Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Incentivizing Crowds and Intergroup Spillovers: Evidence from the Internet Bug Bounty Program

Jiali Zhou, Kai-Lung Hui, Ohchan Kwon

To cite this article:
Jiali Zhou, Kai-Lung Hui, Ohchan Kwon (2026) Incentivizing Crowds and Intergroup Spillovers: Evidence from the Internet Bug Bounty Program. Management Science

Published online in Articles in Advance 04 Aug 2026

. https://doi.org/10.1287/mnsc.2024.04544

https://pubsonline.informs.org/journal/mnsc

MANAGEMENT SCIENCE
Articles in Advance, pp. 1–26
ISSN 0025-1909 (print), ISSN 1526-5501 (online)

Incentivizing Crowds and Intergroup Spillovers: Evidence from the Internet Bug Bounty Program

Jiali Zhou,[a,*] **Kai-Lung Hui,**[b] **Ohchan Kwon**[b]

[a] Kogod School of Business, American University, Washington, District of Columbia 20016; [b] School of Business and Management, The Hong Kong University of Science and Technology, Hong Kong

*Corresponding author

jializ@american.edu (J. Zhou), klhui@ust.hk (K.-L. Hui), ohchankw@ust.hk (O. Kwon)

https://orcid.org/0000-0002-8678-5966 (J. Zhou), https://orcid.org/0000-0002-7074-1176 (K.-L. Hui), https://orcid.org/0000-0002-8837-065X (O. Kwon)

Received: January 18, 2024
Revised: April 28, 2025; December 8, 2025
Accepted: March 10, 2026
Published Online in Articles in Advance: August 4, 2026

https://doi.org/10.1287/mnsc.2024.04544

Abstract. Online communities can create complex knowledge products, such as software, by leveraging collaboration and division of labor between their core members and the crowd across tasks. Recent literature has focused on how incentives stimulate crowd contributions. Limited attention has been paid to potential *intergroup spillovers*, that is, how incentivizing crowd contributions affects nonincentivized core members' contributions. In this paper, we examine how core members of the Python developer community responded to the Internet Bug Bounty program, an initiative that offers monetary incentives for bug reporting by the crowd. Using a difference-in-differences strategy and the Java community as the control group, we find that the Python core members reduced their contributions to the incentivized task of bug reporting after the program's implementation. Importantly, they also reduced their contributions to the nonincentivized task of enhancement reporting, a critical knowledge creation task for improving Python. Such reduction has led to a decline in the overall contributions of enhancement reports and related code commits to Python. Further analysis suggests that the negative intergroup spillover of the incentive on core members' nonincentivized task contributions can plausibly be explained by reduced intertask learning. Our study offers implications for crowdsourcing, the design of incentives for digital public goods, and task delegation.

History: Accepted by Anindya Ghose, information systems.
Funding: This research is supported in part by the HKSAR General Research Fund Project 16503620.
Supplemental Material: The online appendix and data files are available at https://doi.org/10.1287/mnsc.2024.04544.

Keywords: crowdsourcing • online community • learning • complementarity • bug bounty program • open-source software

1. Introduction

Developing complex knowledge products, such as software, involves coordinating many individuals and different knowledge inputs (MacCormack et al. 2001). Although firms have traditionally acted as the primary agents for organizing such complex processes, advances in digital platforms have given rise to an open and distributed community model of production (Baldwin and von Hippel 2011). This alternative approach has been widely used in open-source software (OSS) development, such as Linux (Raymond 1999), and has generated significant economic value and productivity gains (Nagle 2018, Hoffman et al. 2024).

Communities often exhibit a division of labor between their core members and the broader crowd across different empirical contexts (Lee and Cole 2003, Shah 2006). For example, in OSS development, a small group of core developers, also known as maintainers, coordinate and engage in various tasks to drive the development process, playing a critical role in shaping and sustaining the software. With broad expertise across different domains, these core developers are better positioned to solve complex and interdependent problems (Fleming and Waguespack 2007). Meanwhile, a much larger group of crowd contributors makes sporadic contributions and brings diverse perspectives to support the core members' work. Together, these communities develop complex knowledge products by combining the diversity and prowess of crowd participation with the commitment and expertise of core members.

To harness the potential of community-based production, a widely discussed approach in the academic literature and among practitioners is to promote crowd

contributions by offering incentives. This approach aligns with the optimistic perspective that "given a large enough beta-tester and codeveloper base, almost every problem will be characterized quickly and the fix [will be] obvious to someone" (Raymond 1999, p. 1). Attracting more crowd contributions can bring benefits, such as reduced bias (Greenstein and Zhu 2012), improved product quality (Nagaraj 2021), and enhanced reliability (Zhou and Hui 2019).

The literature has examined how incentives stimulate the crowd's contributions to incentivized and nonincentivized tasks (e.g., Gallus 2017, Wang et al. 2022). Much less is known about potential *intergroup spillovers*: how incentivizing crowd contributions affects nonincentivized core members' contributions to knowledge creation. On the one hand, more crowd contributions may reduce the burden of core members, allowing them to shift their attention and effort to other nonincentivized knowledge creation tasks (Holmstrom and Milgrom 1991). On the other hand, core members may need to expend more effort integrating the increasing crowd contributions, outsourcing tasks may hinder the knowledge flow necessary for further innovations (Reitzig and Wagner 2010), and incentives may induce counterproductive psychological effects (e.g., Kuang et al. 2019). Understanding the potential interactions between the crowd's and core members' contributions is key to successful utilization of the community production model. For example, if soliciting crowd contributions impedes the productivity and work output of core community members, then we must use this approach carefully and delicately to ensure optimal work outcomes.

In this paper, we study how incentivizing crowd contributions affects core members' knowledge creation by using field data from a quasi-natural experiment in the open-source Python community. Python has been supported by two groups of contributors, namely, the official maintenance team (core members, henceforth referred to as "maintainers") and the public unaffiliated with Python (crowd contributors, henceforth referred to as "peripheral contributors"). Without a firm dictating the development, Python relies on these contributors to continuously and openly contribute new ideas on software bugs and feature enhancements. Contributors use bug reports to inform the community about bugs in the Python software and enhancement reports to propose new features and improvements in Python.

We exploit a unique information security crowdsourcing initiative—the Internet Bug Bounty (IBB) program—to identify the causal impact of incentivizing crowd contributions on core members' knowledge creation. In November 2013, Microsoft, Facebook, and HackerOne initiated the IBB to solicit vulnerability identification for Python by providing monetary rewards, or "bug bounty," to crowd contributors who successfully report unknown security vulnerabilities (i.e., bug reports). Unlike peripheral contributors, the Python maintainers, despite being the most active contributors of bug reports, do not claim bug bounty rewards because of conflict of interest.

The IBB offers three unique advantages for our identification strategy. First, the decision to include Python in the IBB was made externally and was unexpected to the Python maintainers. Therefore, the introduction of the IBB is exogenous, facilitating the identification of its causal impacts. Second, the IBB provides incentives to bug reporting but not enhancement reporting, allowing us to examine the potential spillover of the IBB on knowledge creation tasks where the IBB incentive does not apply. Third, the IBB incentive applies to peripheral contributors but not maintainers, allowing us to identify how incentivizing crowd contributions affects nonincentivized core members' contributions. This third feature is our key point of departure from the literature.

We use empirical data to examine how the IBB affects Python maintainers' knowledge creation. Our identification strategy is difference-in-differences (DID), employing the maintainers of another widely used open-source project not treated by IBB, OpenJDK (the open-source implementation of Java; henceforth referred to as "Java"), as the control group. Our data comprise monthly bug and enhancement reports submitted by Python and Java maintainers and peripheral contributors from January 2012 to December 2019, which cover the pre-IBB and post-IBB periods. We can observe the IBB's long-term impacts given that this program had been operating in the Python community for more than six years before our data collection.

We find that, relative to Java maintainers, Python maintainers' contributions of bug and enhancement reports reduced significantly after the introduction of the IBB. This reduction was associated with a decline in their contributed code commits. However, the IBB did not lead to significant changes in report or commit quality as measured by report length, valid report ratio, ratio of reports including patches, patch length, and commit length or complexity. We present evidence that this decrease may be attributed to two factors: increased difficulty for Python maintainers to find bugs and the complementarity between bug reporting and enhancement reporting, which could arise from *intertask learning*—Python maintainers learn from the bug reporting process to facilitate their enhancement reporting (Lindbeck and Snower 2000, Schilling et al. 2003). Accordingly, reduced bug reporting could negatively affect their enhancement reporting.

Our findings hold in view of several alternative explanations, including the loss of interest in Python among Python maintainers, an excessive number of

reports for Python maintainers to handle after the introduction of the IBB, substitution of crowd enhancement reports, misclassified enhancement reports, and the use of Java as an inappropriate control group. Finally, we find that, although the IBB has an overall positive security impact by making bugs more difficult to find, the overall contributions of enhancement reports and enhancement-related commits to Python have reduced after the introduction of the IBB.

Our finding that incentivizing crowd contributions affects core members' contributions in broader knowledge creation tasks is of particular interest because leveraging crowd capability is a common way of enhancing community-based knowledge production. Although crowd solutions can be transferred to supplement core members' work, the underlying knowledge that drives these crowd solutions may be sticky and tacit (Reber 1989, von Hippel 1994). Relying on crowd solutions in place of core members' own problem solving could limit core members' learning of the underlying knowledge and, subsequently, impair their performance. This finding highlights learning opportunities as an important consideration in task delegation.

Our study contributes to the literature and practice in three important ways. First, we characterize the nontrivial trade-offs between the contributions of crowds and core members in crowdsourcing. Previous studies have emphasized the benefits of soliciting crowd contributions, such as improving the quantity and quality of the crowdsourced tasks and potentially *increasing the crowd's* nonincentivized contributions (e.g., Terwiesch and Xu 2008, Boudreau et al. 2011, Wang et al. 2022). Our results offer a novel perspective by showing that incentivizing crowd contributions can generate a *negative intergroup* spillover effect on core members' contributions, potentially *reducing the overall* nonincentivized contributions. This negative intergroup spillover on core members' noncrowdsourced tasks counters the conventional wisdom that outsourcing a task would allow an individual to work better in other tasks (e.g., Quinn and Hilmer 1994).

Second, to our knowledge, we are among the first to provide empirical evidence on how incentive provision affects OSS contributions. OSS underpins modern IT infrastructure. Unlike other online content, such as reviews and Q&As, OSS requires technically complex contributions (e.g., software issue reports and code commits) across multiple knowledge domains (e.g., security and feature enhancements) and relies on sustained participation from both core maintainers and a wider community of contributors (Lee and Cole 2003, Dahlander and O'Mahoney 2011). It is unclear whether findings on incentive effects in other online settings generalize to OSS (e.g., Gallus 2017, Burtch et al. 2018). Our findings of spillovers on OSS maintainers and nonincentivized contributions highlight the need to observe caution when implementing OSS incentive policies that target specific individuals or tasks.[1]

Third, we advance the literature on performance-contingent rewards (i.e., rewarding top performers) by highlighting their side effects. Research has emphasized their advantage over completion-contingent rewards (i.e., rewarding anyone who completes the task) in strengthening the recipients' motivation and performance (e.g., Wang et al. 2012, Yu et al. 2022). We show that performance-contingent rewards can intensify competition, potentially reducing contributions from noneligible contributors. Importantly, in complex knowledge production settings where task complementarity and intertask learning are imperative, noneligible users' reduced contributions to the incentivized tasks may impede their nonincentivized task contributions, further dampening their outputs. These insights introduce new considerations for the use of performance-contingent rewards in complex knowledge production settings.

2. Theoretical Background

2.1. Related Literature

This study advances the literature on how incentives stimulate contributions to digital public goods. Prior research has examined how contributors respond to completion- or performance-contingent incentives, mostly using data from online review and knowledge-sharing platforms. Table 1 distinguishes our study from the literature along three key dimensions.

First, the literature has emphasized the direct effects of incentives on targeted tasks and individuals, and *intertask spillovers* where incentivizing one activity affects *recipients'* nonincentivized contributions (Kuang et al. 2019, Qiao et al. 2020, Wang et al. 2022, He et al. 2023). By contrast, we focus on *intergroup spillovers*, that is, an incentive's impacts on *excluded individuals'* task contributions. We complement prior findings to offer a more comprehensive picture of incentives' unintended impacts. Our finding of negative intergroup spillover effects starkly contrasts with the positive spillover effects predicted in the literature (Gallus 2017). For example, Mishra et al. (2022) suggest that users provide more reviews after observing incentivized reviews.

Second, the literature has mostly examined incentive impacts in online review or text-based knowledge-sharing contexts; there is limited evidence on *how incentive provision affects contributions in OSS* where knowledge creation is substantially more complex and interdependent (Bessen 2006). To our knowledge, only a few concurrent working papers have examined developer-solicited sponsorships, where GitHub developers exchange explicit benefits (e.g., consulting or project mentions) for monetary support, and find that such incentive mechanisms can reduce sponsored developers' contributions (e.g., Conti et al. 2023). By contrast,

Table 1. Summary of Studies on Incentivizing Digital Public Goods Contribution

Literature	Direct or spillover impact	Public goods context	Incentive mechanism
Gallus (2017)	Direct	Wikipedia	Completion-contingent
Burtch et al. (2018)	Direct	Online reviews	Completion-contingent
Khern-am-nuai et al. (2018)	Direct	Online reviews	Completion-contingent
Kuang et al. (2019)	Intertask spillovers	Online Q&A	Completion-contingent
Qiao et al. (2020)	Intertask spillovers	Online reviews	Completion-contingent
Wang et al. (2022)	Intertask spillovers	Online Q&A	Completion-contingent
Yu et al. (2022)	Direct	Online reviews	Performance-contingent
Qiao and Rui (2023)	Direct	Online reviews	Completion-contingent
He et al. (2023)	Direct/intertask spillovers	Online experience sharing	Completion-contingent
This study	Intergroup spillovers	OSS as a complex knowledge product	Performance-contingent

Note. For brevity, the review focuses on journal publications, even though several conference or working papers exist (e.g., Wang et al. 2012, Mishra et al. 2022, Conti et al. 2023).

we study incentive mechanisms that do not impose new commitments on OSS developers, and investigate their intergroup spillover effects.

Third, the literature has mostly focused on completion-contingent rewards (rewarding task completion); a few studies have examined the impact of performance-contingent rewards (rewarding only top performers) for highest-quality contributions in the online review context (Wang et al. 2012, Yu et al. 2022). We extend the literature to a context where only the first contributors of unknown bugs are rewarded. We also study how performance-contingent rewards affect nonincentivized contributor groups.

Our paper is also related to the empirical literature on reward-based crowdsourcing and bug bounty programs (BBPs), a specific reward-based crowdsourcing contest for software bugs (Fryer and Simperl 2017). This stream of research focuses on how crowdsourcing (including BBPs) affects crowd behavior, such as the quantity and quality of contributions and behavioral biases (e.g., Terwiesch and Xu 2008, Jeppesen and Lakhani 2010, Lee et al. 2018, Hwang et al. 2019, Zhang et al. 2019, Mo et al. 2021) or the crowd's responses to various contextual designs (e.g., Boudreau et al. 2011, Jian et al. 2019, Jin et al. 2021, Althuizen and Chen 2022, Gu et al. 2022, Koh and Cheung 2022, Lysyakov and Viswanathan 2023). It is relatively silent on the impact of crowdsourcing on nonincentivized individuals' work outputs. Our study is the first to analyze BBPs' spillover to nonsecurity outcomes.

