Online Appendix for "Incentivizing Crowds and Inter-Group Spillovers: Evidence from the Internet Bug Bounty Program"

Jiali Zhou,[1] Kai-Lung Hui,[2] Ohchan Kwon[2]

Jializ@american.edu, klhui@ust.hk, ohchankw@ust.hk

[1] Kogod School of Business, American University, Washington D.C.

[2] School of Business and Management, Hong Kong University of Science and Technology

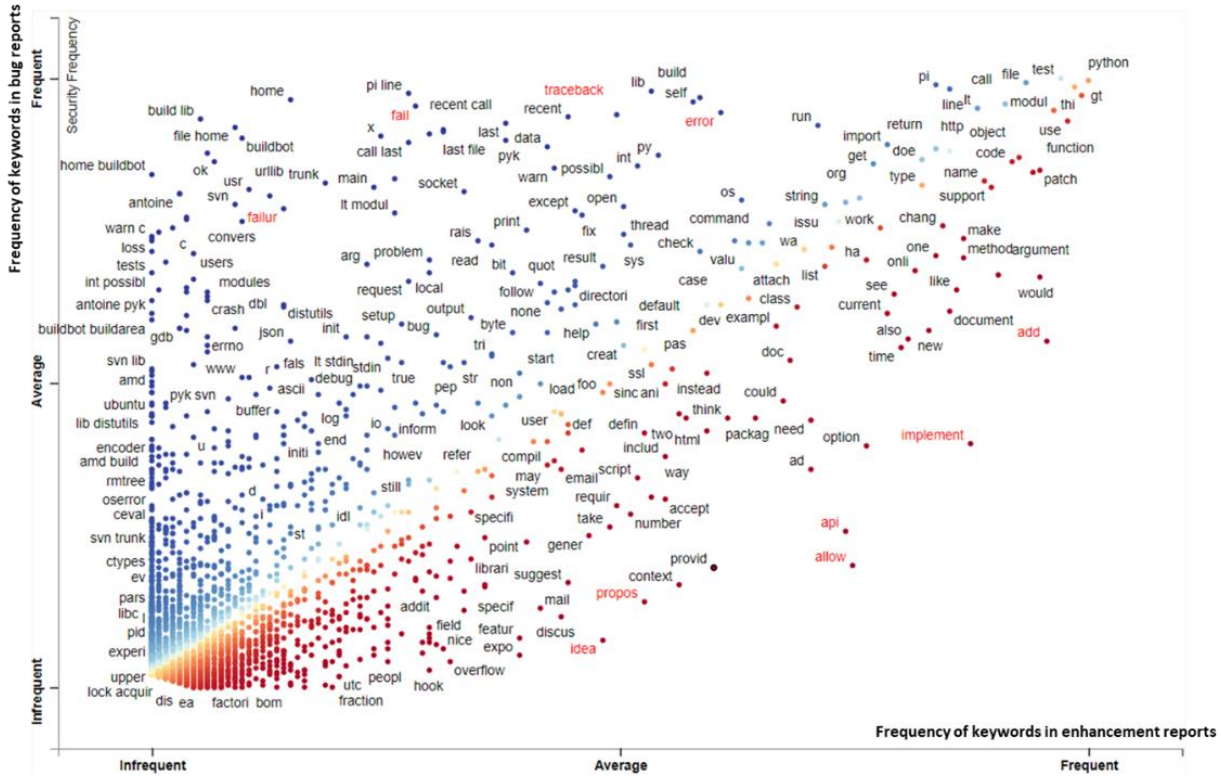
Appendix A: Additional information on the empirical context	- A.1 Keywords in bug and enhancement reports - A.2 Comparing maintainers and peripheral contributors - A.3 The first round of the IBB and the selection of Python - A.4 An illustration of the IBB process - A.5 Google trends index for the IBB
Appendix B: Data	B.1 More details on data collection and coding approach
Appendix C: Supplementary analyses	- C.1 Model-free evidence for changes in report quality measures - C.2 Programming languages used by popular websites - C.3 Additional analyses of code commits - C.4 IBB's impact on non-machine-assisted bug reports - C.5 Additional complementary tests - C.6 Survey - C.7 Anecdotal evidence for learning - C.8 Bug reporting and enhancement reporting on related codes - C.9 Validation from heterogeneous effects - C.10 Module level analysis - C.11 Additional analysis related to the loss of interest explanation - C.12 Assessing the substitution of enhancement report explanation
Appendix D: Robustness checks	See Table D.1 of Appendix D for a summary of robustness checks

Appendix A. Additional information on the empirical context

A.1 Keywords in bug and enhancement reports

Figure A1 compares Python bug and enhancement reports based on the keywords used in their descriptions. For each keyword, the x-value represents its frequency in enhancement reports. The y-value represents its frequency in bug reports. Keywords in the upper right corner, such as "python" and "module," are frequently used in both enhancement and bug reports. Keywords in the lower right corner, such as "implement," "api," and "allow," are frequently used in enhancement reports but not in bug reports. Keywords in the upper left corner are frequently used in bug reports, but not in enhancement reports. As reporters often paste error codes or error messages generated by the Python interpreter (which translates Python codes into executable machine codes) into the bug reports, their bug reports often contain error message keywords such as "fail," "traceback," and "recent call".

Figure A1: Keywords frequently used in bug and enhancement reports for Python



Notes: The graph is generated using Scattertext (Kessler 2017). Keywords in the upper left corner are frequently used in bug reports, but not in enhancement reports. Keywords in the lower right corner are frequently used in enhancement reports, but not in bug reports. Some keywords mentioned in the paper are highlighted.

A.2 Comparing maintainers and peripheral contributors

Table A1: Python maintainers versus peripheral contributors

	Python maintainers	Peripheral contributors
Definition	Members of the official maintenance team, responsible for guiding and coordinating Python development	Casual contributors who are not officially affiliated with Python
Privilege	• Source code commit access • Approve/reject the contributions of peripheral contributors • Decide whether a bug report is eligible for a bug bounty reward	-
Incentives for contribution	• Intrinsic desire to improve the project, fun or enjoyment • Full-time work[1]	• Opportunity to fulfill temporary needs for software function • Bug bounty
Nature of contribution	• More frequent and significant contributions • Long-term involvement	• Temporary contributions • Short-term involvement

A.3 The first round of the IBB and the selection of python

Table A2 lists all open-source software included in the first round of the IBB. We focus on the first round because it is unlikely that any maintainers and peripheral contributors of open-source software projects anticipated the implementation of the IBB before this period. In the time since the first round of the IBB, its scoping criteria have been published. Accordingly, maintainers of other open-source software that satisfies the "scoping criteria" may have altered their behaviors in anticipation of future rounds of the IBB, which makes it difficult to identify the causal impact of the IBB.

We selected Python for our study because of data availability and quality. Most open-source software in the IBB requires that bug reports be sent directly and privately to maintainers via email or the Hackerone platform; these reports are disclosed only when they are rewarded. This prevents us from observing the total number of reports, especially those submitted by maintainers. We are also unable to observe the number of reports submitted before the implementation of the IBB for these software projects. In contrast, most bug reports for Python are first submitted to a public platform, the Python issue tracker (https://bugs.python.org), and then the reporters visit Hackerone to claim their bug bounty rewards. Hence, we can observe most bug reports for Python. Although Python also accommodates private bug reports, its

[1] Two Python maintainers are paid on a full-time basis to maintain Python. See a discussion of this issue at https://discuss.python.org/t/official-list-of-core-developers/924 (accessed December 2021). Our results are robust to excluding these paid maintainers.

maintainers regularly disclose private reports on the same issue tracker (see an example in https://bugs.python.org/issue24522).[2]

Table A2: Open-source software included in the first round of the IBB

Software	Nature
Apache httpd	Web server
Nginx	Web server
Python	Programming language
Perl	Programming language
Ruby on Rails	Programming language
OpenSSL	A software library for secure communications
Django	A Python-based free and open-source web framework
Phabricator	A collection of software development tools

Notes: Information is obtained from https://internetbugbounty.org. Although PHP was originally included in the first round of the IBB, it was subsequently removed owing to communication failure (Walker 2020).

A.4 An illustration of the IBB process

The implementation of the IBB program is non-intrusive, meaning it does not interfere the usual bug reporting process of individual open-source projects. Consider an illustration of the IBB process. A Python user reported a potential vulnerability on 1 October 2016 through the Python issue tracker. In the days that followed, a Python maintainer responded to the report and wrote a fix for the vulnerability. There was no change in the process up to this step. With the implementation of the IBB, however, the user can now claim a bug bounty. Thus, on 11 October 2016, that user went to the Hackerone website to claim a bug bounty and received US$1,000 after two months.[3]

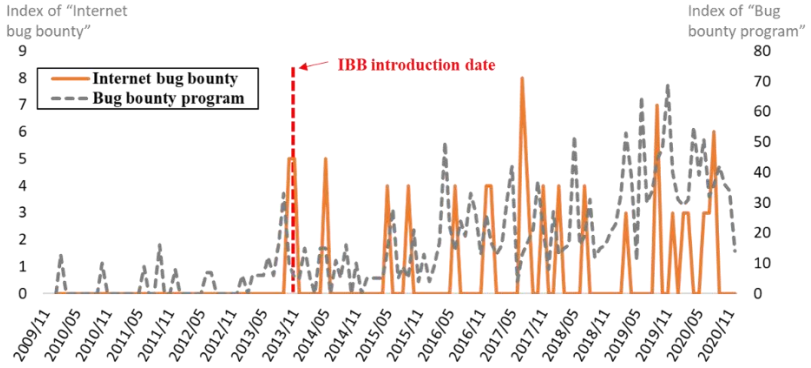
A.5 Google trends index for the IBB

We used the Google Trends Index (https://trends.google.com/trends) to check whether Python maintainers may have anticipated the implementation of the IBB from public information. Specifically, we searched for the keyword “Internet bug bounty” and found no sign that this program was disclosed publicly before the date on which the IBB was introduced (indicated by the solid orange line in Figure A2). As a baseline, the grey dotted line is the Google Trends Index for “bug bounty program,” a more general term that can also refer to other bug bounty programs. These patterns in Google Trends support the exogeneity of the IBB to Python maintainers.

[2] Perl also discloses most of its reports publicly on its issue tracker (https://github.com/Perl/perl5/issues). However, this issue tracker does not require reporters to specify the report type, so we cannot differentiate enhancement reports from bug reports and other types of reports.

[3] The original report is available at https://bugs.python.org/issue28322 (accessed December 2022). Records of bug bounty rewards are available at https://hackerone.com/reports/175091 (accessed December 2022).

