlach, J.H., Wu, C.G., Cunningham, L.F., Young, C.E.: An Exploratory Study of Conflict over Paying Debian Developers **7**(3), 20–38. <https://doi.org/10.4018/ijossp.2016070102>
19. Gong, N.Z., Frank, M., Mittal, P.: Sybilbelief: A semi-supervised learning approach for structure-based sybil detection **9**(6), 976–987. <https://doi.org/10.1109/tifs.2014.2316975>
20. Gonzalez-Barahona, J.M., Izquierdo-Cortazar, D., Maffulli, S., Robles, G.: Understanding How Companies Interact with Free Software Communities **30**(5), 38–45. <https://doi.org/10.1109/ms.2013.95>
21. Halderman, J.A.: To Strengthen Security, Change Developers' Incentives **8**(2), 79–82. <https://doi.org/10.1109/MSP.2010.85>
22. Iaffaldano, G., Steinmacher, I., Calefato, F., Gerosa, M., Lanubile, F.: Why do developers take breaks from contributing to OSS projects? a preliminary analysis. In: Proceedings of the 2nd International Workshop on Software Health. p. 9–16. SoHeal '19, IEEE Press. <https://doi.org/10.1109/SoHeal.2019.00009>, <https://doi.org/10.1109/SoHeal.2019.00009>
23. Joslyn, H.: Is crypto the solution to paying open source developers?, <https://thenewstack.io/is-crypto-the-solution-to-paying-open-source-developers/>

24. Kabir, M.M.A., Wang, Y., Yao, D., Meng, N.: How do developers follow security-relevant best practices when using npm packages? In: 2022 IEEE Secure Development Conference (SecDev). IEEE. <https://doi.org/10.1109/secdev53368.2022.00027>
25. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: An in-depth study of the promises and perils of mining GitHub **21**(5), 2035–2071. <https://doi.org/10.1007/s10664-015-9393-5>
26. Li, X., Zhang, Y., Osborne, C., Zhou, M., Jin, Z., Liu, H.: Systematic Literature Review of Commercial Participation in Open Source Software **34**(2), 1–31. <https://doi.org/10.1145/3690632>
27. Linäker, J., Link, G., Lombard, K.: Sustaining Maintenance Labor for Healthy Open Source Software Projects through Human Infrastructure: A Maintainer Perspective. In: Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 37–48. ESEM '24, ACM. <https://doi.org/10.1145/3674805.3686667>
28. Longtchi, T.T., Rodriguez, R.M., Al-Shawaf, L., Atyabi, A., Xu, S.: Internet-based social engineering psychology, attacks, and defenses: A survey **112**(3), 210–246. <https://doi.org/10.1109/jproc.2024.3379855>
29. Mujahid, S., Costa, D.E., Abdalkareem, R., Shihab, E., Saied, M.A., Adams, B.: Toward Using Package Centrality Trend to Identify Packages in Decline **69**(6), 3618–3632. <https://doi.org/10.1109/tem.2021.3122012>, <http://dx.doi.org/10.1109/TEM.2021.3122012>
30. Müller, W., Plötz, H., Redlich, J.P., Shiraki, T.: Sybil proof anonymous reputation management. In: Proceedings of the 4th international conference on Security and privacy in communication networks. pp. 1–10. Securecomm08, ACM. <https://doi.org/10.1145/1460877.1460887>
31. Open Source Initiative: The open source definition, <https://opensource.org/osd>
32. Open Source Technology Improvement Fund: Open source technology improvement fund, <https://ostif.org>
33. Open Technology Fund: Localization labfree and open source software sustainability fund, <https://www.opentech.fund/funds/free-and-open-source-software-sustainability-fund/>
34. Osborne, C.: Open Source Software Developers' Views on Public and Private Funding: A Case Study on scikit-learn. In: 2024 Conference on Computer-Supported Cooperative Work and Social Computing. pp. 154–161. CSCW '24, ACM. <https://doi.org/10.1145/3678884.3681844>
35. Osborne, C.: Why Companies "Democratise" Artificial Intelligence: The Case of Open Source Software Donations
36. Osborne, C., Sharratt, P., Foster, D., Boehm, M.: A Toolkit for Measuring the Impacts of Public Funding on Open Source Software Development
37. Overney, C., Meinicke, J., Kästner, C., Vasilescu, B.: How to not get rich: an empirical study of donations in open source. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. ICSE '20, ACM. <https://doi.org/10.1145/3377811.3380410>
38. Pohl, T., Ohm, M., Boes, F., Meier, M.: You can run but you can't hide: Runtime protection against malicious package updates for node.js. In: Sicherheit 2024. pp. 231–241. Gesellschaft für Informatik e.V., Bonn (2024). https://doi.org/10.18420/sicherheit2024_015
39. Riehle, D.: The Economic Case for Open Source Foundations **43**(1), 86–90. <https://doi.org/10.1109/mc.2010.24>