This study draws upon various theories on crowdsourcing and multitasking (Holmstrom and Milgrom 1991, Terwiesch and Xu 2008). We discuss the relevant studies in developing the hypotheses.

2.2. Spillover to Core Members' Contributions to Incentivized Tasks

In general, core members' contributions to incentivized tasks are influenced by two factors, *task difficulty* and *motivation*. Regarding task difficulty, extant theories have produced different predictions by viewing the IBB as a reward-based crowdsourcing contest. One stream of research views crowdsourcing as an innovation contest, positing that an increase in crowd contributions intensifies the competition at the task level, thus reducing everyone's chance of obtaining a successful solution (typically defined as being the first to find a solution or finding the best solution (Terwiesch and Xu 2008, Boudreau et al. 2011). For example, in a BBP, increased competition for finding bugs increases the difficulty for everyone, including software maintainers, to be the first to find an unknown bug in the software (Zhou and Hui 2019). Conversely, a different research stream views crowdsourcing as a way for distant search, highlighting the possibility that crowd contributors can offer innovative solutions beyond the knowledge of experts and incumbents (Afuah and Tucci 2012, Poetz and Schreier 2012). Core members are recognized as experts (Lee and Cole 2003), so if the crowd mostly makes contributions that are overlooked by core members, crowdsourcing may not increase the difficulty for core members to complete the task.

Regarding motivation, we consider how incentivizing crowd contributions may affect core members' motivation through self-determination theory, which identifies individuals' three basic psychological needs, namely, competence, autonomy, and relatedness (Deci et al. 1999, Ryan and Deci 2000). First, as discussed above, an incentive may promote competition, which reduces the chances for core members to find a successful solution. The reduced performance in the crowdsourced task could undermine an individual's *perceived competence* (Deci et al. 1999). Second, the increased competition may lead core members to feel pressured into competing with the crowd, potentially eroding their *perceived autonomy* (Reeve and Deci 1996). Third, increased crowd contributions may enhance core members' *perceived relatedness* by signaling the community's greater interest and care for their project—a social benefit supported by prior work (Zhang and Zhu 2011, Baek and Shore 2020), but core members' perceived relatedness may also be reduced if they feel

that their contributions to the crowdsourced task are less needed.

Taken together, a crowd incentive may increase task competition, reducing core members' perceived competence and autonomy, which can negatively affect their crowdsourced task contributions. However, when viewing crowdsourcing as distant search and given the nuanced impacts of incentives on core members' perceived relatedness, a crowd incentive could increase their contributions, too. Based on this discussion, we propose the following competing hypotheses:

Hypothesis 1A. *Introducing a crowdsourcing contest to solicit crowd contributions for a task will decrease core members' contributions to that task.*

Hypothesis 1B. *Introducing a crowdsourcing contest to solicit crowd contributions for a task will increase core members' contributions to that task.*

2.3. Spillover to Core Members' Contributions to Nonincentivized Tasks

Multitasking theory posits that individuals' activities in one task can influence their performance in other tasks as different tasks draw on common resources, such as time, attention, and knowledge (Holmstrom and Milgrom 1991). Unlike sporadic crowd contributors, core members often contribute to multiple tasks across knowledge domains (Nakakoji et al. 2002, Lee and Cole 2003). Hence, a change in their activities in one task, as proposed in Hypothesis 1, can affect their performance in other (nonincentivized) tasks. However, competing theories offer contrasting predictions of this impact.

One perspective suggests that engaging in one task can *negatively* influence one's performance in other tasks, a phenomenon referred to as task substitutability (Balmaceda 2016). Given individuals' limited time and cognitive resources, resources devoted to one task directly reduce available resources for other tasks (Dumont et al. 2008). Prior cognitive research reinforces this view by highlighting that task switching increases cognitive load and hinders performance (Rogers and Monsell 1995, Monsell 2003). This view also aligns with the conventional outsourcing wisdom that outsourcing a task allows one to focus and work better on remaining tasks (Quinn and Hilmer 1994). In the OSS context, bug reporting demands significant time and cognitive effort for deep code comprehension, vulnerability identification, and detailed documentation (e.g., Foreman 2019, Harzevili et al. 2023). If crowd contributions reduce core members' workload in such tasks, core members can allocate more time and attention to other nonincentivized tasks, such as enhancement reporting, potentially increasing their contributions.

Conversely, the concept of task complementarity suggests that engaging in one task can *positively* influence one's productivity and performance in other tasks by fostering intertask learning and knowledge transfer (Lindbeck and Snower 2000, Schilling et al. 2003). Prior empirical studies show that exposure to different yet related tasks can boost productivity in creative tasks by fostering new understanding and cultivating transferable skills (Schilling et al. 2003, Aral et al. 2012, Mo et al. 2021). Studies on knowledge recombination likewise posit that combining insights across domains is central to knowledge creation, underscoring the benefits of engaging in different tasks (e.g., Fleming 2001). Although core members are considered experts, learning remains necessary because for complex knowledge production such as large OSS development, even core members are unlikely to master all knowledge (Akgul et al. 2023).

In our OSS context, although bug reporting demands time and cognitive effort, maintainers' direct involvement in these tasks may benefit their broader knowledge creation in ways that passively reading or fixing the crowd's bug reports may not afford. First, bug reporting fosters a broader information search. Unlike bug fixing, where OSS maintainers typically focus on issues within their area of expertise (known as maintainers' expertise), the exploratory nature of bug identification often pushes maintainers into broader domains (LaToza et al. 2006, Akgul et al. 2023).[2] Research on innovation suggests that narrow focus may stifle creativity, emphasizing the need for breaking boundaries and broader information search (Kaplan and Vakili 2015). Hence, the broader exploration promoted by bug reporting may help maintainers uncover more enhancement opportunities. Second, the firsthand involvement in bug reporting can provide rich tacit knowledge (Reber 1989). As a trial-and-error process, identifying bugs necessitates investigating diverse software functionalities and behavior, a process that often reveals enhancement opportunities and nuanced insights not documented in others' reports (Austin and Williams 2011). Such tacit knowledge and insights may help maintainers in other knowledge creation tasks, such as enhancement reporting. Our survey with OSS maintainers corroborates this view. For example, one maintainer stated that "bug report is a process to learn the details of the behavior, and thus it is also helpful to find potential enhancements"; another maintainer mentioned that "a well-written bug report often involves referencing parts of the code that one thinks is relevant to the bug. This ability to reference relevant parts of the code base can be very useful for most types of enhancement reports."

In sum, arguments exist for both task substitutability (because of time competition and task switching costs) and complementarity (because of learning), with

contrasting implications for the intergroup spillovers on core members' contributions to nonincentivized knowledge creation tasks. Although reducing incentivized task activities can save time and effort, such reduction may limit core members' intertask learning, potentially hampering their nonincentivized knowledge creation. We propose the following competing hypotheses:

Hypothesis 2A (Task Substitutability). *In a crowdsourcing contest to solicit crowd contributions, core members will increase (decrease) their contributions to nonincentivized knowledge creation tasks if their contributions to incentivized task decrease (increase).*

Hypothesis 2B (Task Complementarity). *In a crowdsourcing contest to solicit crowd contributions, core members will decrease (increase) their contributions to nonincentivized knowledge creation tasks if their contributions to incentivized task decrease (increase).*

3. Research Context

Our empirical context is the OSS community for Python, which has been consistently regarded as one of the most popular general-purpose programming languages since its launch in 1991. Python's development follows a canonical community-driven model of OSS development (Lee and Cole 2003). The Python setting provides the following desirable features for our research: (1) Python OSS is an exemplary community-based complex knowledge product supported by peripheral contributors (the crowd) and maintainers (core members). (2) The two important tasks in Python, bug reporting and enhancement reporting, are highly comparable, but only bug reporting by the crowd (cf. core members) is incentivized by the IBB, providing a clean setting for us to test Hypothesis 1 and Hypothesis 2. (3) The IBB shock is exogenous, facilitating causal identification of its impacts on the contributions made by core members.

3.1. Quality Assurance and Innovation in Python

Similar to other OSS, Python's development relies on developers to perform two important functions (Lee and Cole 2003): (1) *quality assurance*, which covers identifying and fixing bugs in the software, and (2) *innovation*, which covers proposing and implementing new features to enhance the usefulness of Python. What sets OSS apart from firm-based, closed-source software is the absence of a firm dictating the quality assurance and innovation processes. Specifically, like other OSS, the evolution of Python relies on community members to share ideas about quality assurance and innovation to guide its development direction. Interested contributors can translate the ideas into code implementations of bug fixes or new features. Figure 1 illustrates the Python development process.

Anyone who discovers a bug can submit a bug report to the Python issue tracker,[3] the official website for bug reports. Figure 2 shows a sample bug report, which typically contains a title, the problem description, the contributor's identity, creation date, issue type, assignee (i.e., the person responsible for resolving the bug), and a "nosy list" (a list of people interested in the bug). Reporters can select a Python maintainer as the assignee based on the "expert index" (https://devguide.python.org/core-team/experts) or just let the Python maintainers assign themselves to the report. The Python issue tracker also facilitates follow-up discussions and records the process until the bug is fixed. All registered users interested in the bug can add themselves to the nosy list, which allows them to receive updates on the report.

Similarly, anyone can propose new features by submitting enhancement reports through the same website. Unlike bug reports that concern potential defects, enhancement reports suggest new features or improvements to Python, such as new modules, functions, or arguments for existing functions. Figure 3 shows a sample enhancement report, where the reporter suggested adding a new function, asyncio.get_running_loop(), to the Python library. Enhancement reports and bug reports typically use different languages and terms. For example, as contrasted in Online Appendix A.1, bug reports often use keywords such as "error" and "fail," whereas enhancement reports often use words such as "implement," "add," and "support."

Figure 1. (Color online) Python Development Process

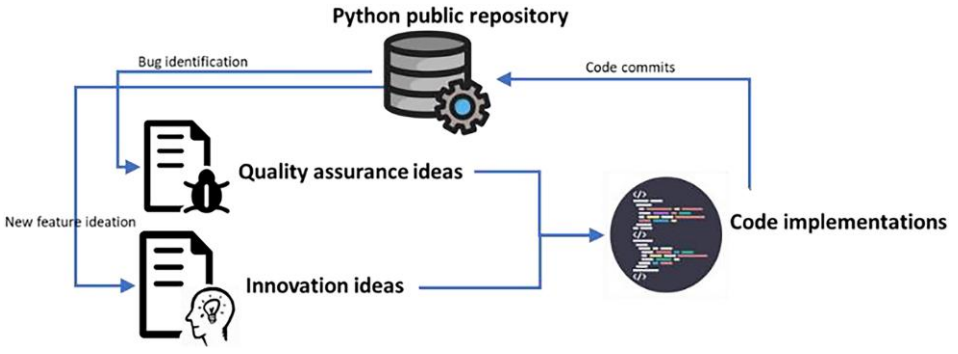
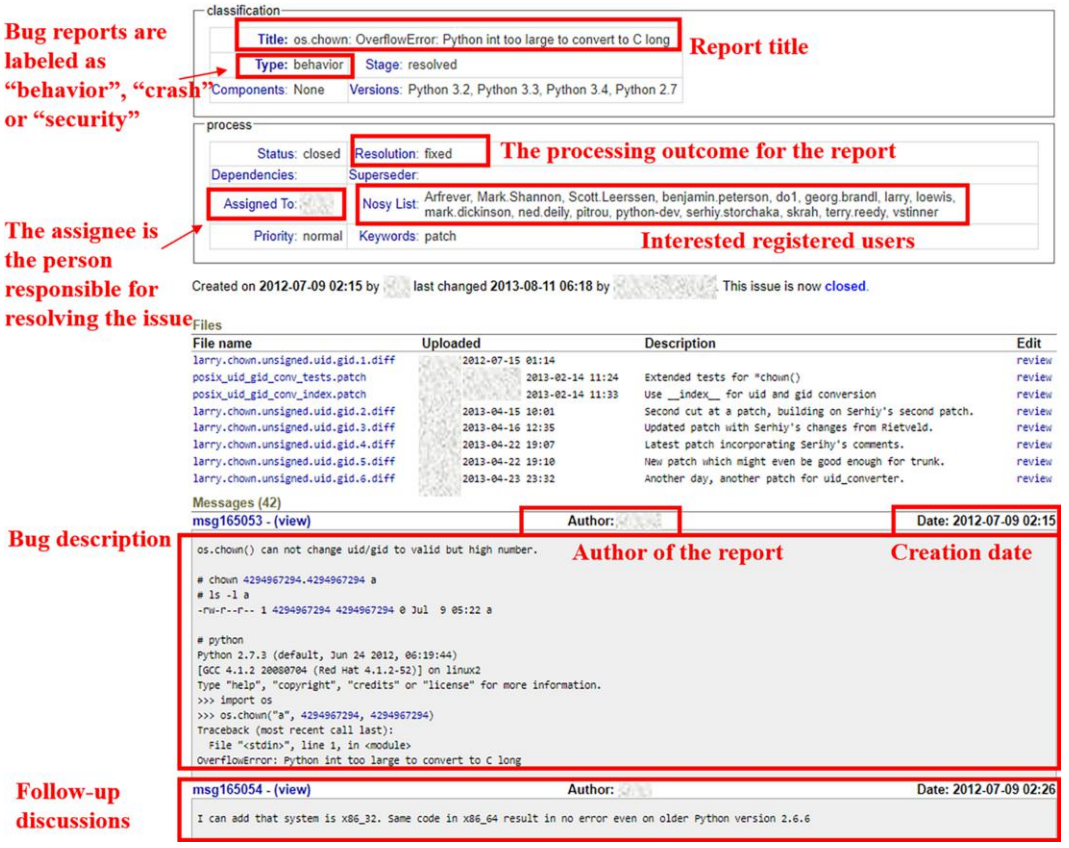


Figure 2. (Color online) Sample Python Bug Report

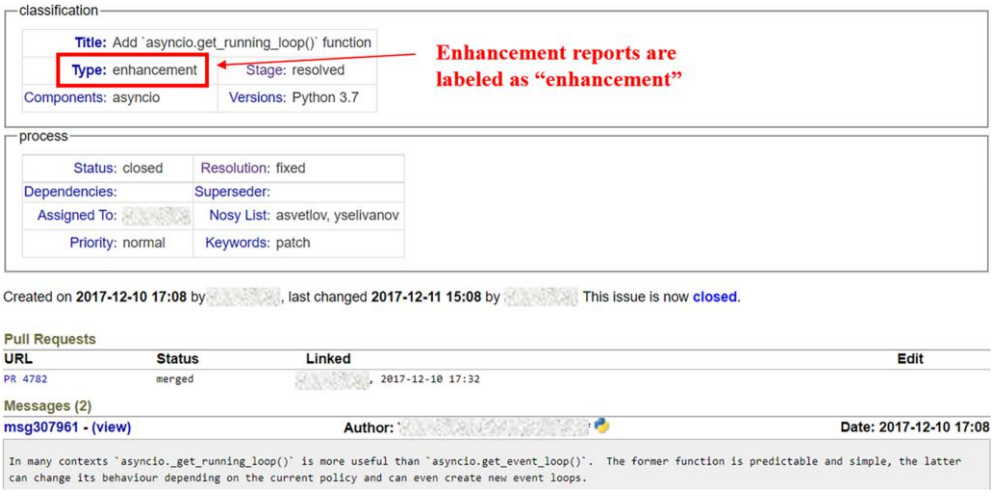


Note. This bug report can be accessed at https://bugs.python.org/issue15301 (accessed December 1, 2021).