Figure A2: Google Trends Index for the terms “Bug Bounty Program” and “Internet Bug Bounty”



Notes: Data are obtained from Google Trends (https://trends.google.com/trends). The value of the Google Trends Index for the term “Internet bug bounty” (orange solid line) increases around the IBB introduction date.

Appendix B: Data

B.1 More details on data collection and coding approach

Identify maintainers reports: As illustrated in Figure B1, the Python issue tracker provides a label (i.e., the Python logo) to flag Python maintainers' reports, which allows us to distinguish maintainers' reports from peripheral contributors' reports. We note that, in some cases, Python maintainers may use their own account to disclose bug reports from peripheral contributors (who may directly send private bug reports to maintainers); therefore, using the label would lead us to misclassify such reports as maintainers' contributions. As Python maintainers typically acknowledge the source of such reports in the description, we manually read all bug reports to determine whether a report is a disclosure of peripheral contributors' report to minimize such potential measurement errors. We also note that the total number of such reports (41, 1.2% of all maintainers' reports) is small, so it is unlikely that they would have a significant effect on our findings.

Figure B1: Labels for maintainers' reports

Bug and enhancement report: We collected all Python bug and enhancement reports submitted before January 2020 from the Python issue tracker (https://bugs.python.org/). We collected information about the report type, author, creation date, description, assignee, and nosy list for each report (see Figure 2 of the paper). As discussed above, we can separate the maintainers' reports from peripheral contributors' reports Python issue tracker. We sum the number of bug and enhancement reports separately for each individual maintainer in each month.

The Python issue tracker allows users to contribute eight types of reports: "behavior," "enhancement," "compile error," "crash," "security," "performance," "resource usage," and "blank" (that is, the type field is left blank). We classified reports labeled "behavior," "crash," and "security" as bug

reports because Python maintainers recommend using these labels when submitting a bug report.[4] We directly classified reports labeled "enhancement" as enhancement reports. We did not include reports labeled "compile error," "performance," and "resource usage" in our analysis because the number of such reports is small (they account for 1.7 percent, 3.7 percent and 2.3 percent of all reports, respectively), and they do not fit into the bug and enhancement categories.

The Python issue tracker also enables users to submit a "blank" type report if the scope of its content does not fall under the other seven report types.[5] These reports may variously contain issues pertaining to Python websites or frequently used tools, or issues about Python auxiliary files (Table B1 provide some example "blank" type reports). We use Python maintainers' monthly contributions of "blank" type reports to test one mechanism in Appendix D.

Table B1: Examples of "blank" type reports on Python

Topic of "blank" type report	**Examples report titles**
Issues pertaining to Python community websites/tools	www.pythontest.net is down (36423)
	Dead links in mailbox doc (38486)
	ftp://www.pythontest.net/ returns error 500 (35386)
	Buildbots fail when new files are added (37043)
Issues pertaining to Python auxiliary files	Misc/python-mode.el is out of date (10214)
	Use generic license web page rather than requiring release-specific license pages (21572)
	`make html` broken (33543)
	Duplicated sections in changelog (38004)
Records of changes in Python auxiliary files	Rewrite of Python 2/3 porting HOWTO (22913)
	Update OpenSSL to 1.1.0h / 1.0.2o (33184)
	Upgrade macOS and Windows installers to Tcl 8.6.9 and Tk 8.6.9.1 (35402)
Issues pertaining to Python internal use package[1]	Delete test importhooks (18158)
	Test signal.test_interprocess_signal() race condition (30358)
	Modernize several tests in test_importlib (37890)

Notes: 1. The Python test package is provided for internal use by Python maintainers only; see https://docs.python.org/3/library/test.html. Report IDs are in brackets. The reports can be accessed at https://bugs.python.org/issue{report id} (accessed October 2021).

Similarly, we collected Java maintainers' bug and enhancement reports from the OpenJDK issue tracker.[6] We identified Java maintainers from their source code commit history, as only maintainers have direct commit access to Java source codes (other contributors need to ask maintainers to help submit code commits) and the OpenJDK issue tracker does not directly label the type of reporters. The OpenJDK issue tracker contains five types of reports: "bug," "enhancement," "task," "sub-task," and "backport." Task, subtask, and backport reports are used by Java maintainers to manage workflows and are therefore irrelevant to our study (the "task" and "subtask" reports together account for less than 6 percent of all reports. "Backport" report represents a record of the bug fixing or feature implementation process; that is,

[4] Instructions for claiming an IBB reward are available at https://firebounty.com/648-python-ibb/ (accessed December 2023).
[5] https://web.archive.org/web/20200330180154/https://devguide.python.org/triaging/#type (accessed April 23, 2025).
[6] https://bugs.openjdk.java.net/projects/JDK/issues/ (accessed April 23, 2025).

applying a bug fix or enhancement implementation to earlier versions of the software). We directly used Java reports labeled “bug” and “enhancement” for our analysis.

Report length is measured by the number of non-empty characters in the report title and description. We use the information in the resolution field of a report to indicate whether a report is valid. If the resolution field reads “duplicate,” “not a bug,” “out of date,” “rejected,” “third party,” or “won’t fix,” then the report is considered to be invalid. Otherwise, a report is considered valid (the resolution field may read “accepted,” “fixed,” “later,” “postponed,” “remind,” etc.). The size of patches is measured by the lines of code contained in reporters’ provided patches.

Construction of the maintainer panel: Our main sample is representative because it includes all maintainers who experienced the implementation of the IBB. But this panel is unbalanced because maintainers may join and quit respective projects at different time. We code zero report contributions when maintainers do not submit reports but remain active, meaning that they continue to make contributions in the future. For maintainers who are considered to have quit (i.e., they do not have future contributions for at least one year), we do not code the zero contributions to avoid comparing quit and active maintainers (Goes et al. 2016, Law and Zuo 2021) and address a concern that these zeros after maintainers’ attrition are driving our results. We checked the robustness of our results by (1) appending a 12-month zero-contribution period after the last observed contributions (Appendix D), and (2) considering a balanced sample of maintainers who remain active throughout the observation period, as explained below.

The balanced panel consists of maintainers who remain active from January 2012 to December 2018 (those who joined before January 2012 and had not left after December 2018) and hence we have full observations during this period. We choose January 2012 as the starting month because it provides sufficient time to observe pre-IBB reporting activities while avoiding a significant loss of maintainers in the sample. We also repeat our analysis with different starting months and obtain similar results. We present the estimation results with the balanced sample in Appendix D.

Code commits and code commits potentially related to new functions or parameters: Following the literature (e.g., Chen et al. 2022), we collected code commits from Python and OpenJDK code repository (https://github.com/python/cpython and https://github.com/openjdk/jdk). We used the Git tool (https://git-scm.com/) to obtain detailed information for each commit, including the date of commit, author, committer (the one who integrates the code commit to the public repository, as noted in Section 3, only maintainers were granted commit access), and detailed code change. We then obtained each maintainer’s monthly commit contributions by aggregating code commits to the author-month level. In cases where a commit is authored by peripheral contributors but committed by a maintainer, we do not count the code commit as the maintainer’s contribution. For each code commit, we can obtain the line-by-line comparison of code changes through the Git "diff" command. This allows us to determine whether each code commit involves

adding modules, functions (including new classes) or parameters (but the rename of modules, functions or parameters is not counted). When a commit contains multiple changes to different program files, we also obtain each individual change (i.e., "diff" for a specific file) for one analysis in Section 6.5.

Decision density, Data density, and Halstead effort: Following Banker et al. (1998), decision density is measured by Cyclomatic complexity, which can be calculated as the number of decision points in a method plus one for the method entry. Decision density is zero if a commit does not involve adding new codes (e.g., cleanup old codes). Data density is measured by Halstead's N2 software science metrics (Banker et al. 1998), capturing the number of unique variables in a program. Halstead effort is calculated based on the formula in IBM (n.d.).

Bug and enhancement reports for third-party libraries: We obtained reports for third-party libraries (Numpy, Matplotlib and Scikit-learn)[7] from their respective issue trackers and maintainers from the projects' code commits history. As these issue trackers do not require reporters to specify the report type, we fine-tuned RoBERTa (Kallis et al. 2023), the state-of-the-art tool for classifying report types on Github issue trackers, to obtain the report type information. The original model is trained using 1.2 million labeled reports and we further fine-tuned the model based on 6,000 labeled reports from the three projects. Our fine-tuned model achieves precision of 93 percent for both bug and enhancement reports, which outperforms the original model (91 percent for bug reports and 89 percent for enhancement reports). We then constructed the maintainer monthly report panel in a similar manner to what we had done for the main sample.

Related enhancement reports: We first obtained all Python modules and their interdependencies through the Python import_deps library (https://pypi.org/project/import-deps/). We obtained the modules mentioned in maintainers' bug reports in each month. Then, for each enhancement report, we examined whether the module mentioned in the report is related to the modules mentioned in the bug reports contributed by the same maintainers in the same or previous month. When the modules are related, we label such enhancement reports as related enhancement reports and vice versa.

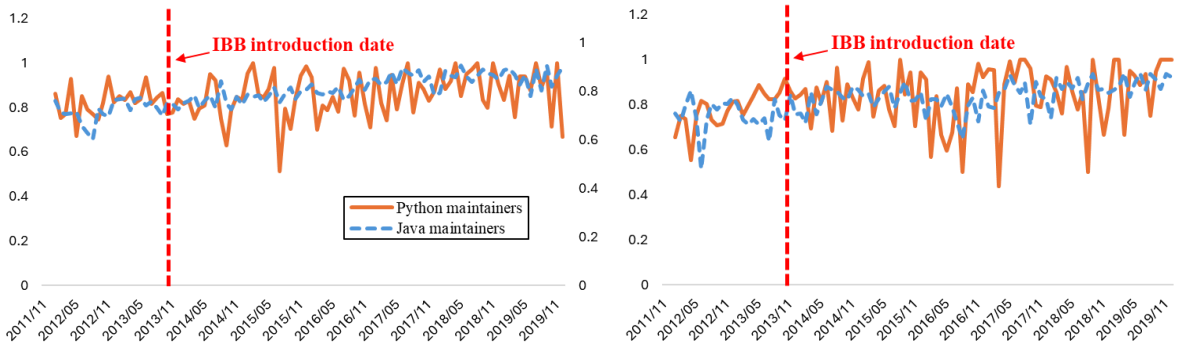
Control variables: We obtained the version information for Python and Java from https://endoflife.date/python and https://endoflife.date/oracle-jdk, and TIOBE index information from https://www.tiobe.com/tiobe-index/ . As TIOBE does not provide popularity information for Python third-party libraries, we used the number of asked questions related to each Python third-party library on Stack Overflow (the largest question-and-answer website for computer programming) to measure popularity for the third-party library sample.