40. Riehle, D., Riemer, P., Kolassa, C., Schmidt, M.: Paid vs. Volunteer Work in Open Source. In: 47th Hawaii International Conference on System Sciences. IEEE. <https://doi.org/10.1109/hicss.2014.407>
41. Ruohonen, J., Choudhary, G., Alami, A.: An Overview of Cyber Security Funding for Open Source Software
42. Sheoran, J., Blincoc, K., Kalliamvakou, E., Damian, D., Ell, J.: Understanding “watchers” on GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 336–339. ICSE ’14, ACM. <https://doi.org/10.1145/2597073.2597114>
43. Shpota, S.: Github activity generator, <https://github.com/Shpota/github-activity-generator>
44. Sovereign Tech Agency: Sovereign tech fund, <https://www.sovereign.tech/programs/fund>
45. Sovereign Tech Agency: Technologist, <https://www.sovereign.tech/jobs/technologist>
46. Strathern, M.: ‘improving ratings’: audit in the british university system **5**(3), 305–321. [https://doi.org/10.1002/\(sici\)1234-981x\(199707\)5:3<305::aid-euro184>3.0.co;2-4](https://doi.org/10.1002/(sici)1234-981x(199707)5:3<305::aid-euro184>3.0.co;2-4), [https://doi.org/10.1002/\(sici\)1234-981x\(199707\)5:3<305::aid-euro184>3.0.co;2-4](https://doi.org/10.1002/(sici)1234-981x(199707)5:3<305::aid-euro184>3.0.co;2-4)
47. tea Association: Tea documentation, <https://docs.tea.xyz/tea>
48. tea Association: What is Proof of Contribution? (technical), <https://docs.tea.xyz/tea/i-want-to.../learn-about-proof-of-contribution/what-is-proof-of-contribution-technical>
49. The Linux Foundation: Alpha-omega, <https://alpha-omega.dev>
50. Tumbleson, C.: The disappointing tea.xyz, <https://connortumbleson.com/2024/02/26/the-disappointing-tea-xyz/>
51. Wang, Z., Feng, Y., Wang, Y., Jones, J.A., Redmiles, D.: Unveiling Elite Developers’ Activities in Open Source Projects **29**(3), 1–35. <https://doi.org/10.1145/3387111>
52. Warren, E.: Foss sustainability fund 2024: the grant proposal is declined, <https://codeberg.org/forgejo/sustainability/pulls/48>
53. West, J., Gallagher, S.: Challenges of open innovation: the paradox of firm investment in open-source software **36**(3), 319–331. <https://doi.org/10.1111/j.1467-9310.2006.00436.x>
54. Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A.E., Ubayashi, N.: Revisiting the applicability of the pareto principle to core development teams in open source software projects. In: Proceedings of the 14th International Workshop on Principles of Software Evolution. ESEC/FSE’15, ACM. <https://doi.org/10.1145/2804360.2804366>
55. Zaoui, M., Yousra, B., Yassine, S., Yassine, M., Karim, O.: A comprehensive taxonomy of social engineering attacks and defense mechanisms: Toward effective mitigation strategies **12**, 72224–72241. <https://doi.org/10.1109/access.2024.3403197>
56. Zhang, X., Wang, T., Yu, Y., Zeng, Q., Li, Z., Wang, H.: Who, what, why and how? towards the monetary incentive in crowd collaboration: A case study of github’s sponsor mechanism. In: CHI Conference on Human Factors in Computing Systems. pp. 1–18. CHI ’22, ACM. <https://doi.org/10.1145/3491102.3501822>
57. Zhang, Y., Liu, H., Tan, X., Zhou, M., Jin, Z., Zhu, J.: Turnover of Companies in OpenStack: Prevalence and Rationale **31**(4), 1–24. <https://doi.org/10.1145/3510849>
58. Zhang, Y., Qin, M., Stol, K.J., Zhou, M., Liu, H.: How Are Paid and Volunteer Open Source Developers Different? A Study of the Rust Project. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. pp. 1–13. ICSE ’24, ACM. <https://doi.org/10.1145/3597503.3639197>

59. Zhang, Y., Stol, K.J., Liu, H., Zhou, M.: Corporate dominance in open source ecosystems: a case study of OpenStack. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1048–1060. ESEC/FSE '22, ACM. <https://doi.org/10.1145/3540250.3549117>
60. Zhou, J., Wang, S., Kamei, Y., Hassan, A.E., Ubayashi, N.: Studying donations and their expenses in open source projects: a case study of GitHub projects collecting donations through open collectives **27**(1). <https://doi.org/10.1007/s10664-021-10060-y>