We discuss the concern related to report misclassification in a robustness check.

After bugs or new features are reported and verified, Python community members can fix the bugs or implement the new features by changing the source code, which will then be reviewed and integrated into the common code repository through code commits. Each code commit represents a confirmed modification of the software. Only Python maintainers have commit access to the Python public repository.

Figure 3. (Color online) Sample Python Enhancement Report



Source. https://bugs.python.org/issue32269 (accessed December 1, 2021).

3.2. Maintainers and Peripheral Contributors

Although anyone can participate in Python development, roles and hierarchies emerge in the community (O'Mahony and Ferraro 2007, Dahlander and O'Mahony 2011). In this study, Python maintainers, who constitute the official maintenance team, are considered core members. These maintainers chart the development of Python by having exclusive commit privileges to the source code and the authority to approve or reject contributions from peripheral contributors (Nakakoji et al. 2002). Similar to other OSS, an individual needs to consistently make high-quality contributions to be selected as a maintainer by existing maintainers. Apart from maintainers, a far larger group of peripheral contributors from the general public contribute to the development of Python. These peripheral contributors do not have formal roles or commit privileges and mostly make temporary and irregular contributions (Nakakoji et al. 2002, Ye and Kishida 2003). Figure 4 shows that among the 9,668 peripheral contributors who contributed at least one bug or enhancement report before January 2020, 7,514 (78%) made only one contribution. By contrast, each Python maintainer contributed an average of 57 bug or enhancement reports during the same period. Although OSS contributors are typically motivated by enjoyment, their desire to improve the product, or the temporary need for functionalities (Shah 2006, Setia et al. 2012, Coelho et al. 2018), the IBB presents an exogenous shock that incentivizes peripheral contributors to contribute bug reports by providing an additional monetary incentive. Online Appendix A.2 further contrasts the characteristics of these two types of contributors.

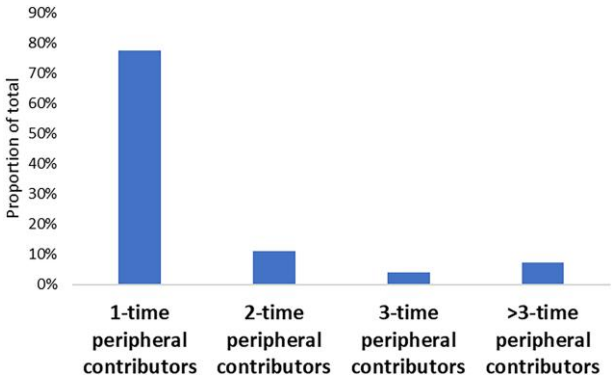
3.3. The Internet Bug Bounty Program

Microsoft, Facebook, and HackerOne initiated the IBB program on November 7, 2013, to engage the public in identifying unknown security vulnerabilities to protect core internet infrastructure and OSS. Under the IBB, peripheral contributors may claim a "bounty reward" ranging from US$500 to US$5,000 by reporting previously unknown security vulnerabilities through bug reports. The official implementation of the Python programming language, CPython (referred to as "Python" throughout this paper), was included in the first round of the IBB in November 2013 alongside several other popular OSS projects. As Python maintainers serve as bug report judges, they do not claim bug bounty rewards despite contributing bug reports themselves (Heimes 2013a). We focus on the impact of the IBB on Python because of data availability (some OSS do not disclose bug reports; see Online Appendix A.3 for a discussion and the complete list of OSS in the IBB). The scope of the IBB (i.e., the OSS to be included) is decided by an independent panel of security experts from large IT organizations and academia based on the following criteria (Internetbugbounty.org 2013): (1) "Widespread Use," meaning the software affects a significant number of users; (2) "Mind the Gap," meaning the IBB will not duplicate the work of other initiatives; and (3) "Active Security Team," which requires the project to be maintained by an active development or security team capable of handling bug reports. As corroborated by Python maintainers, the implementation of the IBB is nonintrusive; that is, this program does not interfere with the usual bug report workflows while allowing reporters to claim a bug bounty reward from HackerOne after report confirmation (Heimes 2013b). Online Appendix A.4 details a specific IBB process.

The IBB was designed to mitigate some potential negative effects of rewards on recipients. First, it is a performance-contingent reward, only to the first reporters of unknown vulnerabilities. Unlike completion-contingent rewards that may crowd out intrinsic motivation (Deci 1975, Deci et al. 1999), performance-contingent rewards strengthen intrinsic motivation by validating recipients' capabilities and enhancing their perceived competence (Deci et al. 1999, Yu et al. 2022). Second, crowd contributors can choose not to claim the IBB reward, so the conventional concern that rewards may undermine the crowd's perceived autonomy does not apply (Deci et al. 1999). Nevertheless, how such incentives create intergroup spillovers affecting nonincentivized core members remains unclear.

The IBB provides a quasi-natural experimental setting to study how incentivizing crowd contributions affects core members' contributions. First, it is an exogenous event because the decision to include Python in the IBB was made by a panel independent of Python maintainers, who were not informed of this decision in advance.[4] Such exogeneity allows us to identify the causal impact of the IBB on Python maintainers' contributions. Second, none of the three scoping criteria for IBB projects is related to the bug reporting activities of

Figure 4. (Color online) Frequency of Python Contributions by Peripheral Contributors



Note. Each bug or enhancement report is counted as one contribution.

maintainers. Therefore, the selection of Python in the IBB is not motivated by endogenous patterns or distributions in report contributions. Third, the IBB was implemented nonintrusively, minimizing the chance of having contemporaneous events, such as changes in the bug or enhancement reporting processes, that could confound its effects.

4. Data and Measures

To evaluate Hypothesis 1 and Hypothesis 2, we analyze the quantity and quality of maintainers' bug reports and enhancement reports, respectively. As discussed in Section 3, the process of enhancement reporting is similar to that of bug reporting, but enhancement reporting does not attract the IBB incentive. This differential treatment provides a clean setting to test the IBB's impact on core members' knowledge creation in tasks where the IBB incentive does not apply (i.e., feature enhancement). We also examine the IBB's impact on maintainers' coding activities, acknowledging that coding activities can be affected directly by bug and enhancement reporting because of upstream and downstream task dependency (e.g., fewer enhancement requests may reduce coding needs; see Figure 1) and indirectly by task complementarity or substitutability. The net impact of the IBB on Python maintainers' coding activities is ambiguous under these dependencies and potential confounds. Therefore, we use enhancement reports to test Hypothesis 2 and the maintainers' coding activities for robustness checks.

We measure quantity as the monthly number of bug and enhancement reports, and we operationalize quality using the average length of these reports based on the evidence that developers perceive longer reports as having higher quality because of their detail and actionability (Zimmermann et al. 2010), and that more substantial issues require more comprehensive documentation. For robustness, we consider multiple alternative quality metrics: (1) the ratio of valid reports—reports may be deemed invalid (outdated, duplicates, inappropriate, or nonissues) after peer review; the ratio of valid reports reflects whether maintainers' reports are of high quality; (2) the ratio of reports including patches, which indicates whether the reporters included solutions; and (3) the size of patches, which indicates whether the included patches involve more substantial changes. In Section 7.2, we also consider patch merge success rate as an alternative report quality measure and obtain consistent findings. Note that these quality measures are undefined if a maintainer did not submit any report in a given month (Burtch et al. 2018, Wang et al. 2022).

We collected all Python bug and enhancement reports submitted before January 2020 from the Python issue tracker. Online Appendix B.1 presents the collection process in detail. Note that report misclassification is not a material concern in our context because maintainers have the expertise and incentive to label reports correctly.[5] Moreover, Python has a triage team that reviews and corrects report information. In one robustness check, we combine the report type labels and keywords to classify the reports and obtain similar findings. Similarly, we collected the Java maintainers' bug and enhancement reports from the OpenJDK issue tracker.

We also collected the following data (see Online Appendix B.1 for the coding procedures): (1) Maintainers' coding activities as measured by the number and size of their code commits (Chen et al. 2022). Each code commit represents a confirmed change to the source code. Commit size is measured by the lines of code (LOC) affected, a standard measure for development effort for code changes (Boehm 2002). (2) Reports contributed by maintainers of well-known Python third-party libraries in different applications (e.g., Numpy, Matplotlib, and Scikit-Learn). (3) Control variables, including (a) the maintainers' reporting experience (*Reporting Experience*) as measured by the number of months since their first report; (b) version releases for Python or Java in each month (*Version Releases*); and (c) the popularity of Python and Java over time (*Language Popularity*) as measured by the TIOBE index, a well-known programming language popularity index based on search statistics from major search engines.

By aggregating the number of bug and enhancement reports contributed by each maintainer in each month, we construct a panel data set with "maintainer" as the cross-sectional unit and "month" as the time unit. We exclude maintainers who did not experience the implementation of the IBB either because they joined after the program or quit before its implementation (OSS projects typically label a maintainer as inactive if the person has not made any contribution in a year; we treat a maintainer as quitting the project if the person did not make any contribution for more than a year). The panel is unbalanced because maintainers may join and quit at different times. We use a balanced panel in a robustness check. Our data set includes 92 Python maintainers (58% of all Python maintainers in history) who were responsible for 89% of all maintainer-generated bug and enhancement reports in Python, and 272 Java maintainers (52% of all Java maintainers in history) who were responsible for 82% of all maintainer-generated bug and enhancement reports in Java. Evidently, these maintainers are active, committed, and representative of the community.

Table 2 presents the summary statistics, which show a 23%–29% drop ($p < 0.01$) in Python maintainers' bug and enhancement report contributions after the IBB implementation. By contrast, the Java maintainers' contributions of bug and enhancement reports remained

Table 2. Summary Statistics

	Before IBB				After IBB				Diff
	Mean	S.D.	Min	Max	Mean	S.D.	Min	Max	*t* stat.
Python maintainers ($n = 92$)									
Monthly bug reports	0.504	1.480	0	16	0.360	1.333	0	32	−3.92***
Monthly enhancement reports	0.415	1.405	0	17	0.320	1.283	0	25	−2.71***
Bug report length	769	1,095	55	20,049	906	1,252	60	25,408	1.91*
Enhancement report length	584	777	36	12,267	594	451	57	3,958	0.28
Ratio of valid bug reports	0.824	0.324	0	1	0.849	0.314	0	1	1.30
Ratio of valid enhancement reports	0.783	0.350	0	1	0.843	0.318	0	1	2.83***
Ratio of bug reports containing patches	0.367	0.430	0	1	0.406	0.437	0	1	1.49
Ratio of enhancement reports containing patches	0.481	0.453	0	1	0.483	0.456	0	1	0.07
Patch lengths in bug reports (LOC)	260	504	10	3,174	485	1,621	4	23,577	1.93*
Patch lengths in enhancement reports (LOC)	1,752	4,270	5	34,414	1,929	6,387	13	59,164	0.36
Monthly commits	3.754	10.161	0	129	2.619	9.089	0	154	−4.55***
Monthly commits related to adding functions/parameters	0.762	2.422	0	42	0.605	2.308	0	38	−2.52**
Commit size (affected LOC)	216	945	1	15,173	239	1,468	0	30,758	0.36
Commit data density	9.011	67.137	0	1,268	6.939	51.235	0	992	−0.76
Commit decision density	0.332	1.288	0	24	0.301	1.060	0	23	−0.56
Controls									
Reporting experience	47	19	0	75	88	30	0	149	57.40***
Version release	0.045	0.207	0	1	0.066	0.248	0	1	3.28***
Language popularity	3.857	0.471	3.11	5	4.469	2.118	1.990	10	12.66***
Java maintainers ($n = 272$)									
Monthly bug reports	1.515	3.688	0	46	1.454	3.376	0	85	−1.10
Monthly enhancement reports	0.428	1.347	0	23	0.510	1.492	0	32	3.44***
Bug report length	920	1,399	29	21,779	938	2,068	7	90,210	0.38
Enhancement report length	519	839	16	17,766	562	792	11	14,609	1.44
Ratio of valid bug reports	0.726	0.358	0	1	0.811	0.310	0	1	10.25***
Ratio of valid enhancement reports	0.751	0.380	0	1	0.843	0.323	0	1	7.46***
Ratio of bug reports containing patches	0.230	0.362	0	1	0.341	0.406	0	1	11.02***
Ratio of enhancement reports containing patches	0.235	0.384	0	1	0.377	0.432	0	1	9.19***
Patch lengths in bug reports (LOC)	543	3,625	8	86,102	564	3,509	1	90,032	0.14
Patch lengths in enhancement reports (LOC)	4,611	61,094	9	1,061,661	2,556	24,162	5	817,347	−0.99
Monthly commits	1.182	3.195	0	44	1.502	4.901	0	433	4.35***
Monthly commits related to adding functions/parameters	0.367	1.189	0	19	0.404	1.04	0	14	2.12**
Commit size (affected LOC)	1,055	11,726	1	304,862	2,057	57,770	1	3,922,706	0.68
Commit data density	3.511	1.732	0	19	3.721	1.787	0	25	4.13***
Commit decision density	0.214	0.285	0	4	0.207	0.278	0	5	−0.82
Controls									
Reporting experience	43	28	0	78	80	39	0	152	62.46***
Version release	0	0	0	0	0.071	0.257	0	1	19.71***
Language popularity	16.844	0.710	15.910	18	16.765	2.213	12.430	21	−2.48***

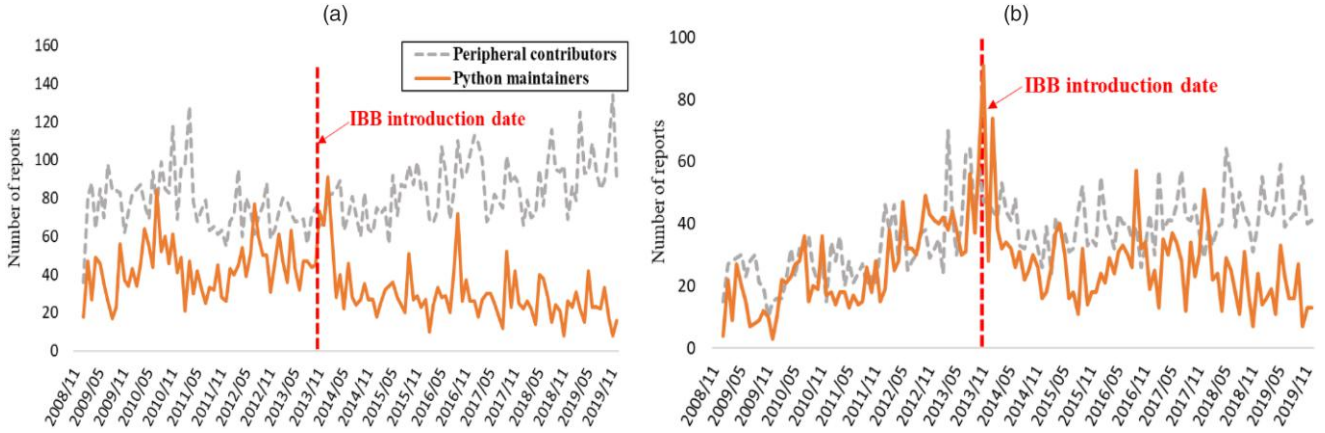
Note. stat., statistic.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

stable or even increased during the same period. We observe no significant changes (at the 0.05 significance level) in the Python maintainers' report quality as measured by report length, valid ratio, or patch ratio, but the ratio of valid enhancement reports has increased. We examine these changes more closely through model-free evidence and regression analyses.