[7] https://github.com/numpy/numpy/issues, https://github.com/matplotlib/matplotlib/issues, and https://github.com/scikit-learn/scikit-learn/issues (accessed April 23, 2025).

Appendix C. Supplementary analyses

C.1 Model-free evidence for changes in report quality measures

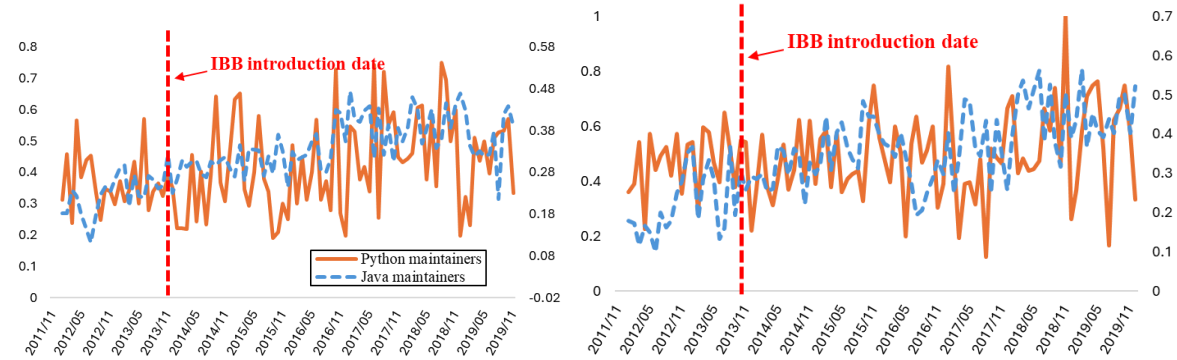
Figure C1: Ratio of valid reports by Python and Java maintainers over time



(a) Bug reports (b) Enhancement reports

Notes: To facilitate comparison of the trends in Panel (a), the valid ratio of Python maintainers' reports is scaled along the left y-axis, while that of Java maintainers' is scaled along the right y-axis.

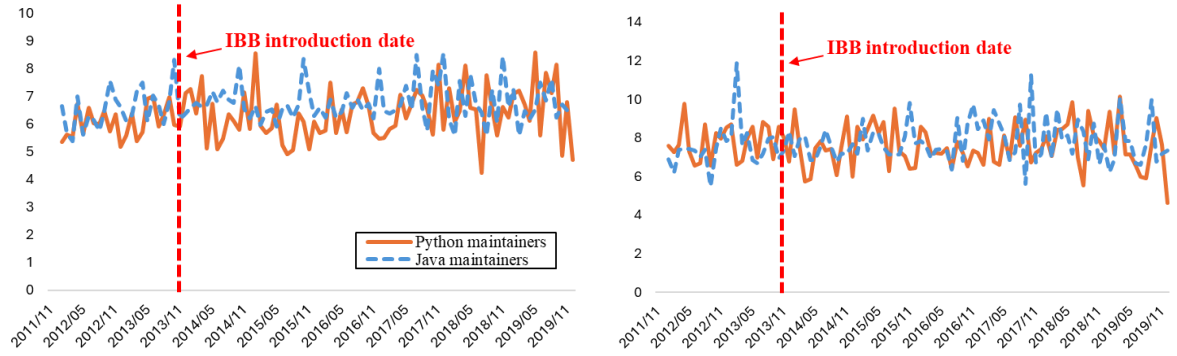
Figure C2: Ratio of reports containing patches by Python and Java maintainers over time



(a) Bug reports (b) Enhancement reports

Notes: To facilitate comparison of the trends in Panel (a) and (b), the ratio of Python maintainers' reports containing patches is scaled along the left y-axis, while that of Java maintainers' is scaled along the right y-axis.

Figure C3: Patch length (IHS transformed) by Python and Java maintainers over time



(a) Bug reports (b) Enhancement reports

C.2 Programming languages used by popular websites

Table C1: Programming languages used by popular websites

Website	Python	Java	C++	Go	C	Erlang	JavaScript	PHP	Hack	Ruby	C#
Google	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	No
YouTube	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No
Facebook	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	No
Yahoo	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No
Pinterest	Yes	No	No	No	No	Yes	No	No	No	No	No

Notes: Compiled from https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites (accessed December 2020). C++ and Go are also popular commercially. However, C++ does not have an official implementation. Go is mainly maintained by Google.

C.3 Additional analyses of code commits

We have examined the IBB's impact by using the ratio of commits to the number of reports[8] as the dependent variable. The negative Python×IBB coefficient in Column (1) of Table C2 is consistent with the finding in Column (2) of Table 6, suggesting that the IBB may have negatively influenced code commits beyond its impact on report numbers. We also examined the IBB's impact on Halstead total effort after controlling for the number of bug and enhancement reports. The result in Column (2) of Table C2 suggests a net reduction in maintainers' overall coding effort after accounting for the reduced report numbers.

C.4 IBB's impact on non-machine-assisted bug reports

As shown in Column (3) of Table C2, we find that the IBB also has a negative impact on Python maintainers' contributions of non-machine-assisted bug reports.

[8] We used overall bug and enhancement reports because maintainers may address others' reports; using maintainers' own reports yield qualitatively similar results.

Table C2: Additional results for code commits and mechanism

VARIABLES	(1) Ratio of commits to overall report numbers	(2) Halstead effort	(3) No. of non-machine assisted bug reports	(4) Enhancement reports (Recent bounty payout as IV)
Python × IBB	**-0.003****	**-0.776****	**-0.270*****	
	(0.001)	**(0.300)**	**(0.051)**	
Bug reports		0.383***		**0.823****
		(0.066)		**(0.386)**
Aderson–Rubin 95% C.I.				**[0.104, 2.337]**
Enhancement reports		0.572***		
		(0.096)		
Overall bug reports		0.001**		
		(0.000)		
Overall enhancement reports		-0.001		
		(0.001)		
Controls	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes
Observations	11,341	11,341	11,341	7,122
No. of maintainers	150	150	150	92

Notes: In Column (3), we exclude bug reports containing buildbot keywords. In Column (4), we use recent IBB payouts as an IV. Robust standard errors clustered by maintainers are in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

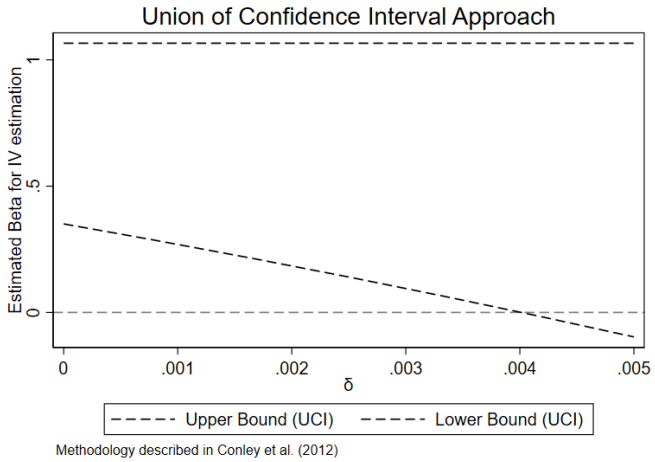
C.5 Additional complementary tests

Recent bounty payout as an alternative IV: Columns (4) of Table C2 reports the complementarity test results by using recent IBB payouts as an alternative IV. We expect higher bounty payouts to stimulate the crowd's bug reporting, which increases the competition for unknown bugs and decreases Python maintainers' bug reporting. Meanwhile, except through influencing bug reporting, IBB payouts should not directly impact maintainers' enhancement reporting. We construct the IV at time t using Python bounty payouts from the two preceding months (t−1, t−2), which are likely to have the greatest negative influence on Python maintainers' bug reporting (alternative timeframes yield similar results). The Kleibergen–Paap Wald statistic for this IV is a modest 6.1, so we report both the point estimate and the weak-instrument-robust Anderson–Rubin confidence intervals (Andrews et al. 2019). Both results support the complementary relationship between bug and enhancement reporting.

Plausibly exogeneous IV test: We used a plausibly exogeneous IV test (Conley et al. 2012) to assess the robustness of the IV result to the exclusion restriction assumption. The idea is to make inferences by allowing the instrument variable to have a nonzero direct effect on the dependent variable; that is, a possible violation of the exclusion restriction assumption. This allows the researchers to examine the sensitivity of their IV estimation to the exclusion restriction assumption (e.g., Frake and Harmon 2024).

Using this approach, we found that the effect of bug reporting on enhancement reporting remains significantly larger than zero at a 10 percent significance level as long as the direct (endogenous) effect of the instrument (i.e., buildbot errors) on maintainers' enhancement reporting is not more than 0.004 (as illustrated in Figure C4). As a reference, the reduced form effect of buildbot errors on maintainers' enhancement reporting is 0.007. This suggests that negating our results would require that most of the effect of buildbot errors on enhancement reporting occurs through a channel other than bug reporting (i.e., the direct effect should be larger than 0.004/0.007=57 percent), which is highly unlikely. This suggests that the complementarity effect of bug reporting on enhancement reporting is robust even after accounting for possible departures from exogeneity.

Figure C4: Sensitivity analysis of violation of exclusion restriction



Notes: x-axis represents the strength of violation of the exclusion restriction, which represents the direct (endogenous) effect of the instrument (buildbot errors) on maintainers' enhancement reporting. The graph shows that the violation must be larger than 0.004 (the point where the lower bound crosses zero on the y-axis) to make the IV estimate statistically insignificant at the 10 percent level.

C.6 Survey

Survey questions

As shown in Table C3, the survey includes four questions related to their contribution activities and one question concerning their preference for the incentive. The survey is designed to be succinct to minimize maintainers' inconvenience and increase the survey completion rate. We initiated the survey by sending out email invitations on October 14, 2023, and closed it on December 14, 2023.

Table C3: Survey

	Question	Answer
Q1	For each of the following kinds of reports, within the past three years, how many have you submitted per year on average?	1(<1/year), 2 (1~3/year), 3 (4~6/year), 4 (7~9/year), 5 (10~12/year), 6 (13~15 year), 7 (+15/year) at three dimensions; that is, Bug reports, enhancement reports, and other reports.
Q2	The skills in developing bug reports and enhancement reports for Python are related.	1 (Strongly Disagree) to 7 (Strongly Agree)
Q3	The knowledge I learn from developing bug reports help me in developing enhancement reports for Python.	1 (Strongly Disagree) to 7 (Strongly Agree)
Q4	Please tell us, in your opinion, whether and how developing bug reports help you in developing enhancement reports for Python. (optional)	Open text
Q5	Thanks for contributing to Python and completing the survey! We'd like to offer you a Starbucks gift card or a PSF donation as a token of our appreciation. Leave an email for us to send the gift card if you're interested. We will delete any email information once the gift card is sent. Email is not needed if you prefer a donation. Please also let us know if you have other comments.	Open text

Java survey

The survey for Java maintainers is similar to the Python survey except that we adjusted the keywords; for example, Q2 becomes "The skills in developing bug reports and enhancement reports for OpenJDK are related." We collected Java maintainers' email addresses from the commit history and found that they use official email addresses ending with @openjdk.org. We encountered email blockade (i.e., many delivery failure notifications after the initial attempts) when sending the email invitation to Java maintainers, and we eventually received a total of 16 responses.