5. Model-Free Evidence

We first consider the IBB's impact on report quantity. Figure 5 plots the monthly number of bug reports and enhancement reports contributed by Python maintainers and peripheral contributors, with the vertical lines indicating the IBB's introduction date. Panel (a) shows that the number of bug reports by both types of contributors followed a similar trend before the IBB, but immediately after the program was introduced, the peripheral contributors started contributing more reports, whereas the maintainers contributed fewer. Panel (b) shows a similar decline in enhancement reports from Python maintainers compared with peripheral contributors after the IBB implementation. Notably, these trend changes occurred around the time the IBB was implemented. We do not find any significant events alongside the IBB by

Figure 5. (Color online) Monthly Reports by Python Contributors from January 2009 to December 2019



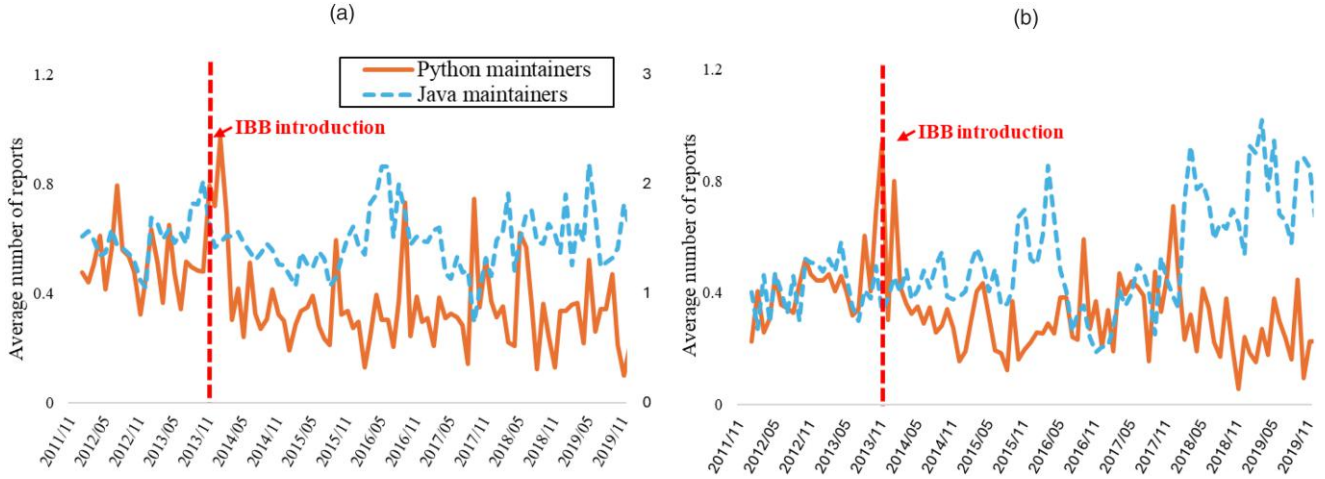
Notes. (a) Bug reports. (b) Enhancement reports. The number of reports before January 2009 is small and is therefore excluded. Panel (a) shows the trend in all bug reports, including those that do not receive rewards under the IBB.

manually reading the reports and email exchanges among Python maintainers during this period.

As the number of reports contributed by Python peripheral contributors is influenced by the IBB, they may not serve as a good control group for investigating the changes in the number of reports contributed by maintainers. In any case, most peripheral contributors make sporadic contributions (see Figure 4). We plot the contributions of Python maintainers relative to those of Java maintainers in Figure 6. Evidently, the contributions of Python and Java maintainers followed a similar trend before the IBB implementation. However, immediately after the IBB, the monthly average number of bug reports and enhancement reports contributed by Python maintainers decreased, whereas those by Java maintainers remained similar to their pre-IBB levels.

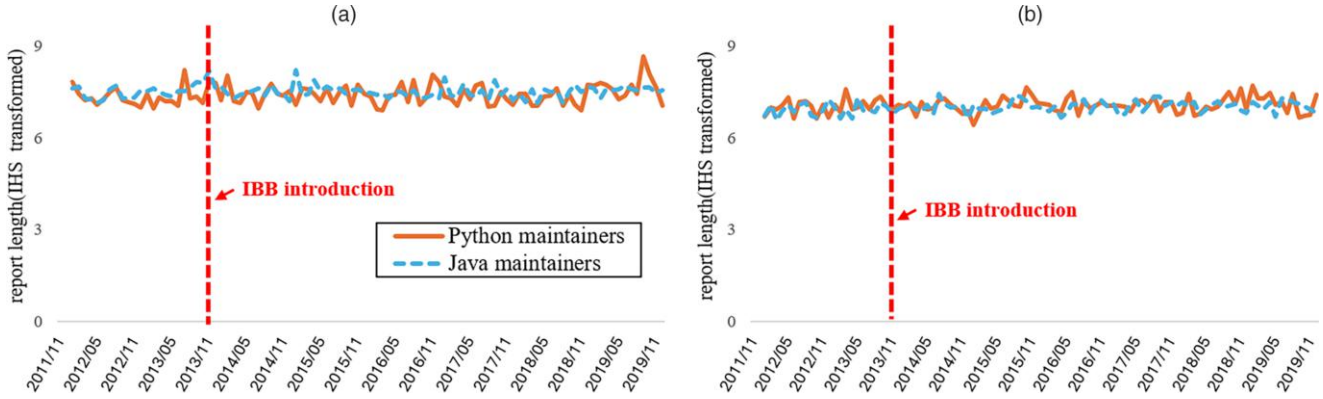
We then assess the impact of the IBB on report quality. Figure 7 examines whether Python maintainers made their reports longer or more substantial compared with Java maintainers. Unlike the significant drop in report quantity, we do not find a notable change in report length following the IBB. Figures in Online Appendix C.1 show no clear changes in the validity ratio, patch ratio, and patch length of Python maintainers' reports compared with Java maintainers. Overall, Figures 5–7 indicate that the IBB leads to fewer bug and enhancement report contributions from Python maintainers, but its effect on report quality is not apparent.

Figure 6. (Color online) Monthly Average Reports by Python and Java Maintainers



Notes. (a) Bug reports. (b) Enhancement reports. To facilitate comparison of the trends in panel (a), the monthly contributions by Python maintainers are scaled along the left *y*-axis, whereas those of Java maintainers are scaled along the right *y*-axis. Contribution trends are based on reporting activities of Python and Java maintainers from January 2012 to December 2019. The monthly average number of reports is calculated by dividing the total number of reports in a given month by the number of active maintainers.

Figure 7. (Color online) Average Report Length by Python and Java Maintainers



Notes. (a) Bug reports. (b) Enhancement reports. We use inverse hyperbolic sine (IHS) transformation to reduce skewness.

6. Empirical Analysis

6.1. Identification Strategy

We employ the DID strategy to estimate the impact of the IBB on Python maintainers' contributions by comparing the changes in their monthly contributions relative to Java maintainers for several reasons.[6] First, the creator of Python has listed Java as its top comparator (Van Rossum 1997). Second, Python and Java are frequently listed as two of the most popular programming languages in the world (Cass 2022), and both have large contributor communities. Each of them has a large group of maintainers spanning diverse backgrounds, which is lacking in most other programming languages (e.g., R has fewer than 20 maintainers). Third, with reference to Online Appendix C.2, many popular websites that use Python at the backend also use Java as a programming language, suggesting that Python and Java have a comparable commercial use.

Our identification strategy is premised on the assumption that the contribution trends of Python and Java maintainers are similar in the absence of the IBB. The model-free evidence in Figure 6 shows that these maintainers' contributions followed a similar trend before the IBB and diverged only after the IBB implementation. Section 7.2 presents statistical tests of the parallel trends and various placebo tests.

We undertake three additional steps to strengthen our identification strategy. First, following the literature (e.g., Wang et al. 2022), we construct a matched sample of Python and Java maintainers with similar observed characteristics using coarsened exact matching (CEM). CEM has several advantages over other matching methods such as propensity score matching because it does not assume the underlying data-generation process and it reduces imbalance, model dependence, and estimation errors (Iacus et al. 2012). Combining DID with matching allows for a more precise comparison of the contribution behavior between Java and Python maintainers. We perform one-to-one CEM based on the maintainers' pretreatment average reporting activities (average monthly number of bug reports and enhancement reports), average bug and enhancement report lengths, average monthly number and size of commits, and report experience.[7] The matching yields 75 matched pairs of Python and Java maintainers (11,341 maintainer-month observations) with no statistically significant differences in their pretreatment reporting and code commit behaviors or experiences (see Table 3). We also examine our results using alternative methods (see Online Appendix D): (1) an unmatched full sample, (2) extra matching covariates (resulting in a smaller sample), and (3) different matching strategies such as entropy balancing, to ensure that our results are not driven by sample selection, observable covariate imbalances, and specific matching strategies.

Second, instead of Java maintainers, we repeat the same analysis using maintainers of Python third-party libraries as an alternative control group in Online Appendix D, noting that the IBB only rewards bug reports in the Python core library but not in third-party libraries. Therefore, the maintainers of these third-party libraries can serve as an alternative control group. Our results remain qualitatively similar when using this alternative control group.

Third, we apply the synthetic DID method, which constructs a control group to mirror the trends of the treatment group and hence satisfies the parallel trends assumption by construction. This approach alleviates omitted variable concerns that might affect identification (Arkhangelsky et al. 2021). As reported in Online Appendix D, the synthetic DID produces similar conclusions. As these strategies lead to qualitatively similar findings, we focus on the results from the DID estimations with CEM.

Table 3. Balance Checks of Variables Before and After CEM

		Mean (Python)	Mean (Java)	Std. Diff.	*t*-test after matching (*p*-value)
Number of bug reports per month (IHS)	Before	0.30	0.67	−0.675	0.678
	After	0.28	0.30	−0.068	
Number of enhancement reports per month (IHS)	Before	0.24	0.25	−0.022	0.835
	After	0.21	0.20	0.034	
Average bug report length (IHS)	Before	5.21	5.46	−0.092	0.719
	After	4.97	5.15	−0.059	
Average enhancement report length (IHS)	Before	4.76	4.36	0.135	0.725
	After	4.41	4.23	0.058	
Number of commits per month (IHS)	Before	0.77	0.54	0.271	0.593
	After	0.49	0.43	0.087	
Average commit size (IHS)	Before	3.26	4.05	−0.270	0.979
	After	2.93	2.94	−0.004	
Maintainer experience (Log)	Before	3.85	3.37	0.466	0.945
	After	3.82	3.81	0.011	

Notes. We use inverse hyperbolic sine (IHS) transformation or logarithmic transformation to mitigate the influence of extreme values. We use inverse hyperbolic sine transformation for dependent variables to be consistent with the regressions. Results are robust to different transformations. The standardized difference is based on Austin (2009).

6.2. Main Results

The DID regression takes the following form:

$$Reports_{it} = \alpha_0 + \beta Python_i \times IBB_t + \alpha X_{it} + \tau_t + f_i + \epsilon_{it}, \quad (1)$$

where $Reports_{it}$ represents the monthly quantity or quality of bug or enhancement reports contributed by maintainers. We use inverse hyperbolic sine transformation to scale the dependent variables so that they become more normally distributed (Hui 2020, Bahar et al. 2023). Inverse hyperbolic sine transformation may be preferred over log-transformation when a variable takes on zero values (Bellemare and Wichman 2020). Using raw-scale (untransformed) dependent variables or alternative transformations, such as Poisson or negative binomial regressions, does not qualitatively change our results (Online Appendix D), suggesting that these results are robust to the potential bias because of the transformation of the dependent variable (Cohn et al. 2022, Chen and Roth 2024).[8] We do not use nonlinear regression models for our main analysis because they require strong distributional assumptions to identify the DID estimator (Hoetker 2007). $Python_i$ is a dummy variable that equals one for Python maintainers and zero for Java maintainers, and IBB_t is a dummy variable that equals one for every month on or after November 2013 and zero otherwise. The vector f_i represents the maintainer fixed effects, which capture time-invariant maintainer features, such as personal characteristics that could affect the contributions. τ_t represents time-varying fixed effects, including year and month-of-year fixed effects, which account for the seasonality and trends of online contributions (e.g., different days in different months (Xu et al. 2020)). X_{it} contains the time-varying control variables *Reporting Experience*, *Version Releases*, and *Language Popularity*. We include all reports from January 2012 to December 2019 in the analysis. We cluster all standard errors at the individual maintainer level.

Column (1) in Table 4 presents the estimation with the monthly number of bug report contributions as the dependent variable. The focal variable, Python × IBB, has a negative and statistically significant coefficient of −0.267. Column (2) shows a similar result with the control variables included. These estimates indicate an average 48% decrease in the number of bug report contributions by Python maintainers relative to Java maintainers.[9] Referring to Figures 5(a) and 6(a), these estimates reflect that the IBB has had a substantial impact on Python maintainers' bug report contributions. Column (3) assesses the IBB's impact on report length and shows no significant increase in content length.[10] We consider several alternative quality measures and obtain similar findings. Column (4) shows no change in the valid ratio of Python maintainers' bug reports, column (5) shows no change in the likelihood for their bug reports to contain patches, and column (6) suggests that these patches, if offered, do not become more substantial. The absence of significant influence on the maintainers' bug report quality is unsurprising given the high standards and quality of contributions that OSS maintainers consistently uphold (Gousios et al. 2015).

To assess the IBB's impact on Python maintainers' enhancement reporting, we replicate the analysis using maintainers' enhancement reports as the dependent variable. Columns (1)–(6) of Table 5 present the results. The coefficients for Python × IBB in columns (1) and (2) are negative and statistically significant, indicating a 38% reduction in the number of enhancement reports

Table 4. IBB on Bug Reporting

Variables	(1) No. of bug reports	(2) No. of bug reports	(3) Length of bug reports	(4) Ratio of valid bug reports	(5) Ratio of bug reports containing patches	(6) Patch length
Python × IBB	−0.267***	−0.274***	0.108	−0.052	−0.056	0.244
	(0.051)	(0.051)	(0.113)	(0.032)	(0.054)	(0.201)
Report experience		0.023	0.011	−0.046	−0.040	−0.502***
		(0.043)	(0.088)	(0.030)	(0.045)	(0.150)
Version releases		0.006	0.146*	0.022	−0.022	0.124
		(0.023)	(0.076)	(0.032)	(0.036)	(0.159)
Language popularity		−0.140**	−0.001	−0.023	−0.012	0.092
		(0.059)	(0.100)	(0.032)	(0.044)	(0.239)
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes	Yes
Observations	11,341	11,341	2,794	2,794	2,794	1,302
No. of maintainers	150	150	150	150	150	150

Notes. Robust standard errors clustered by maintainers are reported in parentheses. FE, fixed effects.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

from Python maintainers (relative to Java maintainers) after the IBB. Columns (3)–(6) indicate no significant changes in the length, valid ratio, patch ratio, or the size of patches contained in Python maintainers' enhancement reports.