Table C4: Survey responses

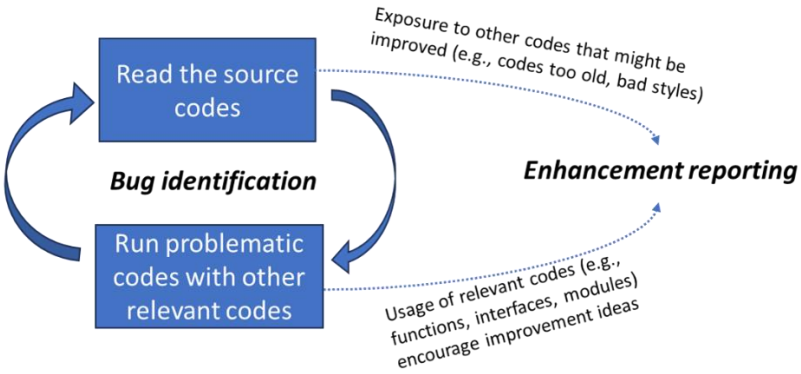
Questions	Python	Java
Q1	Bug reports: 15+/year (46.2%), 13~15 /year (15.4%), 4~6/year (30.8%), 1~3/year (7.7%) Enhancement Reports: 15+/year (38.5%), 10~12/year (15.4%), 7~9/year(15.4%), 4~6/year (15.4%), 1~3/year (15.4%) Other reports: 15+/year (38.5%), 13~15/year (7.7%), 10~12/year	Bug reports: 15+/year (37.5%), 10~12/year (6.3%), 7~9/year (6.3%), 4~6/year (12.5%), 1~3/year (37.5%) Enhancement Reports: 15+/year (35.7%), 13~15/year (7.1%), 10~12/year (7.1%), 4~6/year (21.4%), 1~3/year (21.4%), <1/year (7.1%) Other reports: 15+/year (8.3%), 4~6/year (8.3%), 1~3/year (33.3%), <1/year (50%)

	(7.7%), 7~9/year (7.7%), 4~6 (7.7%), <1/year (15.4%)	
Q2	Strongly agree: 3 (23.1%) Agree: 4 (30.8%) Somewhat agree: 5 (38.5%) Neutral: 1 (7.7%) Somewhat disagree: 0 Disagree: 0 Strongly disagree: 0	Strongly agree: 4 (37.5%) Agree: 5 (43.8%) Somewhat agree: 1 (6.3%) Neutral: 2 (12.5%) Somewhat disagree: 0 Disagree: 0 Strongly disagree: 0
Q3	Strongly agree: 4 (30.8%) Agree: 2 (15.4%) Somewhat agree: 5 (38.5%) Neutral: 2 (15.4%) Strongly disagree: 0 Disagree: 0 Somewhat disagree: 0	Strongly agree: 4 (18.8%) Agree: 2 (43.8%) Somewhat agree: 5 (31.3%) Neutral: 1 (6.3%) Somewhat disagree: 0 Disagree: 0 Strongly disagree: 0
Q4	See Table 8	12/16 (75%) provided responses **Bug reporting helps understand codes, improve skills (66.7%),** for example, - While chasing down the root cause, usually helps to come up ideas to improve/enhance code debuggability or improve code quality, etc - Bug report is a process to learn the details of the behavior and thus it is also help to find the potential enhancement. - A well written bug report often involves referencing parts of the JDK 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. - By fixing a bug, a developer can gain more understanding about a feature. With more knowledge about a feature, more ideas would come up to extend the feature, i.e. an enhancement. **Bug reporting helps write better reports (8.3%);** for example, - They definitely have some overlapping (e.g., clear description about the bug/problem) Two additional maintainers do not provide details and one additional maintainer mentions that bug reports may contain suggestions for enhancements.

C.7 Anecdotal evidence for learning

As illustrated in Figure C5, to identify bugs in Python, a maintainer must search extensively through the source codes and trace problematic pieces of codes. This search process can expose the maintainer to many other codes that may not cause immediate problems, but can be improved in other ways. Consequently, the maintainer may be motivated to submit a new enhancement report to improve these codes. Similarly, identifying bugs requires Python maintainers to run potentially problematic codes in different ways to ensure the codes do not behave unexpectedly. It is necessary for maintainers to develop test cases that demonstrate the problems (a typical bug report includes instructions to reproduce the problem in a test case). This process naturally involves using the potentially problematic codes together with other modules, interfaces, or functions (the "related codes"). The extensive use of related codes can give Python maintainers the opportunity to think of ideas for improving these related modules, interfaces, or functions.

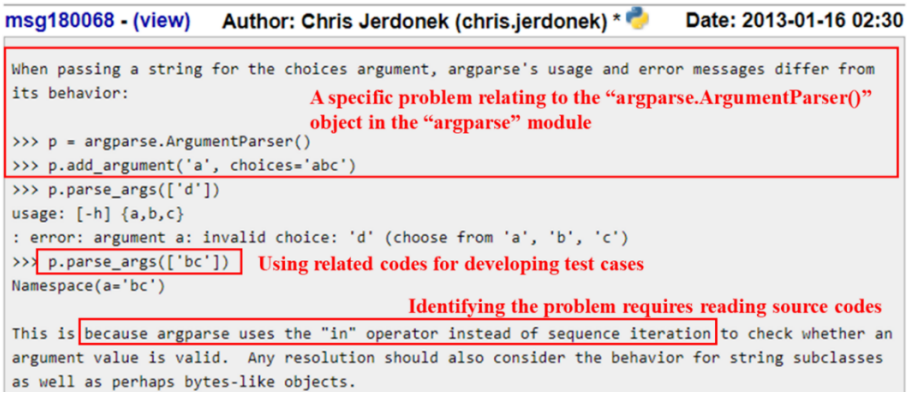
Figure C5: Underlying process of complementarity



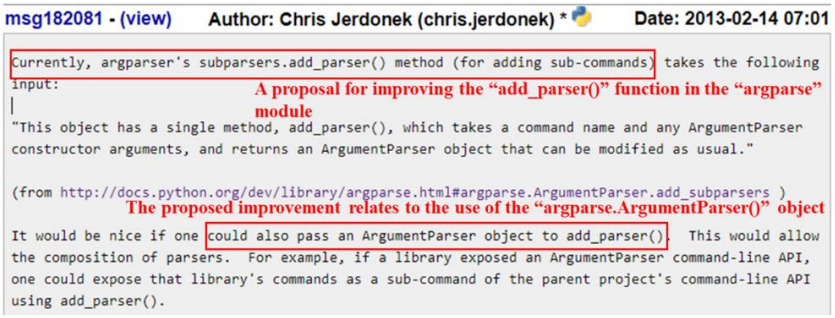
Anecdotal evidence: Figure C6, which includes a bug report and a related enhancement report, illustrates a specific scenario that reflects the aforementioned process. Figure C6a is a bug report, in which a Python maintainer identified a bug with the "argparse.ArgumentParser()" object (i.e., "when passing a string for the choices argument, argparse's usage and error message differ from its behavior"). To identify this bug, the maintainer might have done two things that could facilitate his/her follow-up enhancement reporting. First, he/she had to read through the codes in the "argparse" module to identify the cause of the problem (i.e., "argparse uses the 'in' operator [in the source codes]"), a process that would expose the maintainer to other functions in the same module that may have potentials for improvement. Second, the maintainer had to run the codes of "argparse.ArgumentParser()" in different ways to develop test cases, perhaps by using the "argparse.ArgumentParser()" object with the related modules, interfaces, or functions (e.g., using the "parse_args()" function to develop the test case). The maintainer's increased exposure to

the source codes and extensive usage of different modules, interfaces, or functions may give him/her the opportunity to contemplate ideas for enhancements.

Figure C6: Examples of complementarity in a Python maintainer's bug and enhancement reports



(a) A bug report by a Python maintainer detailing a problem with the "argparse.ArgumentParser()" object in the "argparse" module.



(b) An enhancement report by the same Python maintainer detailing a proposed improvement to the "add_parser()" function in the "argparse" module.

Notes: Figure a is an excerpt from this bug report: https://bugs.python.org/issue16977 (accessed December 2020). Figure b is an excerpt from this enhancement report: https://bugs.python.org/issue17204 (accessed December 2020).

Figure C6b shows that after submitting the bug report, the same Python maintainer proposed an enhancement report regarding the "subparsers.add_parser()" function found in the "argparser" module (i.e., the same module involved in the bug report in Figure C6a). Here, the maintainer suggested including "argparse.ArgumentParser()" (the earlier problematic object) as an argument in the function "subparsers.add_parser()." Had this Python maintainer not pursued the codes of the "argparse" module or explored the usage of "argparse.ArgumentParser()" with "subparsers.add_parser()", he probably would not have been able to generate this enhancement report. Table C5 lists more examples where contributing bug reports may provide maintainers with information that is useful for their contributions of enhancement reports.

Note that although our anecdotal evidence suggests that Python maintainers can acquire knowledge by searching and experimenting with the codes, it does not mean that they cannot learn from others' indirect experience (Argote and Miron-Spektor 2011), for example, by reading or fixing bug reports submitted by

peripheral contributors. However, as suggested in our discussion of H2, core members' direct involvement in bug reporting may benefit their enhancement reporting in ways that passively reading or fixing the crowd's bug reports may not afford. Furthermore, the indirect experience (gained from reading others' reports) can sometimes reduce a learner's engagement in direct learning experience, which is essential for the development of tacit knowledge (Reber 1989). For example, by reading a bug report submitted by a peripheral contributor, a Python maintainer may directly use the information provided in the report (e.g., steps to reproduce, test cases, or solutions) to locate and address bugs. This, in turn, may reduce that maintainer's opportunity to explore and experiment with the codes, thereby decreasing her opportunity to develop the tacit knowledge.

Table C5: Additional anecdotal evidence

Bug report	Enhancement reports
Fatal error on thread creation in low memory condition: local storage (8197)	Display Python backtrace on SIGSEGV SIGFPE and fatal error (8863)
cStringIO.StringIO aborted when more than INT_MAX bytes written (17054)	Test cPickle with real files (17299)
Resolve undefined behaviour in struct (30242)	Remove outdated checks in struct (30224)
Argument Clinic generates invalid code for optional parameter (20196)	Argument Clinic: meaningful names for group flags (20302)
Encoding errors in xmlrpc (26147)	Add parsing support for more types in xmlrpc (26885)
Possible refleaks in PyType_Ready in error condition (27248)	Replace size_t with Py_ssize_t as the type of local variable in list_resize (27660)
Argument Clinic: backslashes in docstrings are not escaped (20376)	Argument Clinic: meaningful names for group flags (20302)
SystemError or crash in sorted (iterable=[]) (29327)	Simplify argument parsing in sorted() and list.sort() (29331)
Stack overflow when parsing a long expression to AST (32758)	ast.literal_eval() should not accept booleans as numbers in AST (32893)
bytes and bytearray constructors replace unexpected exceptions (34974)	Improve error messages in bytes and bytearray constructors (34984)

Notes: This table lists bug or enhancement report titles. Report IDs are in brackets. They can be accessed at https://bugs.python.org/issue{report id} (accessed October 2021).

C.8 Bug reporting and enhancement reporting on related codes

To test the idea that the complementarity benefit may be stronger for maintainers to propose enhancements for related codes, we examined the impact of bug reporting on a maintainer's different types of enhancement reporting: enhancement reports on related codes and enhancement reports on unrelated codes. We consider an enhancement report to be about related codes when the enhancement idea is about the same or related modules that were recently investigated by a Python maintainer in bug reports (in the current or the previous month). We consider two modules to be related when one module imports or is imported by the other module.[9] We examine whether bug reporting affects these two types of enhancement reporting differently.

[9] In Python, "import" is the predefined keyword that makes codes in one module available in another. It is used when one module applies functions in the other module.

Comparing Columns (1) and (2) in Table C6, we find that bug reporting significantly enhances Python maintainers' productivity in generating enhancement reports on related codes, as suggested by the significant effects in Columns (1). For enhancement reporting on unrelated codes, the complementarity benefit becomes much weaker, as indicated by the smaller and statistically insignificant coefficient.

Table C6: Source of complementarity

VARIABLES	(1) Enhancement reports on related codes (Buildbot as IV)	(2) Enhancement reports on unrelated codes (Buildbot as IV)
Bug reports	**0.541*** **	**0.219**
	(0.129)	**(0.174)**
Controls	Yes	Yes
Maintainer fixed effects	Yes	Yes
Month fixed effects	Yes	Yes
Observations	7,122	7,122
Number of maintainers	92	92

Notes: The Kleibergen–Paap Wald statistic in the first stage is 13.2 for Column (1) and (2). Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

C.9 Validation from heterogeneous effects

Supposing task complementarity is the right explanation, the following implications emerge: *(1) Maintainers who had more reduction in bug reporting should have also had more reduction in enhancement reporting because they lost more complementarity benefit. (2) Maintainers whose enhancement reporting benefits more from the complementarity should be more affected. (3) Maintainers do not need to reduce their contributions to tasks that are not complementary with bug or enhancement reporting.*

These three predictions need not hold if our finding is driven by other explanations. To test Implication 1, we computed the extent of reduction in bug report contributions for each maintainer, denoted by BugReportReduction, from their monthly average number of bug reports contributed before and after the IBB implementation. We then included BugReportReduction as a moderator to assess whether the reduction in enhancement report contributions varies across maintainers.

Column (1) of Table C7 reports the regression results. The negative and statistically significant coefficient of the three-way interaction, Python×IBB×BugReportReduction, indicates that a greater reduction in the number of bug reports contributed by a Python maintainer is indeed associated with a greater reduction in the number of enhancement reports contributed by the maintainer, which is consistent with the change in Python maintainers' enhancement reporting behavior being related to task complementarity, thus supporting Implication 1.

Table C7: Heterogeneous effects

VARIABLES	(1) Enhancement reports (Moderated by BugReportChange)	(2) Enhancement reports (Complementarity score<0.16)	(3) Enhancement reports (Complementarity score>0.16)	(4) "Blank" type reports (level change after the IBB)
IBB ×Python	-0.123***	**-0.040**	**-0.232*** **	**0.022**
	(0.035)	**(0.030)**	**(0.065)**	**(0.034)**
Python × IBB × BugReportChange	**-0.039** **			
	(0.016)			
Controls	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes
Year and Month FE	Yes	Yes	Yes	Yes
Observations	10,894	5,631	5,710	7,122
Number of maintainers	142	74	76	92

Notes: BugReportChange represents the extent of reduction in bug report contributions (in percentage) for each maintainer, calculated based on their monthly average number of bug reports contributed before and after the IBB implementation. All two-way interaction terms have been included in the estimation. Column (1) contains 142 maintainers as BugReportReduction for eight maintainers is undefined because of zero bug report contribution before the IBB. We use the correlations before the IBB to calculate the complementarity score for Columns (2)–(3). Column (4) includes all 92 Python maintainers for the estimation. Robust standard errors clustered by maintainers in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$.

To test Implication 2, we infer the relative importance of complementarity to each maintainer's enhancement reporting based on their contribution pattern. If the complementarity between bug and enhancement reporting is important to facilitate a Python maintainer's enhancement reporting, the theory (Brynjolfsson and Milgrom 2013) suggests that their bug and enhancement reporting activities should be strongly correlated (i.e., more output of enhancement reporting with more bug reporting activities). Therefore, the strength of the correlation between a maintainer's bug and enhancement reporting can serve as an indicator of the relative importance of complementarity to that maintainer's enhancement reporting.

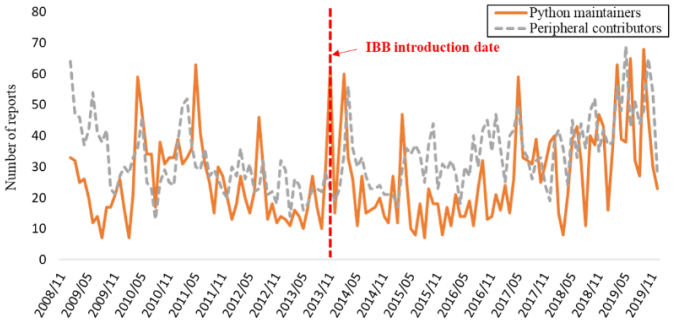
Based on this idea, we divided our sample of Python maintainers into subsamples based on the correlations between their bug and enhancement reporting activities before the IBB (denoted by their "complementarity score"). Our assumption was that maintainers with low correlations are likely to benefit less from complementarity. We check subsamples of Python maintainers whose complementarity scores are below 0.16 (median of complementarity score), and use values above 0.16 for comparison. If these correlations capture the importance of complementarity to a maintainer's enhancement reporting, we expect enhancement reporting by maintainers with low correlations to be less affected by the IBB implementation. As expected, Columns (2)–(3) of Table C7 show that, for Python maintainers whose scores are low (e.g., below 0.16), their enhancement report contributions are not significantly reduced (i.e., the coefficients of Python×IBB are insignificant and the effect is relatively small). The maintainers who substantially reduced enhancement report contributions are mostly those who benefit from task complementarity (i.e., maintainers whose complementarity scores are high).

To test Implication 3, we leverage the "blank" type reports. As discussed in Appendix B, "blank" type reports detail general issues in Python that cannot be regarded as any of the other pre-defined report types. Unlike bug or enhancement reports, many "blank" type reports are not directly related to Python functions, but address issues such as Python websites, development tools, or auxiliary files. We expect less complementarity between maintainers' bug and enhancement reporting activities and their reporting of "blank" type reports (this is supported by the non-significant correlations of 0.24 ($p = 0.11$) between bug reporting and the reporting of "blank" type reports).

We examined the effect of the IBB on Python maintainers' contribution of "blank" type reports. Because "blank" type reports are not allowed in the OpenJDK issue tracker, we used a simple before–after IBB comparison of Python maintainers' contributions of "blank" type reports after controlling for maintainer and time fixed effects. Our assumption was that the absence of a significant change in a maintainer's contribution level implies the IBB has no significant effect on Python maintainers' contributions of "blank" type reports. As expected, both the graphical evidence (Figure C7) and estimation in Column (4) of Table C7 show that Python maintainers' contribution of "blank" type reports have not reduced after the IBB implementation.

Collectively, these results lend further support to the explanation that task complementarity affects Python maintainers' contribution behavior after the IBB implementation.

Figure C7: Monthly "blank" type reports by Python maintainers and peripheral contributors over time



Notes: The graph shows the monthly number of blank reports by Python maintainers (orange solid line) and peripheral contributors (grey dotted line). This figure should be compared with Figure 5. There is no noticeable change in Python maintainers' blank report contributions, either in terms of overall trend or relative to peripheral contributors.

C.10 Module level analysis

For our module-level analysis, we leverage GraphCodeBERT—a state-of-the-art large language model for programming languages that captures both semantics and structure information of source code (Guo et al. 2020) —to identify comparable Python and Java modules that share similar characteristics in terms of pre-treatment size, age, average enhancement-related commits, and semantics. GraphCodeBERT is a multi-programming-lingual model and has been widely used in tasks such as code search, clone detection, and

translation, making it well-suited for identifying semantically similar modules (e.g., Hou et al. 2024). Specifically, we adopt a two-step matching procedure (Yu et al. 2022). We first apply GraphCodeBert to identify Python and Java modules that exhibit strong similarities in both textual content and structural properties based on source code embeddings, enforcing a cosine similarity threshold of 0.8 to ensure that only modules with high similarity are selected. After matching modules (with replacement) based on source code embeddings, we further apply CEM with additional criteria, including the module's pre-treatment size, age, and average monthly enhancement-related commits, to ensure the final matched sample of Python and Java modules exhibits no statistically significant differences in these observed characteristics before the IBB. Columns (1) and (2) of Table C8 show that, consistent with the findings in the paper, Python commits related to new modules, functions and parameters have decreased after the IBB. To enhance robustness of our findings, we also repeat the analysis using an alternative model, CodeBERT (Feng et al. 2020), and obtain consistent results.