Online Appendix D presents validation for the parallel trends assumption and shows that the results above are robust to alternative control groups, matching strategies, data transformation, and data samples. Overall, consistent with the model-free evidence, our regression results show that the IBB significantly reduces the number of bug and enhancement reports contributed by Python maintainers. The impact of the IBB on report quality is not statistically significant.

6.3. Code Commits

We examine the impact of the IBB on Python maintainers' coding activities as measured by the number and size of their code commits. The main observation, shown in columns (1) and (4) of Table 6, is that the IBB has overall led Python maintainers to contribute fewer code commits without significant changes in their commit size. Because coding is a downstream task from contributor reports (Figure 1), the changes in maintainers' coding activities could be driven by both the number of newly identified bugs or enhancement ideas (e.g., fewer enhancement ideas would require less coding) and the extent of task complementarity or substitutability (Hypothesis 2). To better account for the influence of report inflow, we reestimate the IBB's impact on code commits after controlling for monthly bug and enhancement report numbers, including both the maintainers' own reports and overall Python reports. The attenuated Python ×IBB coefficient in column (2) confirms that part of the IBB's impact on code commits could be mediated by the reduced Python bug and enhancement reports, which negatively affects the downstream code commit task. This net negative

Table 5. IBB on Enhancement Reporting

Variables	(1) No. of enhancement reports	(2) No. of enhancement reports	(3) Length of enhancement reports	(4) Ratio of valid enhancement reports	(5) Ratio of enhancement reports containing patches	(6) Patch length
Python × IBB	−0.137***	−0.148***	−0.119	0.005	0.004	−0.253
	(0.037)	(0.038)	(0.113)	(0.042)	(0.067)	(0.317)
Report experience		0.081*	0.130	−0.009	0.001	0.298
		(0.041)	(0.089)	(0.042)	(0.061)	(0.236)
Version releases		0.008	−0.016	0.011	−0.011	−0.229
		(0.018)	(0.114)	(0.038)	(0.038)	(0.275)
Language popularity		−0.136***	0.014	0.065	0.055	−0.092
		(0.044)	(0.116)	(0.042)	(0.060)	(0.232)
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes	Yes
Observations	11,341	11,341	1,820	1,820	1,820	944
No. of maintainers	150	150	150	150	150	150

Note. Robust standard errors clustered by maintainers are in parentheses.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

Table 6. IBB on Code Commits

	Quantity			Quality			Halstead effort
Variables	(1) Total code commits	(2) Total code commits (additional control of report numbers)	(3) Commits related to new mod./func./param.	(4) Affected LOC per commit	(5) Decision density per commit	(6) Data density per commit	(7) Halstead effort
Python × IBB	−0.213***	−0.143*	−0.082**	0.013	0.020	−0.083	−1.012***
	(0.081)	(0.073)	(0.039)	(0.210)	(0.039)	(0.112)	(0.334)
Bug reports		0.107***					
		(0.015)					
Enhancement reports		0.153***					
		(0.019)					
Overall bug reports		0.0003**					
		(0.0001)					
Overall enhancement reports		−0.0004*					
		(0.0003)					
Controls	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Observations	11,341	11,341	11,341	2,986	2,986	2,986	11,341
No. of maintainers	150	150	150	150	150	150	150

Notes. Robust standard errors clustered by maintainers are reported in parentheses. mod., module; func., function; param., parameter.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

Python×IBB coefficient suggests that the IBB may have negatively influenced code commits beyond the reduced reporting route, possibly because of task complementarity (e.g., reduced learning of code restructuring opportunities or alternative solutions beyond addressing reported issues (Swanson 1976)). In Online Appendix C.3, our analysis of commit-to-report ratios yields consistent findings.

We conduct additional tests for further insights. First, we examine how the IBB affects code commits related to new modules, functions, or parameters. We identify such commits by checking whether the code changes involve adding new modules, functions, or parameter names (excluding renames). Consistent with the reduction in enhancement reporting, column (3) of Table 6 shows that the IBB has led Python maintainers to contribute fewer of such commits. Second, we consider alternative commit quality measures from the literature (Banker et al. 1998): (a) decision density, which is measured by McCabe's cyclomatic complexity and captures the complexity of program logic, and (b) data density, which is measured by Halstead's N2 software science metrics and captures the amount of data required to perceive and process. Columns (5) and (6) show no significant IBB effects on these commit quality measures. Finally, following the software engineering literature, we examine the estimated overall coding effort using the Halstead effort metric, which accounts for both the quantity and complexity of code commits (Halstead 1977). Consistent with the results in columns (1)–(6), column (7) suggests a reduction in maintainers' overall coding effort after the IBB (Online Appendix C.3 shows a similar result with report numbers as control variables).

6.4. Mechanism Exploration

We present two sets of results exploring explanations for Python maintainers' contribution changes. First, Hypothesis 1A posits that IBB may affect both bug identification difficulty and maintainers' motivation. Given that we cannot measure motivation directly and that the motivation effects may mingle with task difficulty (e.g., increased task difficulty may reduce perceived competency; see Hypothesis 1), we test whether finding bugs in Python has become more difficult after the IBB or if the decreased bug reporting is solely due to decreased motivation (Section 6.4.1). We also show that the loss of interest—a concept reflecting intrinsic motivation (Deci 1992)—may not explain our findings in Section 7.1. Second, to support the explanation underlying Hypothesis 2B, we demonstrate that bug and enhancement reporting are complementary activities (Section 6.4.2), and our survey and post hoc analysis suggest that such complementarity may be due to intertask learning (Section 6.4.3). We assess alternative explanations, including (a) Python maintainers losing interest, (b) Python maintainers being burdened with resolving reports, and (c) increased enhancement reports by peripheral contributors substituting Python maintainers' enhancement reports, in Section 7.1.

6.4.1. Increased Difficulty of Finding Bugs. We examine whether the introduction of the IBB increased the difficulty of finding bugs in Python. One way for Python maintainers to find bugs is by using the Python buildbot, an automated tool that routinely builds and tests Python across different environments (i.e., systems, architectures, and configurations).[11] When Python fails

to run in an environment, the buildbot generates error messages, prompting Python maintainers to investigate the causes and file bug reports.

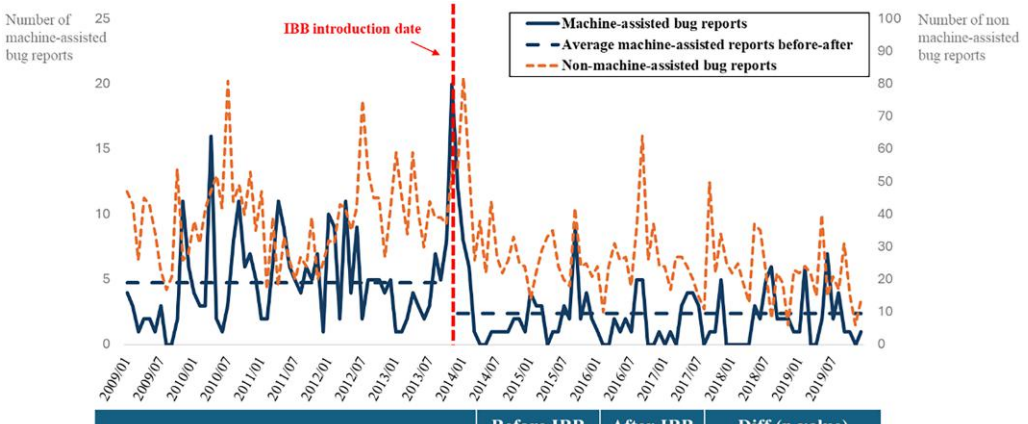
Because the buildbot mechanically tests Python, its activity should not be susceptible to maintainers' incentives. The main determinant of the number of bug reports prompted by the buildbot (which we call "machine-assisted bug reports") is the objective difficulty of finding bugs in Python. If the IBB made bugs more difficult to find, then we would expect the buildbot to detect fewer errors and prompt fewer machine-assisted bug reports. Because Python maintainers typically reveal how a bug is discovered in their bug reports, we can identify machine-assisted bug reports by searching for the keyword "buildbot" in the titles or descriptions of these reports.

We plot the monthly number of machine-assisted bug reports in Figure 8 (solid line), which shows an evident decline in the number of machine-assisted bug reports after the IBB, indicating increased difficulty in discovering bugs. The *t*-test in the summary table shows that this decline is statistically significant. Additionally, the dashed orange line in Figure 8 and the regression analysis in Online Appendix C.4 show that Python maintainers also reduced their contributions of non-machine-assisted bug reports. Overall, these results suggest that finding bugs in Python has become more difficult after the IBB implementation, providing a plausible explanation for the decline in the bug reports contributed by Python maintainers.

6.4.2. Complementarity Between Bug and Enhancement Reporting. Hypothesis 2B posits that if Python maintainers' bug and enhancement reporting are complementary, such that their productivity of enhancement reporting is lower (higher) when their bug reporting activity is lower (higher), then their reduced bug reporting after the IBB will negatively impact their enhancement reporting. This section offers evidence for this complementary relationship. Following the literature (e.g., Brynjolfsson and Milgrom 2013, Wu et al. 2020, Gong and Png 2024), we examine complementarity by testing whether Python maintainers produce more enhancement reports when they engage more in bug reporting. We construct a Python maintainer monthly contribution panel and regress these maintainers' enhancement report outputs on their bug reporting activities while controlling for the maintainer and time fixed effects and other control variables. Column (1) of Table 7 suggests that a higher level of bug reporting activity is associated with a higher output of enhancement reports, providing support that bug and enhancement reporting are complementary.

However, time-varying unobservable factors may pose a concern. For example, maintainers may contribute more reports during their free time (such as holidays) and fewer reports when they are busy, which could lead to spurious correlations between their bug and enhancement reporting. To address such endogeneity, we apply an instrumental variable (IV) strategy, where the IV is the number of errors detected by the Python buildbot in each month. As discussed above, the buildbot mechanically tests Python, so the errors detected by the buildbot should be independent of the individual maintainer's schedule. However, these errors should motivate Python maintainers to investigate the underlying causes, which can lead them to compose bug reports.[12] Aside from prompting maintainers to investigate bugs, these buildbot-generated errors are unlikely to directly impact maintainers'

Figure 8. (Color online) Trends in the Number of Machine-Assisted and Non-Machine-Assisted Bug Reports in Python



	Before IBB	After IBB	Diff (p value)
Monthly machine-assisted bug reports (avg.)	4.79	2.39	-2.40*** (<0.001)

Notes. The graph shows that machine-assisted and non-machine-assisted bug reports have decreased after the IBB. We extend the plot to January 2009 to illustrate a longer pre-IBB trend (other thresholds yield consistent results). ***$p < 0.01$.

Table 7. Complementarity Between Bug and Enhancement Reporting

Variables	(1) Enhancement reports (fixed effects)	(2) Enhancement reports (buildbot errors as IV)	(3) Enhancement reports (buildbot errors as IV, limit to >2 months after version update)
Bug reports	0.319***	0.731***	0.781***
	(0.033)	(0.189)	(0.227)
Controls	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes
Observations	7,122	7,122	5,844
Number of maintainers	92	92	92

Notes. All 92 Python maintainers are included in the estimation. Robust standard errors clustered by maintainers are reported in parentheses.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

enhancement reporting. Column (2) of Table 7 reports the IV regression result. The Kleibergen-Paap Wald statistic (Kleibergen and Paap 2006) of 13.16 in the first-stage regression suggests that the IV is not weak. With this IV, we continue to find a positive and statistically significant coefficient, reinforcing the notion that bug and enhancement reporting are complementary activities. In column (3), we perform another test to address a concern that the above result is driven by observations during Python major updates, which might affect bug and enhancement reporting. We exclude all observations within two months following Python major updates and obtain a similar coefficient estimate, suggesting that our results are not driven by Python updates (using other thresholds leads to similar findings). We obtain similar results by randomly excluding 20% of observations for 100 times.

Additional evidence suggests that the complementarity between bug and enhancement reporting is robust. First, the survey in Section 6.4.3 shows that the majority of the Python maintainers believe that bug reporting facilitates their enhancement reporting. Second, we corroborate the complementarity test result using recent IBB payouts as an alternative IV (Online Appendix C.5). Finally, we examine the sensitivity of the IV results to the exclusion restriction assumption by using the "plausibly exogenous" tests from Conley et al. (2012), which confirm that our estimation results are robust even after accounting for the possible violation of the exclusion restriction assumption (Online Appendix C.5).

Taken together, our evidence consistently indicates that Python maintainers' bug and enhancement reporting are complementary. Python maintainers' output of enhancement reports is lower (higher) when their bug reporting activities are lower (higher), suggesting that the reduction in bug reporting after the IBB can have a negative impact on their enhancement reporting.

6.4.3. Post Hoc Analysis: Learning as the Potential Source of Complementarity. Although our hypothesis highlights learning as the potential driver of task complementarity, the implicit nature of learning poses measurement challenges. Previous research has tested the learning effect by examining whether the performance in one task improves as individuals complete other tasks, which is consistent with our analysis in Section 6.4.2 (Schilling et al. 2003, Mo et al. 2021). To substantiate the complementarity relationship and learning as the potential source of influence, we (1) directly survey maintainers for their opinions on the relationship between bug and enhancement reporting, and (2) review Python maintainers' reports, emails, and blogs for concrete examples.

6.4.3.1. Survey. We sent a short survey to 92 Python maintainers whose email addresses could be identified and received 13 responses, yielding a 14% response rate, which is higher than that achieved in recent studies (e.g., 6% in Wessel et al. 2020 and 11% in Constantino et al. 2023). Most of the respondents are active contributors, with 61.6% contributing over 13 bug reports and 53.8% contributing over 10 enhancement reports per year within the past three years. Cross-checking against the Python issue tracker reveals that these maintainers are among the top 10% most active maintainers. Consistent with the findings in Table 7, we observe a strong correlation (0.908, $p < 0.01$) between these maintainers' bug and enhancement reporting activities. Online Appendix C.6 provides the details of the survey.

When asked whether they feel that the skills in developing bug reports and enhancement reports are related, 53.8% of the respondents answered "agree" or "strongly agree," whereas 38.5% answered "somewhat agree." Most of them believe that the knowledge from bug reporting helps them develop enhancement reports, with 46.2% answering "agree" or "strongly agree" and the other 38.5% choosing "somewhat agree." None of the respondents disagreed with these two questions. These responses provide direct support for the notion that bug reporting provides knowledge and learning benefits to enhancement reporting.