Table C8: Module level analysis

	Module similarity based on GraphCodeBert		Module similarity based on CodeBert	
VARIABLES	(1) Commits related to new mod./func./param.	(2) Total commits	(3) Commits related to new mod./func./param.	(4) Total commits
Python × IBB	**-0.015***** **(0.002)**	**-0.037***** **(0.009)**	**-0.013***** **(0.002)**	**-0.027***** **(0.007)**
Controls	Yes	Yes	Yes	Yes
Module FE	Yes	Yes	Yes	Yes
Year and Month FE	Yes	Yes	Yes	Yes
Observations	138,506	138,506	133,567	133,567
No. of modules	1,268	1,268	1,246	1,246

Notes: The module panels are unbalanced different module creation dates. Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

C.11 Additional analysis related to the loss of interest explanation

Given that other individuals can add a Python maintainer to the nosy list, we repeat our analyses in Columns (1) and (2) of Table 9 under a stricter rule, that is, we regard Python maintainers as interested only when they voluntarily added themselves to this list. The results in Column (1) and (2) of Table C9 remain similar. In addition, we infer the change in Python maintainers' interest level reflected by their presence on the nosy lists of new reports each month. As shown in Column (3) of Table C9, we did not find a significant reduction in Python maintainers' presence on the nosy lists of reports after the IBB.

C.12 Assessing the substitution of enhancement report explanation

If the decreased maintainers' enhancement reports are mainly substituted by peripheral contributors' enhancement reports, we would expect maintainers to reduce more enhancement reports when there are

more enhancement reports from peripheral contributors. However, Column (4) of Table C9, using peripheral contributors' monthly enhancement report contribution as the moderator shows that this is not the case. By contrast, Column (1) of Table C7 shows that a greater reduction in bug reports by a Python maintainer is associated with a greater reduction in their enhancement reports, supporting the complementarity explanation.

Table C9: Additional tests for alternative explanations

VARIABLES	(1) Bug reports (Voluntarily listed on the nosy list)	(2) Enhancement reports (Voluntarily listed on the nosy list)	(3) Interest indicated by presence on the nosy list of new reports	(4) Enhancement reports
Python × IBB	**-0.354*** **	**-0.221*** **	**0.018**	-0.208***
	(0.065)	**(0.048)**	**(0.031)**	(0.073)
Python × IBB × CrowdEnhanceReports				**0.002**
				(0.002)
Controls	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes
Year and Month FE	Yes	Yes	Yes	Yes
Observations	8,698	8,698	11,341	11,341
No. of maintainers	150	150	150	150

Notes: In Column (1) and (2), we regard Python maintainers as interested only when they voluntarily added themselves to nosy lists. In Column (3), Java maintainers are included in the sample as we assume they remain interested in Java. The result is robust when comparing Python maintainers' interest before and after the IBB by excluding Java maintainers from the sample. In Column (4), CrowdEnhanceReports represents the monthly number of enhancement reports contributed by peripheral contributors. All two-way interaction terms have been included in the estimation. Robust standard errors clustered by maintainers are in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$.

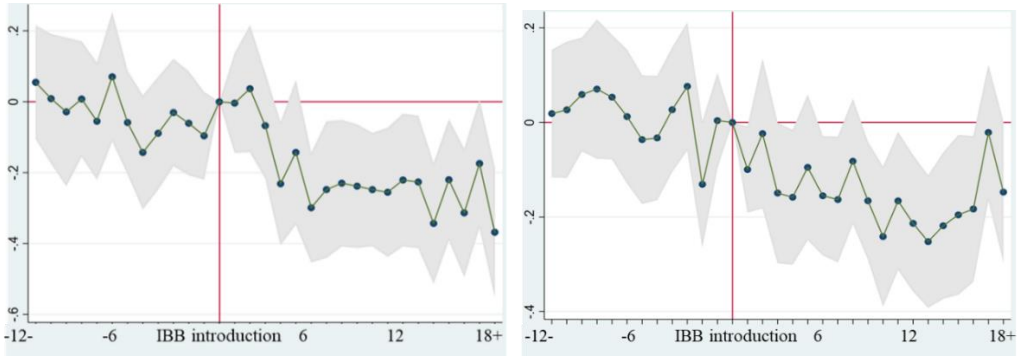
Appendix D. Robustness checks

Table D1: summary of robustness checks

Robustness checks		Key insights
Identification assumption	Parallel trends test	The reporting activities of Python and Java maintainers followed a similar trend before the IBB (Appendix D.1).
	Assessing SUTVA	We do not find any overlap maintainers, nor do we find a significant impact of the IBB on Java maintainers' contributions (Appendix D.2).
Identification strategy	Alternative control group (Appendix D.3), Entropy balancing (Table D4), synthetic DID (Table D4), Additional matching covariates (Table D5)	Results are similar.
Sample construction	Balanced sample (Table D5), full sample (Table D6), and other coding approach for zero contributions (Table D6)	
Outliers	Winsorized dependent variables (Table D6), or drop outlier maintainers (Table D7)	
Estimation models	Untransformed contribution numbers (Table D7) or Poisson regression (Table D7, binomial regression result is similar)	
Report quality measures	Patch merge success as an alternative quality measure (Tabel D8)	
Falsification	Placebo treatments and random inference tests	We find the likelihood that our results are due to coincidence is exceptionally low (Appendix D.5), and do not find significant effects for placebo treatments in time (Table D8).
Alternative explanation	Report misclassification	We find that the IBB has a negative effect on Python maintainers' enhancement reporting specifically about addition, support or implementation of new features (Appendix D.6).
	Maintainers cheating for reward	Python maintainers do not claim any IBB rewards, and we find that cheating for reward is difficult, and would bring substantial reputation risks for Python maintainers (Appendix D.7).
Sensitivity test	Sensitivity to OpenJDK development	Our results are robust to the consideration of Oracle JDK influences (Appendix D.8).

D.1 Parallel trends test: Following the literature (e.g., Burtch et al. 2018), we examine the month-by-month differences in the report numbers or lengths between Python and Java maintainers in the 12 months before and 18 months after the IBB implementation using the regression $Reports_{it} = \alpha_0 + \sum_{t=-12}^{18} \beta_t Python_i \times D_t + \alpha X_{it} + \tau_t + f_i + \epsilon_{it}$, where D_t denotes the time dummies, while β_t captures the month-by-month differences in reporting activity in the months before and after the treatment. Figures D1a and D1b plot the estimations for β_t (the values are reported in Table D2) with report number as the dependent variable. The coefficients are not statistically different from zero before the IBB and significantly decreased after the IBB.[10] Table D2 also reports the estimations with report length as the dependent variable, yielding coefficients not statistically different from zero both before and after the IBB. Therefore, the reporting activities of Python and Java maintainers followed a similar trend before the IBB, which supports the parallel trends assumption.

Figure D1: Dynamic effects of the IBB on Python maintainers' bug and enhancement reporting



(a) Bug reports (b) Enhancement reports

Notes: The shaded regions represent the 90 percent confidence intervals. We use November 2013, the IBB implementation month, as the reference point. The endpoints represent 12 (18) or more months before (after) the implementation of the IBB.

D.2 Assessing Stable Unit Treatment Values Assumption (SUTVA): One concern is the possible overlap between Python and Java maintainers, which could mean that Java maintainers were also treated and lead to a SUTVA violation. We checked the names of Python and Java maintainers (since maintainers typically disclose their real names) and found no such overlap. We also did not find any incidents where Python or Java maintainers claimed the IBB reward (including other IBB projects). Finally, we checked whether the IBB could have an impact on Java maintainers' contributions by examining Java maintainers' bug and enhancement report contributions near the IBB window (+/- 12 months. Other thresholds produce similar

[10] The jumps near the date on which the IBB was introduced may be driven by two explanations: (1) IBB publicity, which might have changed maintainers' behavior in the short term (as it was also the first time that Python maintainers learn about the program); (2) one maintainer's high contribution in t+2 (32 bug reports and 15 enhancement reports). Our results are robust to excluding two-month observations around the IBB implementation month or the outlier maintainer.

results). The coefficient estimation in Column (1) of Table D3 is small and insignificant, suggesting that the IBB did not have a detectable impact on Java maintainers' bug reporting behavior. This suggests SUTVA is likely to hold.

D.3 Alternative control groups: Although the model-free evidence suggests that our results are not driven by the control group (e.g., Figure 5, which does not rely on Java maintainers, shows a marked change in Python maintainers' reporting behavior after the IBB), we corroborate this conclusion by contrasting the contribution patterns of Python maintainers with those of alternative control groups. Note that the IBB only rewards bug reports in the Python core library but not in third-party libraries, making the maintainers of third-party libraries a suitable alternative control group. We consider maintainers for well-known Python third-party libraries in different application domains, including Numpy (scientific computing), Matplotlib (visualization), and Scikit-Learn (machine learning), as an alternative control group and repeat the analyses. To minimize the chances of violating the stable unit treatment value assumption (SUTVA), we exclude cross-project contributors, that is, those maintainers who contribute to both Python and third-party libraries or across multiple third-party libraries.[11] Following the same CEM matching approach, we obtain qualitatively similar results as reported in Columns (2)–(5) of Table D3, reinforcing that our results are not caused by the choice of a particular control group.

D.4 Alternative identification strategies, sample construction, measurement, and estimation models: Referring to the summary Table D1 and results from Table D4 to D8, the findings are consistent.

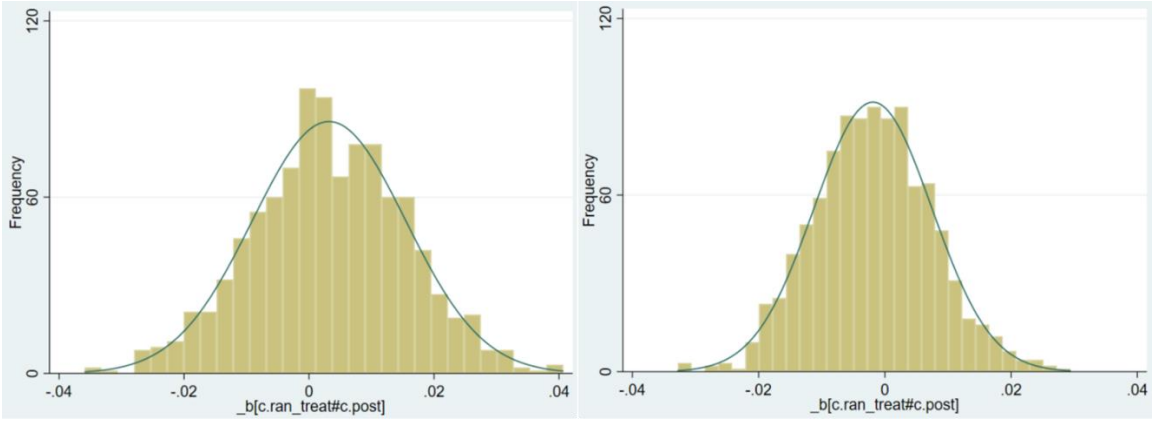
D.5 Falsification: We conducted two sets of falsification tests for our results: (1) a randomized treatment test (Bertrand et al., 2004), which randomizes the assignment of the treatment and control group; and (2) a placebo treatment test (e.g., Pamuru et al. 2021), which examines the impact of placebo treatments.

For the random inference test, we followed the literature (e.g., Cheng et al., 2020) by randomly generating and assigning placebo treatments to each maintainer. We then estimated the impact of the IBB on contributions based on these placebo treatments. We repeated the estimation 1,000 times, resulting in a distribution of the estimates for the placebo treatment effects (Figure D2). Comparing the distribution with the true point estimations (Columns (2) in Table 4 and 5), the likelihood that our estimation is due to coincidence is exceptionally low ($p<0.001$, point estimates are located outside of the two figure), providing evidence that the significant effect of the IBB on Python maintainers' contributions is valid.

[11] We have also followed prior research (e.g., Hong and Raudenbush 2006; Jo et al. 2020) to assess the influence of potential interference on our results (by controlling for the potential interference effect estimated from cross-project maintainers). We obtain consistent findings with this different way of dealing with potential violation of SUTVA.

For the pre-treatment placebo test, we followed Pamuru et al. (2021) to assign placebo treatment dates 6 and 12 months (other placebo treatments yield similar results) before the actual IBB implementation date and repeat the estimations. As reported in Columns (3)–(6) of Table D8, these placebo treatments do not have a significant effect on Python maintainers' bug or enhancement report contributions. The significant effect arises only after the IBB, thus supporting our results being driven by the IBB rather than coincidence. Note that the absence of effects during the period before the actual treatment date implies that the selected control group can reproduce the trajectory of the outcome variable for Python maintainers before the treatment. Hence, this result also supports the selection of Java maintainers as the control group.

Figure D2: Distribution of coefficients based on random treatments



Effects of random treatments on bug reports

Effects of random treatments on enhancement reports

Notes: the two figures show that the placebo treatment effects are distributed around zero, with true points estimations (-0.274 and -0.148 from Table 4 and 5) located outside the two figures, suggesting that our results are unlikely due to coincidence.

D.6 Report misclassification: One concern is that the reduction in Python maintainers' enhancement report contributions is driven by report misclassification. Some bug reports of maintainers may have been misclassified as enhancement reports owing to carelessness or the ambiguous nature of the content of these reports. However, Python maintainers generally have the expertise and incentive to properly label their reports, which means that such misclassification should occur rarely and is unlikely to systematically bias our results.

In addition, we pruned the enhancement reports and only retained those with keywords such as "add," "support," and "implement" to ensure that they detail enhancements ideas (i.e., they are specifically about the addition, support, or implementation of new functions, classes, and modules), and are therefore unlikely to have been misclassified. As reported in Column (7) of Table D8, we found that the implementation of the IBB has a similar effect on Python maintainers' contributions of these pruned enhancement reports.

D.7 Cheating or spending effort to other IBB projects: An additional concern is that some Python maintainers may use accounts other than their own to submit bug reports in exchange for the bug bounty. Nonetheless, Python maintainers value their reputation (Lerner and Tirole 2002), and "creating additional online accounts in order to … circumvent a ban" is explicitly identified as inappropriate behavior in Python.[12] Furthermore, to claim bug bounty rewards from Hackerone, bug reporters must use their real name to approve tax forms and publicize their reports to other maintainers.[13] In this regard, cheating on bug reports puts a maintainer's reputation at risk. Also, Python maintainers' use of other accounts to submit bug reports does not explain their reduced contributions of enhancement reports. We also checked the names of IBB awardees (including other IBB projects) and did not find any instances of Python maintainers being rewarded in other IBB projects. Hence, it is unlikely that our findings can be explained by Python maintainers' cheating for reward or diverting effort to other IBB projects.

D.8 The influence of Oracle JDK development: In Columns (1) to (6) of Table D9, we test the sensitivity of our results to Oracle JDK developments. Note that our identification strategy should have controlled for the time-invariant Oracle JDK factors through individual fixed effects. The key concern is time-variant Oracle JDK factors, which may affect our estimation. To alleviate this concern, we considered two indicators of Oracle JDK development for sensitivity analyses. (1) Google Search Index for Oracle JDK. We used this index to proxy Oracle JDK development, and included this index as a control variable for Java maintainers' reporting. Similarly, we used Google search index for Anaconda, the most popular commercial version of Python as a control variable for Python maintainers' reporting (Columns (1) and (2) in Table D9). (2) Oracle JDK license change. Since April 2019, Oracle JDK changed the license from BCL (Binary Code License), where users are licensed to use Oracle JDK for free, to OTN (Oracle Technology Network), which requires users to buy a license for commercial usage in production. We redid the estimation either by removing observations after April 2019 (Columns (3) and (4) in Table D9) or including Oracle JDK's license type as a control variable for Java maintainers' contributions (Column (5) and (6) in Table D9). As reported in Table D9, we found qualitatively similar results after the different ways of controlling for Oracle JDK factors, which suggests that our results are robust to the consideration of Oracle JDK influences.

[12] https://www.python.org/psf/conduct (accessed December 2020)

[13] The tax requirements for Hackerone are available at https://docs.hackerone.com/hackers/tax-forms.html (accessed November 2020).

Table D2: Relative time model

VARIABLES	(1) Bug reports	(2) Enhancement reports	(3) Bug report length	(4) Enhancement report length
$Python \times D^{-12}$	0.054	0.019	0.047	0.072
	(0.097)	(0.081)	(0.253)	(0.330)
$Python \times D^{-11}$	0.009	0.027	-0.218	-0.063
	(0.110)	(0.087)	(0.317)	(0.337)
$Python \times D^{-10}$	-0.029	0.059	-0.058	0.330
	(0.126)	(0.073)	(0.300)	(0.342)
$Python \times D^{-9}$	0.008	0.071	-0.329	-0.162
	(0.098)	(0.089)	(0.300)	(0.391)
$Python \times D^{-8}$	-0.055	0.053	-0.059	-0.210
	(0.100)	(0.079)	(0.313)	(0.324)
$Python \times D^{-7}$	0.058	0.013	0.037	0.197
	(0.109)	(0.085)	(0.244)	(0.331)
$Python \times D^{-6}$	-0.058	-0.037	-0.119	-0.014
	(0.088)	(0.082)	(0.264)	(0.349)
$Python \times D^{-5}$	-0.143	-0.033	-0.084	0.250
	(0.097)	(0.079)	(0.276)	(0.302)
$Python \times D^{-4}$	-0.088	0.027	-0.363	-0.034
	(0.096)	(0.079)	(0.249)	(0.349)
$Python \times D^{-3}$	-0.042	0.076	-0.092	0.245
	(0.090)	(0.081)	(0.267)	(0.317)
$Python \times D^{-2}$	-0.061	-0.131	-0.115	0.508
	(0.088)	(0.080)	(0.235)	(0.520)
$Python \times D^{-1}$	-0.095	0.004	-0.300	0.037
	(0.075)	(0.060)	(0.260)	(0.306)
Baseline				
$Python \times D^{1}$	-0.004	-0.100*	0.410	-0.134
	(0.085)	(0.055)	(0.274)	(0.460)
$Python \times D^{2}$	0.018	-0.023	0.158	0.115
	(0.108)	(0.095)	(0.254)	(0.335)
$Python \times D^{3}$	-0.067	-0.149*	-0.015	-0.434
	(0.090)	(0.090)	(0.335)	(0.328)
$Python \times D^{4}$	-0.231**	-0.158*	-0.116	-0.356
	(0.105)	(0.086)	(0.283)	(0.359)
$Python \times D^{5}$	-0.143	-0.095	0.104	0.038
	(0.123)	(0.093)	(0.260)	(0.427)
$Python \times D^{6}$	-0.299***	-0.147**	-0.220	-0.027
	(0.093)	(0.074)	(0.270)	(0.322)
$Python \times D^{7}$	-0.248**	-0.163**	0.359	-0.342
	(0.116)	(0.080)	(0.410)	(0.450)
$Python \times D^{8}$	-0.229**	-0.082	-0.008	0.073
	(0.108)	(0.080)	(0.259)	(0.339)
$Python \times D^{9}$	-0.238**	-0.166**	0.100	0.118
	(0.105)	(0.075)	(0.372)	(0.342)
$Python \times D^{10}$	-0.248**	-0.241***	0.229	0.019
	(0.096)	(0.089)	(0.265)	(0.468)
$Python \times D^{11}$	-0.254**	-0.166*	0.058	0.141
	(0.110)	(0.088)	(0.375)	(0.364)
$Python \times D^{12}$	-0.220*	-0.214**	-0.091	-0.276
	(0.113)	(0.087)	(0.300)	(0.380)
$Python \times D^{13}$	-0.226**	-0.252***	0.148	0.004
	(0.113)	(0.085)	(0.277)	(0.314)

$Python \times D^{14}$	-0.343***	-0.218**	-0.210	-0.516
	(0.102)	(0.093)	(0.304)	(0.366)
$Python \times D^{15}$	-0.219**	-0.195*	-0.019	-0.299
	(0.103)	(0.102)	(0.284)	(0.366)
$Python \times D^{16}$	-0.313***	-0.183*	0.253	-0.122
	(0.109)	(0.093)	(0.402)	(0.421)
$Python \times D^{17}$	-0.174	-0.021	-0.038	-0.021
	(0.109)	(0.086)	(0.335)	(0.325)
$Python \times D^{18}$	-0.368***	-0.147	0.016	-0.024
	(0.110)	(0.090)	(0.256)	(0.311)
Controls	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes
Year and Month FE	Yes	Yes	Yes	Yes
Observations	11,341	11,341	2,794	1,820
No. of maintainers	150	150	150	150

Notes: Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1. At the endpoints, D_{-12} (D_{18}) takes a value of 1 if t is 12 (18) or more months before (after) the implementation of the IBB and 0 otherwise. Robust standard errors clustered by maintainers are in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

Table D3: Robustness checks

	SUTVA	Third-party libraries as the control group			
	(1)	(2)	(3)	(4)	(5)
VARIABLES	Bug reports (IBB on Java maintainers)	No. of bug reports	No. of enhancement reports	Length of bug reports	Length of enhancement reports
Python × IBB		**-0.121*** **	**-0.114** **	**0.260**	**0.094**
		(0.043)	**(0.044)**	**(0.171)**	**(0.175)**
Java×IBB	**0.010**				
	(0.066)				
Controls	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes
Observations	6,244	6,245	6,245	721	719
No. of maintainers	272	96	96	96	96

Notes: In Column (1), we used all Java maintainers for estimation. Robust standard errors clustered by maintainers are in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

Table D4: Robustness checks

VARIABLES	Entropy balancing				Synthetic DID	
	(1) Bug reports	(2) Enhancement reports	(3) Bug report length	(4) Enhancement report length	(5) Bug reports	(6) Enhancement reports
Python × IBB	**-0.214*** (0.050)**	**-0.091** (0.042)**	**0.021 (0.121)**	**-0.042 (0.082)**	**-0.224** (0.102)**	**-0.143*** (0.056)**
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	26,915	26,915	9,441	5,229	13,692	13,692
No. of Maintainers	364	364	364	364	163	163

Notes: In Columns (1)-(4), the entropy balancing strategy allowed us to retain all maintainers. In Column (5) and (6), we use STATA command "SDID" for estimation. We include all maintainers who were active through from Jan 2012 and Dec 2018 for estimation but cannot obtain estimations for quality dependent variables, as the synthetic DID estimation requires a balanced sample without missing values (Pailañir and Clarke 2023). Robust standard errors clustered by maintainers in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$.