A total of nine respondents (69%) shared additional insights on why bug reporting helps enhancement reporting. As shown in Table 8, the key reason mentioned by 55.6% of them (five respondents) is that bug reporting helps them gain a deeper understanding of the code and improve their development skills, which in turn benefits their enhancement reporting. This finding aligns with our hypotheses. Three respondents (33.3%) mentioned that bug reporting enhances their ability to write enhancement reports. One respondent mentioned that in some cases, bugs can be regarded as missing features. These findings are corroborated by a parallel survey of Java maintainers (see Online Appendix C.6).

6.4.3.2. Process, Concrete Examples, and Additional Tests. In Online Appendix C.7, we include examples of maintainers' contributions to illustrate the potential learning. The described intertask learning process suggests the following implication: because bug reporting provides maintainers with more opportunities to explore related code, the complementarity benefit may be stronger when maintainers propose enhancements for this related code. This reasoning aligns with the literature suggesting that the learning effect is stronger when individuals perform different but related tasks (Schilling et al. 2003). Supporting this implication, our analysis in Online Appendix C.8 shows that bug reporting mainly benefits enhancement reporting on related code (the enhancement ideas are about the same or related modules that were recently investigated by a Python maintainer in bug reports).

6.4.3.3. Additional Validation from Heterogeneous Effects. If task complementarity is the correct explanation, then the following implications emerge: (1) maintainers who demonstrated a greater reduction in their bug reporting should also demonstrate a greater reduction in their enhancement reporting because they lost more complementarity benefits, (2) maintainers whose enhancement reporting benefits more from the complementarity should be more affected, and (3) maintainers should not reduce their contributions to tasks that are not complementary to bug or enhancement reporting. The analysis in Online Appendix C.9 supports these predictions.

6.5. Overall Impact on Python Enhancement

From the buildbot analysis, we find that bugs became more difficult to find after the IBB, which is consistent with the IBB's positive security impact. We further explore the IBB's impact on Python enhancement.

We first examine the enhancement report contributions from all Python contributors. Figure 9 shows a significant reduction in monthly enhancement report contributions since the IBB (purple solid line). This reduction is mainly driven by a significant decrease in the enhancement reports contributed by maintainers (second row in the summary table and blue dashed line), which has not been sufficiently compensated by the slight increase in crowd contributions (third row in the summary table and gray dotted line). By contrast, the overall bug report contributions have remained stable, as the substantial increase in bug reports from peripheral contributors has nearly countered the decline in the reports from maintainers (fourth to sixth rows in the summary table).

We next measure Python contributors' attempts to add new modules, functions, or parameters to the Python codebase. Our first metric is the total number of commits related to such attempts (as seen in column (3) of Table 6). As shown in Figure 10 (first row of the summary table and green line), these commits have

Table 8. Summary of Python Maintainers' Comments

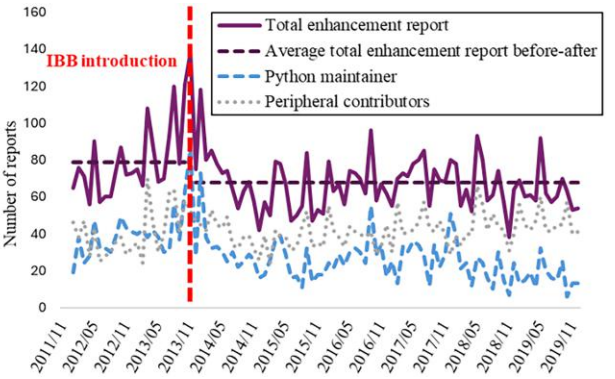
Summary	Sample comments from Python maintainers
Bug reporting helps understand codes and improve skills (55.5% of the respondents)	• Writing a good bug report requires digging deep to track down root causes. The knowledge gathered from writing bug reports has really helped me figure out how to write feature requests/enhancements properly. • You need to understand the code you are working on both when you are fixing it and when you are improving it. If you get familiar with it while fixing bugs, it will be easier to add enhancements, and vice versa. • I think the two kinds of reports require many of the same skills and so doing one helps improve skill at the other. For enhancement reports, the focus is more on explaining and justifying why a feature or change is a good idea. For bug reports, it is more about correctly identifying where the bug is located and what's wrong.
Bug reporting helps write better reports (33.3% of the respondents)	• Both practices force you to write clearly and concisely, for an audience that may be aware neither of your local environment and uses of Python … • Submitting bugs does help understand the basic process for submitting GitHub issues,[a] the basic information to fill out, how to interact with other contributors …

[a]Python started to use the GitHub issue tracker to receive issue reports in 2022.

Figure 9. (Color online) IBB on Total Enhancement Reports

	Before IBB	After IBB	Diff (p-value)
Monthly enhancement reports (avg.)	**79**	**68**	**-10.9*** (<0.01)**
By maintainers	38	26	-12.3***(<0.01)
By crowd	40	42	1.4 (0.555)
Monthly bug reports (avg.)	119	115	-3.9 (0.401)
By maintainers	48	29	-18.3*** (<0.01)
By crowd	72	86	14.4*** (<0.01)

Notes: *** p < 0.01, ** p < 0.05, * p < 0.1.



declined after the IBB. Given that maintainers contribute over 90% of such commits, the overall drop is primarily driven by the significant reduction in maintainers' contributions (contrasting rows 1–3 of the summary table). As a single commit may bundle many changes to different modules, some of which may not be about module, function, or parameter additions, we also analyze each individual change to a program file in a commit (known as a "diff" for a specific file) as the unit of analysis. As shown in Figure 10 (fourth row of the summary table and blue line), the results remain similar. We also obtain a similar finding if we weight each "diff" by the affected lines of code.

Finally, we conduct a module-level analysis by matching individual modules between Python and Java to compare commits at the module level. As shown in Online Appendix C.10, the finding is similar. Collectively, we find that the reduced enhancement report contributions by Python maintainers have negatively impacted Python's overall enhancement, including enhancement reports and enhancement-related code commit contributions.

7. Alternative Explanations and Robustness Checks

7.1. Alternative Explanations

7.1.1. Loss of Interest. An alternative explanation is that Python maintainers lost their interest in Python after the IBB because they felt unfairly treated, as their contributions were not rewarded, hence reducing their contributions.

To assess this explanation, we check the post-IBB contributions of Python maintainers who show a clear interest in Python. As discussed in Section 3, in the Python issue tracker, contributors can add themselves to nosy lists to receive updates on issues that interest them. Thus, we can use the maintainers' listing on the nosy lists of new reports as an indicator of their interest in Python (Python Software Foundation 2020). If a loss of interest in Python accounts for maintainers' reduced contributions, then we expect that the IBB should not affect the contributions made by maintainers on the nosy lists.

In columns (1) and (2) of Table 9, we estimate the effect of the IBB on the bug and enhancement report

Figure 10. (Color online) IBB on Commits/Diffs Potentially Related to Enhancements

	Before IBB	After IBB	Diff (p-value)
Monthly commits involving adding modules/functions/parameters	**73**	**56**	**-17.3*** (<0.01)**
By maintainers	73	45	-27.8***(<0.01)
By crowd	0.1	10.6	10.5*** (<0.01)
Monthly diffs involving adding modules/functions/parameters	**106**	**83**	**-23.0** (0.05)**
By maintainers	106	69	-37.1***(<0.01)
By crowd	0.1	14.2	14.1*** (<0.01)

Notes: *** p < 0.01, ** p < 0.05, * p < 0.1.

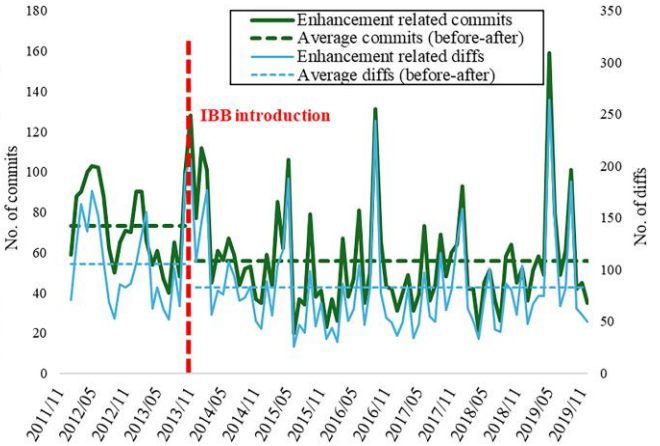


Table 9. Contribution Behavior Among Python Maintainers Who Indicate Interest

	Interest based on the nosy lists		Interest based on report quantity		Interest based on contribution months	
Variables	(1) Bug reports	(2) Enhancement reports	(3) Bug reports	(4) Enhancement reports	(5) Bug reports	(6) Enhancement reports
Python × IBB	−0.356***	−0.217***	−0.455***	−0.244***	−0.427***	−0.236***
	(0.060)	(0.044)	(0.074)	(0.059)	(0.074)	(0.058)
Controls	Yes	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes	Yes
Observations	9,355	9,355	6,707	6,707	6,911	6,911
No. of maintainers	150	150	76	76	78	78

Notes. The number of maintainers in columns (3)–(6) is slightly more than half of those in the main sample because of ties at the median. Robust standard errors clustered by maintainers are in parentheses.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

contributions of Python maintainers who are added on the report nosy lists in the same month.[13] The coefficient of Python × IBB remains negative, large, and statistically significant, suggesting that the maintainers' bug and enhancement report contributions have reduced even among those who are clearly interested in Python.

We corroborate these findings in several ways. (a) We regard the "remain interested" maintainers as those who contribute actively after the IBB (i.e., interest revealed by their actual contribution activities), specifically those maintainers whose total report contributions or active contribution months after the IBB exceed their group median. We apply these categorizations to both the treatment and control groups. As shown in columns (3)–(6) of Table 9, the interested maintainers significantly reduced their contributions after the IBB. (b) Given that other individuals can add a Python maintainer to the nosy list, we repeat our analyses in columns (1) and (2) under a stricter rule; that is, we regard Python maintainers as interested only when they voluntarily added themselves to this list. The results in Online Appendix C.11 remain similar. (c) The evidence in Online Appendix C.11 does not support the notion that the IBB has significantly influenced Python maintainers' interest as measured by their presence on the nosy lists of reports. Collectively, the substantial and significant reduction in the contributions of interested maintainers suggests that losing interest is unlikely to be the explanation for our findings. Because interest often directly reflects intrinsic motivation (Deci 1992), lack of motivation may also not be the primary driver of our findings.

7.1.2. Burden of Resolving Reports. Another alternative explanation is that Python maintainers are burdened by resolving too many reports after the IBB (e.g., fixing bugs or implementing proposed enhancements), which reduces their time or energy for proposing new bug or enhancement reports. Note that the total number of reports that Python maintainers need to resolve, including reports from themselves and peripheral contributors, need not increase because their own reporting has decreased. Furthermore, there is no formal requirement regarding the nature and amount of work from Python maintainers. If a Python maintainer does not wish to resolve reports, the person can choose not to do so (directly assigning reports to Python maintainers is uncommon). This means that Python maintainers do not necessarily spend more time resolving reports after the IBB.

We examine the number of reports resolved by each Python maintainer over time. We consider a Python maintainer to be responsible for resolving a report when the person appears in the "assignee" field in the bug or enhancement reports. To address measurement errors, we also consider a broader assignee definition to include maintainers who provide links to their code changes in report discussions as assignees. The rationale is that assignees are likely to provide updates on the fixing process to inform the community. This alternative approach accounts for situations when multiple maintainers work on the same report or when maintainers are not formally recorded in the assignee fields. We examine each maintainer's workload in resolving four types of reports, namely, bug and enhancement reports from peripheral contributors and Python maintainers, and we do the same for Java maintainers. We then use the monthly number of each type of report that a maintainer needs to resolve as the dependent variable in Equation (1).

Panels A (assignees based on the "assignee" field) and B (broader assignee definition) of Table 10 present the results, and column (1) of these panels presents the impact of the IBB on the total number of reports that these maintainers have resolved. The results indicate a drop in the total number of reports that Python maintainers need to resolve after the IBB. Columns (2)–(5) separate the results based on whether the assigned

Table 10. Workload in Resolving Reports

Variables	(1) Total assigned reports	(2) Assigned bug reports from peripheral contributors	(3) Assigned bug reports from maintainers	(4) Assigned enhancement reports from peripheral contributors	(5) Assigned enhancement reports from maintainers
			Panel A: Labeled as assignee		
Python × IBB	−0.186***	0.021	−0.162***	0.008	−0.078**
	(0.067)	(0.041)	(0.043)	(0.012)	(0.032)
			Panel B: Broader assignee definition		
Python × IBB	−0.186**	0.023	−0.168***	0.020	−0.076**
	(0.076)	(0.046)	(0.051)	(0.014)	(0.037)
Controls	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes
Observations	11,341	11,341	11,341	11,341	11,341
No. of maintainers	150	150	150	150	150

Note. Robust standard errors clustered by maintainers are reported in parentheses.
***$p < 0.01$; **$p < 0.05$; *$p < 0.1$.

reports are bug or enhancement reports and whether they were submitted by peripheral contributors or maintainers. The results do not support the notion that Python maintainers are burdened with resolving more reports after the IBB. Finally, recall Table 6, where we find that the IBB reduces Python maintainers' coding outputs, a metric that can incorporate all development effort of the maintainers, including report resolution. This finding does not support the explanation that maintainers diverted their effort toward addressing reports after the IBB.

7.1.3. Substitution of Enhancement Reports. Could the reduction in enhancement reports from Python maintainers be due to a substitution by the increased peripheral contributors' enhancement reports, similar to bug reports? Although the number of bugs in a software project is theoretically limited at any given time, enhancement ideas are unlimited. If a reduction in maintainers' enhancement reports is due to substitution, then we would expect the substitution effect to be weaker. However, the summary table in Figure 9 of Section 6.5 shows that although the peripheral contributors' monthly enhancement reports have slightly increased (third row in the summary table), the decrease in the maintainers' enhancement reports is much larger (second row in the summary table), which cannot be explained by the substitution effect.

Furthermore, if the decreased maintainers' enhancement reports are mainly substituted by peripheral contributors' enhancement reports, then we would expect maintainers to reduce more enhancement report contributions when there are more enhancement reports from peripheral contributors. However, our analysis in Online Appendix C.12 disproves this case. We instead find that a greater reduction in enhancement reports contributed by a Python maintainer is associated with a greater reduction in their contributed bug reports, supporting the complementarity explanation (Online Appendix C.9).

7.2. Robustness

As detailed in Online Appendix D, we validate the identification assumptions by testing parallel trends and assessing the stable unit treatment value assumption (SUTVA). We also conduct various robustness checks (summarized in Table D.1 of Online Appendix D). Our results are robust to different control groups (maintainers of Python third-party libraries), matching strategies (entropy balancing, synthetic DID, and additional matching covariates), sample construction (the balanced sample, the full sample, winsorized dependent variables, or exclusion of outlier maintainers), alternative quality measures (patch merge success), and estimation models (untransformed dependent variables or Poisson regression model). Finally, we consider various falsification tests (random treatment and pretreatment placebo tests) and show that two other alternative explanations, namely, report misclassification and maintainers submitting bug reports with fake accounts, are unlikely to explain our findings.