Table D5: Robustness checks (cont.)

VARIABLES	Additional matching covariates				Balanced panel			
	(1) Bug reports	(2) Enhancement reports	(3) Bug report length	(4) Enhancement report length	(5) Bug reports	(6) Enhancement reports	(7) Bug report length	(8) Enhancement report length
Python × IBB	**-0.225*** (0.057)**	**-0.106** (0.042)**	**0.156 (0.126)**	**-0.135 (0.132)**	**-0.283*** (0.076)**	**-0.185*** (0.044)**	**0.174 (0.142)**	**-0.052 (0.094)**
Controls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Year and Month FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Observations	6,726	6,726	1,418	817	6,552	6,552	1,618	943
No. of Maintainers	88	88	88	88	78	78	78	78

Notes: In Column (1) - (4), we use additional covariates for the CEM matching, including pretreatment bug and enhancement report numbers, report length, valid ratios, patch ratios, patch lengths, report experience, commit number, and commit size as matching covariates. In Columns (5) - (8), we consider only maintainers who joined before January 2012 and contributed reports after December 2018, and are therefore considered as "active" throughout January 2012 to December 2018 to create a balanced panel. We also follow the same CEM matching approach as our main sample. Robust standard errors clustered by maintainers in parentheses. *** $p < 0.01$, ** $p < 0.05$, * $p < 0.1$.

Table D6: Robustness checks (cont.)

VARIABLES	Full sample				Appending 12-months zero contributions at the end		Winsorized dependent variables			
	(1) Bug reports	(2) Enhancement reports	(3) Bug report length	(4) Enhancement report length	(5) Bug reports	(6) Enhancement reports	(7) Bug reports	(8) Enhancement reports	(9) Bug report length	(10) Enhancement report length
Python × IBB	**-0.094****	**-0.105*****	**0.125***	**-0.028**	**-0.230*****	**-0.123*****	**-0.255*****	**-0.141*****	**0.076**	**-0.128**
	(0.046)	**(0.030)**	**(0.071)**	**(0.082)**	**(0.048)**	**(0.037)**	**(0.048)**	**(0.035)**	**(0.110)**	**(0.111)**
Controls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Year and month FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Observations	26,915	26,915	9,441	5,229	12,188	12,188	11,341	11,341	2,794s	1,820
No. of Maintainers	364	364	364	364	150	150	150	150	150	150

Notes: Column (1)-(4) represent the estimation results without matching. Column (5)-(6) show that our main results are robust by appending additional 12 zero contribution periods after maintainers' last observed contributions. In Column (7)-(10), we winsorize the dependent variables at 95 percentiles of its distributions. Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

Table D7: Robustness checks (cont.)

VARIABLES	Exclude 10% mostly affected maintainers				Untransformed contribution numbers		Poisson model	
	(1) Bug reports	(2) Enhancement reports	(3) Bug report length	(4) Enhancement report length	(5) Bug reports	(6) Enhancement reports	(7) Bug reports	(8) Enhancement reports
Python × IBB	**-0.269*****	**-0.147*****	**0.109**	**-0.123**	**-0.626*****	**-0.270*****	**-1.012*****	**-0.802*****
	(0.053)	**(0.039)**	**(0.113)**	**(0.114)**	**(0.124)**	**(0.097)**	**(0.215)**	**(0.230)**
Controls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Maintainer FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Year and Month FE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Observations	10,811	10,811	2,760	1,791	11,341	11,341	11,341	11,341
No. of Maintainers	142	142	142	142	150	150	150	150

Notes: In Column (1) -(4), we exclude the top 10 percent of Python maintainers (eight maintainers) most affected by the IBB from the estimations. These most affected individuals are those with the greatest average reduction in bug reporting after the implementation of the IBB. Columns (5)-(8) show that we obtain consistent findings by not transforming the report contribution numbers (which may contain zero contribution) or using Possion models, thus addressing the concern that our results are driven by transformation of zero contributions. Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

Table D8: Robustness checks (cont.)

VARIABLES	Patch merge success rate		Placebo treatment (six-month before IBB)		Placebo treatment (one year before IBB)		Report misclassification
	(1) Rate for bug reports	(2) Rate for enhancement reports	(3) Bug reports	(4) Enhancement reports	(5) Bug reports	(6) Enhancement reports	(7) Pruned enhancement reports
Python × IBB	**-0.045**	**-0.003**	**-0.074**	**-0.043**	**-0.099**	**-0.071**	**-0.095*** **
	(0.051)	**(0.064)**	**(0.060)**	**(0.050)**	**(0.063)**	**(0.048)**	**(0.025)**
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	2,794	1,820	3,105	3,105	3,105	3,105	11,341
No. of Maintainers	150	150	150	150	150	150	150

Notes: In Column (1) and (2), patch merge success rate measures the proportion of maintainers' report patches that are successfully merged (typically the associated report would be marked as "fixed" or "accepted" in the resolution field). Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

Table D9: Sensitivity tests

VARIABLE	(1) Bug reports (Google Index for Oracle JDK)	(2) Enhancement reports (Google Index for Oracle JDK)	(3) Bug reports (before Oracle JDK license change)	(4) Enhancement report (before Oracle JDK license change)	(5) Bug reports (Google Index+license control)	(6) Enhancement reports (Google Index+license control)
Python × IBB	**-0.256*** **	**-0.145*** **	**-0.274*** **	**-0.141*** **	**-0.254*** **	**-0.136*** **
	(0.051)	**(0.037)**	**(0.053)**	**(0.038)**	**(0.053)**	**(0.037)**
Google Index for Commercial version	-0.002***	-0.000			-0.002***	-0.000
	(0.001)	(0.001)			(0.001)	(0.001)
License					0.023	0.097*
					(0.082)	(0.051)
Other 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	11,341	11,341	10,646	10,646	11,341	11,341
Number of maintainers	150	150	150	150	150	150

Notes: Report length results are omitted for brevity; consistent with our main findings, there is no significant influence. Robust standard errors clustered by maintainers in parentheses. *** p < 0.01, ** p < 0.05, * p < 0.1.

References

Andrews, I., Stock, J. H., & Sun, L. (2019). Weak instruments in instrumental variables regression: Theory and practice. Annual Review of Economics, 11, 727-753.

Argote, L., & Miron-Spektor, E. (2011). Organizational learning: From experience to knowledge. Organization science, 22(5), 1123-1137.

Banker, R. D., Datar, S. M., & Kemerer, C. F. (1991). A model to evaluate variables impacting the productivity of software maintenance projects. Management Science, 37(1), 1-18.

Bertrand, M., Duflo, E., & Mullainathan, S. (2004). How much should we trust differences-in-differences estimates?. The Quarterly journal of economics, 119(1), 249-275.
Burtch, G., Hong, Y., Bapna, R., & Griskevicius, V. (2018). Stimulating online reviews by combining financial incentives and social norms. Management Science, 64(5), 2065-2082.
Brynjolfsson E, Milgrom P (2013) Complementarity in organizations. The handbook of organizational economics:11–55.
Chen, W., Jin, F., & Xue, L. (2022). Flourish or perish? The impact of technological acquisitions on contributions to open-source software. Information Systems Research, 33(3), 867–886.
Cheng Z, Pang MS, Pavlou PA (2020) Mitigating traffic congestion: The role of intelligent transportation systems. Information Systems Research, 31(3):653–674.
Conley, T. G., Hansen, C. B., & Rossi, P. E. (2012). Plausibly exogenous. Review of Economics and Statistics, 94(1), 260-272.
Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., ... & Zhou, M. (2020). Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155.
Frake, J., & Harmon, D. (2024). Intergenerational transmission of organizational misconduct: Evidence from the Chicago Police Department. Management Science, 70(6), 3856-3878.
Goes, P. B., Guo, C., & Lin, M. (2016). Do incentive hierarchies induce user effort? Evidence from an online knowledge exchange. Information Systems Research, 27(3), 497-516.
Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., ... & Zhou, M. (2020). Graphcodebert: Pre-training code representations with data flow. arXiv preprint arXiv:2009.08366.
Hong, G., & Raudenbush, S. W. (2006). Evaluating kindergarten retention policy: A case study of causal inference for multilevel observational data. Journal of the American Statistical Association, 101(475), 901-910.
Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology, 33(8), 1-79.
IBM. (n.d.). Halstead effort. IBM Documentation. https://www.ibm.com/docs/en/raa/6.1.0?topic=metrics-halstead-effort (Accessed December 2024)
Jo, W., Sunder, S., Choi, J., & Trivedi, M. (2020). Protecting consumers from themselves: Assessing consequences of usage restriction laws on online game usage and spending. Marketing Science, 39(1), 117-133.
Kessler J (2017) Scattertext: a browser-based tool for visualizing how corpora differ. Proc. ACL 2017 Syst. (Association for Computational Linguistics, Vancouver, Canada), 85–90.
Law, K. K., & Zuo, L. (2021). How does the economy shape the financial advisory profession?. Management Science, 67(4), 2466-2482.
Lerner, J., & Tirole, J. (2002). Some simple economics of open source. The journal of industrial economics, 50(2), 197-234.
Walker J (2021) PHP removed from Internet Bug Bounty program – but scripting language custodians were 'never involved' from the outset. Retrieved February 21, https://portswigger.net/daily-swig/php-removed-from-internet-bug-bounty-program-but-scripting-language-custodians-were-never-involved-from-the-outset.
Pamuru V, Khern-am-nuai W, Kannan K (2021) The impact of an augmented-reality game on local businesses: A study of Pokémon go on restaurants. Information Systems Research, 32(3):950–966.
Yu, Y., Khern-am-nuai, W., & Pinsonneault, A. (2022). When Paying for Reviews Pays Off: The Case of Performance-Contingent Monetary Rewards. MIS Quarterly, 46(1).