8. Concluding Remarks

Our study investigates how incentivizing crowd contributors impacts core members' contributions in a community-based complex knowledge production setting. Through a quasi-natural experiment, we find that incentivizing crowd contributions substitutes core members' contributions in the crowdsourced task. More importantly, such substitution negatively influences core members' nonincentivized knowledge creation without improving their contribution quality. Additional analyses suggest that this influence may

be driven by core members' reduced intertask learning. When the core members' contributions to the crowdsourced task serve as a knowledge source and experience that benefits related tasks, their substitution by crowd contributions can negatively influence these core members' contributions to other tasks.

Our findings reveal a managerial challenge in soliciting crowd contributions as the spillovers can be substantial. For instance, Python maintainers' enhancement report contributions decreased by an average of 38% after the IBB, leading to reduced overall contributions of enhancement reports and related code commits. Furthermore, it may be difficult to predict which tasks could be affected ex ante. More broadly, such spillovers provide implications beyond reward effects in OSS: in complex collaborative production systems, interventions that target one group's tasks can unintentionally trigger sizable, difficult-to-foresee intergroup spillovers that dampen other groups' contributions to complementary or downstream tasks.

8.1. Implications for OSS and Online Community Management

The significant economic benefits and efficiency of OSS make it appealing for governments and companies to sponsor incentives for OSS projects (Hoffmann et al. 2024). However, we caution against blanket incentivizing of crowd contributions without considering potential intergroup spillovers. One possible solution is to incentivize the core members, but monetary rewards can backfire and impede these members' intrinsic motivations to contribute (Shah 2006, Coelho et al. 2018). Incentives should align with core members' motivations (e.g., recognition, reciprocity, fun, and altruism), such as acknowledgment or symbolic rewards for their contributions (Gallus 2017). Given that spillovers can be difficult to foresee, we advise gradual implementation, starting from smaller and less important projects, to allow potential externalities to surface and for policies to adjust. More broadly, incentive policies for specific contributor groups or contributions should evaluate possible externalities holistically.

Our study highlights the learning-by-contributing effect in fostering online contributions. This is especially important for large projects such as Python, where few members master every aspect of the project and members often rely on making contributions to learn. Our findings suggest that prioritizing the speed of completing tasks over learning opportunities may not be optimal for the community. Coincidentally, Python maintainers sometimes intentionally mark some bug reports as "easy issues" for less experienced contributors to address. It is worth noting that efficiency and learning need not be in conflict: we may increase community productivity without sacrificing learning opportunities by promoting communication and collaboration and by providing training and educational opportunities (von Krogh et al. 2003).

8.2. Broader Managerial Relevance Beyond OSS

Although our findings on intergroup spillovers emerge from the OSS context, they point to a broader managerial challenge that may be relevant to other knowledge production systems: managing unintended spillovers associated with task delegation and labor substitution. Consider some examples:

8.2.1. Scientific Collaboration. In large research projects, delegating seemingly routine tasks to research assistants or specialized technical staff can accelerate data production but risks depriving senior researchers of direct engagement with critical data, tools, and methods. Such direct engagement can be vital for developing nuanced understanding and tacit knowledge that fuel significant discoveries and cross-domain innovations (Cetina 1999, Latour et al. 2013). When core researchers lose direct learning opportunities, the field's collective capacity to generate innovations can be impaired.

8.2.2. Education. The widespread adoption of automated tutoring systems and outsourced teaching content creation has the potential to scale access and efficiency, but it may reduce instructors' involvement in designing materials and experimenting with teaching pedagogy. Such disengagement could degrade overall curriculum quality and erode long-term pedagogical innovation (Létourneau et al. 2025).

8.2.3. AI and Automation. AI can now substitute a wide range of complex human knowledge work, yet short-term efficiency gains from AI may come at the cost of eroding humans' learning of necessary skills, potentially affecting human capability and productivity in derived or complementary tasks. Recent empirical findings have a striking correspondence to this implication. For example, students relying on AI for writing or math perform worse in related non-AI-assisted tasks, in part because the use of AI can reduce brain connectivity (Kosmyna et al. 2025). Endoscopists using AI-assisted colonoscopy detect fewer adenomas in unaided procedures (Budzyń et al. 2025). Consultants using AI underperform when tasks fall outside AI's frontier (Dell'Acqua et al. 2026).

These examples suggest that intergroup spillovers are not confined to OSS communities but represent a broader phenomenon in collaborative knowledge production systems where task complementarity and intertask learning are critical. Our research underscores the need to evaluate strategies that could lead to labor substitution—be they new incentives, outsourcing, or automation tools such as AI—not just for their

immediate effects on the targeted outputs, but for their broader, often-hidden impacts on other groups and activities.

8.3. Theoretical Implications and Conclusion

This study also offers several theoretical implications. First, our intergroup spillover findings imply that optimizing the outcomes of crowd contributions in crowdsourcing may not maximize the overall benefits. Crowdsourcing research has often conceptualized crowdsourcing as isolated tournaments (e.g., Terwiesch and Xu 2008, Chawla et al. 2019), focusing solely on optimizing crowd contributions. Intergroup spillovers suggest that the findings from the literature may not offer the "global optimum" in overall contributions. Specifically, we identify two types of externalities, *direct externalities* due to strategic substitutions and *indirect externalities* due to task complementarity, as a basis for further research on maximizing the overall benefits of crowdsourcing.

Second, our findings suggest that insights from other public goods contexts may not apply to OSS. For example, although studies on online Q&A platforms show a positive intertask spillover—where incentives increase nonincentivized contributions among recipients (Kuang et al. 2019, Wang et al. 2022)—we find that in OSS, the negative intergroup spillover (on core members) can ultimately reduce the overall nonincentivized contributions. This qualitative difference in implications, along with limited research in this domain, underscores the need for further investigations into the unique incentive effects in OSS.

Third, this study provides the first empirical evidence that BBPs can influence nonsecurity outcomes, broadening the focus of existing research and calling for attention to BBPs' unintended consequences when future research assesses their effectiveness. More broadly, our study extends the literature on the economies of learning by suggesting that scope and complementarity could be important for learning and that seeking external help may not benefit the learner. Future research should rigorously assess whether this result is general across other settings beyond crowdsourcing.

This study has several limitations. First, although we have employed multiple approaches, such as different control groups, different identification strategies, and a survey, to ensure that our findings are not driven by specific sampling strategies, differences between treated and untreated maintainers may still exist, which may bias our estimations of the IBB's impact. Second, although our theory implies that intergroup spillovers have general implications in settings characterized by division of labor and task complementarity, we cannot offer direct evidence in corporate settings. Future research may verify whether our insights hold in broader organizational contexts.

In conclusion, our study provides the first evidence showing that incentivizing crowd contributors can influence core members' work outcomes. Our study highlights the substitution between crowd and core member contributions and the core members' intertask learning in driving their reduced contributions. We offer important implications for crowdsourcing research and practice, the design of incentives for digital public goods, and discussions related to task delegation or labor substitution.

Acknowledgments

The authors thank the Department Editor, the Associate Editor, and three anonymous reviewers for their constructive feedback and guidance, which greatly enhanced the quality of the paper. The authors also thank Ivan Png, Liangfei Qiu, Jingyuan Li, and seminar participants at the Hong Kong University of Science and Technology, American University, and University of Mannheim for valuable comments and suggestions. This work is informed by our theory paper titled "Crowdsourcing from Hackers: Strategic Coopetition and Governance in Bug Bounty Programs."

Endnotes

[1] We have privately informed Python maintainers of our findings. The IBB program for Python has ceased since September 2021 (BusinessWire.com 2021).

[2] We examine whether Python maintainers report bugs in broader areas than the bugs they fix by comparing the Herfindahl-Hirschman index (HHI) of their bug reports and bug fixing (in terms of related modules). We find that their reported bugs are significantly more diverse (HHI = 0.241) than the bugs they fix (HHI = 0.458, Diff = 0.458 − 0.241 = 0.217, $p < 0.01$). We test whether bug reporting facilitates enhancement reporting in Section 6.4.2.

[3] The website is https://bugs.python.org, which migrated to GitHub issue trackers in 2022.

[4] The email discussion regarding the IBB by Python maintainers suggests that maintainers were not aware of this program before November 2013. See https://tinyurl.com/yh5m8pjc (accessed December 1, 2020). We corroborate this fact in Online Appendix A.5 using Google Search Index.

[5] According to the Python developers' guide, maintainers "represent the project and your fellow core developers whenever you discuss Python with anyone." See https://devguide.python.org/core-team/responsibilities (accessed March 1, 2023).

[6] Java maintainers refer to maintainers of OpenJDK. Similar to CPython, OpenJDK is the official open-source implementation of Java. The IBB does not include OpenJDK possibly because Oracle maintains a commercial Java implementation, OracleJDK. The relationship between OpenJDK and OracleJDK resembles that between Linux and Red Hat Linux (Lerner and Tirole 2002). OracleJDK's existence might violate the IBB's "Mind the Gap" criterion because Java has Oracle's support. Our results remain robust after controlling for various OracleJDK influence measures (Online Appendix D).

[7] Given the limited number of maintainers, we do not include all report quality measures for matching because Ripollone et al. (2020) recommend matching on a small vector of covariates to avoid heavy losses in sample size that would introduce more severe bias. Our results remain robust when matching on additional report quality measures (Online Appendix D).

[8] In trading off reducing skewness in data using variable transformations and the potential bias arising from the transformation, we choose the transformation path because contribution skewness is a more salient issue in online contribution contexts (Wang et al. 2022). We follow the literature (e.g., Coles et al. 2022, Chen and Roth 2024) to show that our results are robust without transformation or when using nonlinear models.

[9] The percentage change in y is calculated as $\beta\sqrt{1+1/y^2}$ (Bellemare and Wichman 2020), where β represents the estimated coefficient, and y represents the sample mean.

[10] There are fewer observations for quality outcomes than for quantity outcomes because quality outcomes are undefined in months when a maintainer did not contribute. Maintainer activity levels are comparable to other OSS projects such as Linux.

[11] A description of the Python buildbot is available at https://www.python.org/dev/buildbot/ (accessed September 1, 2023).

[12] Interestingly, buildbot and crowd bug reports impact maintainers' bug reporting differently. Buildbot prompts errors without directly generating bug reports, requiring Python maintainers to investigate the causes and compose bug reports themselves. By contrast, peripheral contributors directly contribute bug reports and often directly advise on the causes.

[13] The estimation is the treatment effect conditional on maintainers clearly expressing interest (i.e., inclusion in the nosy lists). As the nosy list information in the OpenJDK issue tracker is not public, in our estimation, we assume that Java maintainers are still interested in Java and thus include all Java maintainer-month observations in the sample.

References

Afuah A, Tucci CL (2012) Crowdsourcing as a solution to distant search. *Acad. Management Rev.* 37(3):355–375.

Akgul O, Eghtesad T, Elazari A, Gnawali O, Grossklags J, Mazurek ML, Votipka D, Laszka A (2023) Bug hunters' perspectives on the challenges and benefits of the bug bounty ecosystem. Calandrino J, Troncoso C, eds. *Proc. 32nd USENIX Conf. Security Sympos.* (USENIX Association, Berkeley, CA), 2275–2291.

Althuizen N, Chen B (2022) Crowdsourcing ideas using product prototypes: The joint effect of prototype enhancement and the product design goal on idea novelty. *Management Sci.* 68(4): 3008–3025.

Aral S, Brynjolfsson E, Van Alstyne M (2012) Information, technology, and information worker productivity. *Inform. Systems Res.* 23(3-part-2):849–867.

Arkhangelsky D, Athey S, Hirshberg DA, Imbens GW, Wager S (2021) Synthetic difference-in-differences. *Amer. Econom. Rev.* 111(12):4088–4118.

Austin PC (2009) Using the standardized difference to compare the prevalence of a binary variable between two groups in observational research. *Comm. Statist. Simulation Comput.* 38(6): 1228–1234.

Austin A, Williams L (2011) One technique is not enough: A comparison of vulnerability discovery techniques. *Proc. 2011 Internat. Sympos. Empirical Software Engrg. Measurement* (IEEE Computer Society, Washington, DC), 97–106.

Baek J, Shore J (2020) Forum size and content contribution per person: A field experiment. *Management Sci.* 66(12):5906–5924.

Bahar D, Choudhury P, Kim DY, Koo WW (2023) Innovation on wings: Nonstop flights and firm innovation in the global context. *Management Sci.* 69(10):6202–6223.

Baldwin C, von Hippel E (2011) Modeling a paradigm shift: From producer innovation to user and open collaborative innovation. *Organ. Sci.* 22(6):1399–1417.

Balmaceda F (2016) Optimal task assignments. *Games Econom. Behav.* 98(1):1–18.

Banker RD, Davis GB, Slaughter SA (1998) Software development practices, software complexity, and software maintenance performance: A field study. *Management Sci.* 44(4):433–450.

Bellemare MF, Wichman CJ (2020) Elasticities and the inverse hyperbolic sine transformation. *Oxford Bull. Econom. Statist.* 82(1):50–61.

Bessen J (2006) Open source software: Free provision of complex public goods. Bitzer J, Schröder PJH, eds. *The Economics of Open Source Software Development* (Elsevier, Amsterdam), 57–81.

Boehm BW (2002) Software engineering economics. Broy M, Denert E, eds. *Software Pioneers* (Springer, Berlin), 641–686.

Boudreau KJ, Lacetera N, Lakhani KR (2011) Incentives and problem uncertainty in innovation contests: An empirical analysis. *Management Sci.* 57(5):843–863.

Brynjolfsson E, Milgrom P (2013) Complementarity in organizations. Gibbons RS, Roberts J, eds. *The Handbook of Organizational Economics* (Princeton University Press, Princeton, NJ), 11–55.

Budzyń K, Romańczyk M, Kitala D, Kołodziej P, Bugajski M, Adami HO, Blom J, et al. (2025) Endoscopist deskilling risk after exposure to artificial intelligence in colonoscopy: A multicentre, observational study. *Lancet Gastroenterology Hepatology* 10(10):896–903.

Burtch G, Hong Y, Bapna R, Griskevicius V (2018) Stimulating online reviews by combining financial incentives and social norms. *Management Sci.* 64(5):2065–2082.

BusinessWire.com (2021) HackerOne expands the Internet Bug Bounty to improve the collective security of software supply chains. Accessed March 1, 2022, https://www.businesswire.com/news/home/20210921005457/en/HackerOne-Expands-the-Internet-Bug-Bounty-to-Improve-the-Collective-Security-of-Software-Supply-Chains.

Cass S (2022) Top programming languages 2022. *IEEE Spectrum* (August 23, 2023), https://spectrum.ieee.org/top-programming-languages-2022.

Cetina KK (1999) *Epistemic Cultures: How the Sciences Make Knowledge* (Harvard University Press, Cambridge, MA).

Chawla S, Hartline JD, Sivan B (2019) Optimal crowdsourcing contests. *Games Econom. Behav.* 113(1):80–96.

Chen J, Roth J (2024) Logs with zeros? Some problems and solutions. *Quart. J. Econom.* 139(2):891–936.

Chen W, Jin F, Xue L (2022) Flourish or perish? The impact of technological acquisitions on contributions to open-source software. *Inform. Systems Res.* 33(3):867–886.

Coelho J, Valente MT, Silva LL, Hora A (2018) Why we engage in FLOSS: Answers from core developers. *Proc. 11th Internat. Workshop Cooperative Human Aspects Software Engrg. CHASE '18* (Association for Computing Machinery, New York), 114–121.

Cohn JB, Liu Z, Wardlaw MI (2022) Count (and count-like) data in finance. *J. Financial Econom.* 146(2):529–551.

Coles JL, Heath D, Ringgenberg MC (2022) On index investing. *J. Financial Econom.* 145(3):665–683.

Conley TG, Hansen CB, Rossi PE (2012) Plausibly exogenous. *Rev. Econom. Statist.* 94(1):260–272.

Constantino K, Souza M, Zhou S, Figueiredo E, Kästner C (2023) Perceptions of open-source software developers on collaborations: An interview and survey study. *J. Software Evolution Process* 35(5):e2393.

Conti A, Gupta V, Guzman J, Roche MP (2023) Incentivizing innovation in open source: Evidence from the GitHub sponsors program. NBER Working Paper No. 31668, National Bureau of Economic Research, Cambridge, MA.

Dahlander L, O'Mahony S (2011) Progressing to the center: Coordinating project work. *Organ. Sci.* 22(4):961–979.

Deci E (1975) *Intrinsic Motivation* (Plenum Press, New York).

Deci EL (1992) The relation of interest to the motivation of behavior: A self-determination theory perspective. Renninger KA, Hidi S,

Krapp A, eds. *The Role of Interest in Learning and Development* (Lawrence Erlbaum Associates, Inc., Mahwah, NJ), 43–70.

Deci EL, Koestner R, Ryan RM (1999) A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. *Psych. Bull.* 125(6):627–668.

Dell'Acqua F, McFowland E III, Mollick E, Lifshitz H, Kellogg KC, Rajendran S, Krayer L, Candelon F, Lakhani KR (2026) Navigating the jagged technological frontier: Field experimental evidence of the effects of AI on knowledge worker productivity and quality. *Organ. Sci.* 37(2):403–423.

Dumont E, Fortin B, Jacquemet N, Shearer B (2008) Physicians' multitasking and incentives: Empirical evidence from a natural experiment. *J. Health Econom.* 27(6):1436–1450.

Fleming L (2001) Recombinant uncertainty in technological search. *Management Sci.* 47(1):117–132.

Fleming L, Waguespack DM (2007) Brokerage, boundary spanning, and leadership in open innovation communities. *Organ. Sci.* 18(2):165–180.

Foreman P (2019) *Vulnerability Management* (Auerbach Publications, Boca Raton, FL).

Fryer H, Simperl E (2017) Web science challenges in researching bug bounties. *Proc. 2017 ACM Web Sci. Conf.* (Association for Computing Machinery, New York), 273–277.

Gallus J (2017) Fostering public good contributions with symbolic awards: A large-scale natural field experiment at Wikipedia. *Management Sci.* 63(12):3999–4015.

Gong J, Png IP (2024) Automation enables specialization: Field evidence. *Management Sci.* 70(3):1580–1595.

Gousios G, Zaidman A, Storey MA, Van Deursen A (2015) Work practices and challenges in pull-based development: The integrator's perspective. *2015 IEEE/ACM 37th IEEE Internat. Conf. Software Engrg.*, vol. 1 (IEEE, New York), 358–368.

Greenstein S, Zhu F (2012) Is Wikipedia biased? *Amer. Econom. Rev.* 102(3):343–348.

Gu Z, Bapna R, Chan J, Gupta A (2022) Measuring the impact of crowdsourcing features on mobile app user engagement and retention: A randomized field experiment. *Management Sci.* 68(2):1297–1329.

Halstead MH (1977) *Elements of Software Science*, Operating and Programming Systems Series (Elsevier Science Inc., Amsterdam).

Harzevili NS, Shin J, Wang J, Wang S, Nagappan N (2023) Automatic static vulnerability detection for machine learning libraries: Are we there yet? *2023 IEEE 34th Internat. Sympos. Software Reliability Engrg. (ISSRE)* (IEEE, New York), 795–806.

He L, Luo J, Tang Y, Wu Z, Zhang H (2023) Motivating user-generated content: The unintended consequences of incentive thresholds. *MIS Quart.* 47(3):1015–1044.

Heimes C (2013a) Python at HackerOne [Email to python-committers mailing list]. Python.org. Retrieved December 1, 2022, https://mail.python.org/archives/list/python-committers@python.org/message/K2VZY5FKBGPWLOCU35MWGSO7TQSQAT2I/.

Heimes C (2013b) Python at HackerOne [Email to python-committers mailing list]. Python.org. Retrieved December 1, 2022, https://mail.python.org/archives/list/python-committers@python.org/message/MKPWUCYNOI6LT63FLQNXBVQC7FV573LS/.

Hoetker G (2007) The use of logit and probit models in strategic management research: Critical issues. *Strategic Management J.* 28(4):331–343.

Hoffmann M, Nagle F, Zhou Y (2024) The value of open source software. Working Paper No. 24-038, Harvard Business School, Boston.

Holmstrom B, Milgrom P (1991) Multitask principal-agent analyses: Incentive contracts, asset ownership, and job design. *J. Law, Econom., Organ.* 7(Special Issue):24–52.

Hui X (2020) Facilitating inclusive global trade: Evidence from a field experiment. *Management Sci.* 66(4):1737–1755.

Hwang EH, Singh PV, Argote L (2019) Jack of all, master of some: Information network and innovation in crowdsourcing communities. *Inform. Systems Res.* 30(2):389–410.

Iacus SM, King G, Porro G (2012) Causal inference without balance checking: Coarsened exact matching. *Political Anal.* 20(1):1–24.

Internetbugbounty.org (2013) Internet Bug Bounty charter. Retrieved February 1, 2022, https://web.archive.org/web/20161129192559/https://internetbugbounty.org/charter.html.

Jeppesen LB, Lakhani KR (2010) Marginality and problem-solving effectiveness in broadcast search. *Organ. Sci.* 21(5):1016–1033.

Jian L, Yang S, Ba S, Lu L, Jiang LC (2019) Managing the crowds: The effect of prize guarantees and in-process feedback on participation in crowdsourcing contests. *MIS Quart.* 43(1):97–112.

Jin Y, Lee HCB, Ba S, Stallaert J (2021) Winning by learning? Effect of knowledge sharing in crowdsourcing contests. *Inform. Systems Res.* 32(3):836–859.

Kaplan S, Vakili K (2015) The double-edged sword of recombination in breakthrough innovation. *Strategic Management J.* 36(10):1435–1457.

Khern-am-nuai W, Kannan K, Ghasemkhani H (2018) Extrinsic versus intrinsic rewards for contributing reviews in an online platform. *Inform. Systems Res.* 29(4):871–892.

Kleibergen F, Paap R (2006) Generalized reduced rank tests using the singular value decomposition. *J. Econom.* 133(1):97–126.

Koh TK, Cheung MY (2022) Seeker exemplars and quantitative ideation outcomes in crowdsourcing contests. *Inform. Systems Res.* 33(1):265–284.

Kosmyna N, Hauptmann E, Yuan YT, Situ J, Liao XH, Beresnitzky AV, Braunstein I, Maes P (2025) Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task. Preprint, submitted June 10, https://arxiv.org/abs/2506.08872v1.

Kuang L, Huang N, Hong Y, Yan Z (2019) Spillover effects of financial incentives on non-incentivized user engagement: Evidence from an online knowledge exchange platform. *J. Management Inform. Systems* 36(1):289–320.

Latour B, Woolgar S, Salk J (2013) *Laboratory Life: The Construction of Scientific Facts* (Princeton University Press, Princeton, NJ).

LaToza TD, Venolia G, DeLine R (2006) Maintaining mental models: A study of developer work habits. *Proc. 28th Internat. Conf. Software Engrg.* (Association for Computing Machinery, New York), 492–501.

Lee GK, Cole RE (2003) From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organ. Sci.* 14(6):633–649.

Lee HCB, Ba S, Li X, Stallaert J (2018) Salience bias in crowdsourcing contests. *Inform. Systems Res.* 29(2):401–418.

Lerner J, Tirole J (2002) Some simple economics of open source. *J. Indust. Econom.* 50(2):197–234.

Létourneau A, Deslandes Martineau M, Charland P, Karran JA, Boasen J, Léger PM (2025) A systematic review of AI-driven intelligent tutoring systems (ITS) in K-12 education. *NPJ Sci. Learn.* 10(1):29.

Lindbeck A, Snower DJ (2000) Multitask learning and the reorganization of work: From Tayloristic to holistic organization. *J. Labor Econom.* 18(3):353–376.

Lysyakov M, Viswanathan S (2023) Threatened by AI: Analyzing users' responses to the introduction of AI in a crowd-sourcing platform. *Inform. Systems Res.* 34(3):1191–1210.

MacCormack A, Verganti R, Iansiti M (2001) Developing products on "Internet time": The anatomy of a flexible development process. *Management Sci.* 47(1):133–150.

Mishra R, Kar W, Khern-Am-Nuai W, Kannan KN (2022) Review writing: The social influence of "exemplar" reviews. Preprint, submitted December 2, http://dx.doi.org/10.2139/ssrn.4292180.

Mo J, Sarkar S, Menon S (2021) Competing tasks and task quality: An empirical study of crowdsourcing contests. *MIS Quart.* 45(4):1921–1948.

Monsell S (2003) Task switching. *Trends Cognitive Sci.* 7(3):134–140.

Nagaraj A (2021) Information seeding and knowledge production in online communities: Evidence from OpenStreetMap. *Management Sci.* 67(8):4908–4934.

Nagle F (2018) Learning by contributing: Gaining competitive advantage through contribution to crowdsourced public goods. *Organ. Sci.* 29(4):569–587.

Nakakoji K, Yamamoto Y, Nishinaka Y, Kishida K, Ye Y (2002) Evolution patterns of open-source software systems and communities. *Proc. Internat. Workshop Principles Software Evolution* (Association for Computing Machinery, New York), 76–85.

O'Mahony S, Ferraro F (2007) The emergence of governance in an open source community. *Acad. Management J.* 50(5):1079–1106.

Poetz MK, Schreier M (2012) The value of crowdsourcing: Can users really compete with professionals in generating new product ideas? *J. Product Innovation Management* 29(2):245–256.

Python Software Foundation (2020) Triaging an issue. Python Developer's Guide. Retrieved March 1, 2023, https://web.archive.org/web/20200330180154/https://devguide.python.org/triaging.

Qiao D, Rui H (2023) Text performance on the Vine stage? The effect of incentive on product review text quality. *Inform. Systems Res.* 34(2):676–697.

Qiao D, Lee SY, Whinston AB, Wei Q (2020) Financial incentives dampen altruism in online prosocial contributions: A study of online reviews. *Inform. Systems Res.* 31(4):1361–1375.

Quinn JB, Hilmer FG (1994) Strategic outsourcing. *MIT Sloan Management Rev.* 35(4):43–55.

Raymond E (1999) The cathedral and the bazaar. *Knowledge Tech. Policy* 12(3):23–49.

Reber AS (1989) Implicit learning and tacit knowledge. *J. Experiment. Psych. General* 118(3):219–235.

Reeve J, Deci EL (1996) Elements of the competitive situation that affect intrinsic motivation. *Personality Soc. Psych. Bull.* 22(1):24–33.

Reitzig M, Wagner S (2010) The hidden costs of outsourcing: Evidence from patent data. *Strategic Management J.* 31(11):1183–1201.

Ripollone JE, Huybrechts KF, Rothman KJ, Ferguson RE, Franklin JM (2020) Evaluating the utility of coarsened exact matching for pharmacoepidemiology using real and simulated claims data. *Amer. J. Epidemiology* 189(6):613–622.

Rogers RD, Monsell S (1995) Costs of a predictible switch between simple cognitive tasks. *J. Experiment. Psych.: General* 124(2):207–231.

Ryan RM, Deci EL (2000) Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *Amer. Psych.* 55(1):68–78.

Schilling MA, Vidal P, Ployhart RE, Marangoni A (2003) Learning by doing something else: Variation, relatedness, and the learning curve. *Management Sci.* 49(1):39–56.

Setia P, Rajagopalan B, Sambamurthy V, Calantone R (2012) How peripheral developers contribute to open-source software development. *Inform. Systems Res.* 23(1):144–163.

Shah SK (2006) Motivation, governance, and the viability of hybrid forms in open source software development. *Management Sci.* 52(7):1000–1014.

Swanson EB (1976) The dimensions of maintenance. *Proc. 2nd Internat. Conf. Software Engrg.* (IEEE Computer Society Press, Washington, DC), 492–497.

Terwiesch C, Xu Y (2008) Innovation contests, open innovation, and multiagent problem solving. *Management Sci.* 54(9):1529–1543.

Van Rossum G (1997) Comparing Python to other languages. Retrieved February 21, 2022, https://www.python.org/doc/essays/comparisons/.

von Hippel E (1994) "Sticky information" and the locus of problem solving: Implications for innovation. *Management Sci.* 40(4):429–439.

von Krogh G, Spaeth S, Lakhani KR (2003) Community, joining, and specialization in open source software innovation: A case study. *Res. Policy* 32(7):1217–1241.

Wang J, Ghose A, Ipeirotis P (2012) Bonus, disclosure, and choice: What motivates the creation of high-quality paid reviews? *Proc. 33rd Internat. Conf. Inform. Systems* (Association for Information Systems, Atlanta).

Wang J, Li G, Hui KL (2022) Monetary incentives and knowledge spillover: Evidence from a natural experiment. *Management Sci.* 68(5):3549–3572.

Wessel M, Serebrenik A, Wiese I, Steinmacher I, Gerosa MA (2020) What to expect from code review bots on GitHub? A survey with OSS maintainers. *Proc. XXXIV Brazilian Sympos. Software Engrg.* (Association for Computing Machinery, New York), 457–462.

Wu L, Hitt L, Lou B (2020) Data analytics, innovation, and firm productivity. *Management Sci.* 66(5):2017–2039.

Xu L, Nian T, Cabral L (2020) What makes geeks tick? A study of Stack Overflow careers. *Management Sci.* 66(2):587–604.

Ye Y, Kishida K (2003) Toward an understanding of the motivation of open source software developers. *Proc. 25th Internat. Conf. Software Engrg.* (IEEE, New York), 419–429.

Yu Y, Khern-am-nuai W, Pinsonneault A (2022) When paying for reviews pays off: The case of performance-contingent monetary rewards. *MIS Quart.* 46(1):609–626.

Zhang X, Zhu F (2011) Group size and incentives to contribute: A natural experiment at Chinese Wikipedia. *Amer. Econom. Rev.* 101(4):1601–1615.

Zhang S, Singh PV, Ghose A (2019) A structural analysis of the role of superstars in crowdsourcing contests. *Inform. Systems Res.* 30(1):15–33.

Zhou J, Hui KL (2019) Bug bounty programs, security investment and law enforcement: A security game perspective. *Proc. 2019 Workshop Econom. Inform. Security* (WEIS, Boston), 3–4.

Zimmermann T, Premraj R, Bettenburg N, Just S, Schroter A, Weiss C (2010) What makes a good bug report? *IEEE Trans. Software Engrg.* 36(5):618–643.