# Towards Automated Detection of Unethical Behavior in Open-Source Software Projects

Hsu Myat Win
Southern University of Science and Technology
China
11960003@mail.sustech.edu.cn

Haibo Wang
Southern University of Science and Technology
China
wanghb2020@mail.sustech.edu.cn

Shin Hwei Tan*
Concordia University
Canada
shinhwei.tan@concordia.ca

## ABSTRACT

Given the rapid growth of Open-Source Software (OSS) projects, ethical considerations are becoming more important. Past studies focused on specific ethical issues (e.g., gender bias and fairness in OSS). There is little to no study on the different types of unethical behavior in OSS projects. We present the first study of unethical behavior in OSS projects from the stakeholders' perspective. Our study of 316 GitHub issues provides a taxonomy of 15 types of unethical behavior guided by six ethical principles (e.g., autonomy). Examples of new unethical behavior include *soft forking* (copying a repository without forking) and *self-promotion* (promoting a repository without self-identifying as contributor to the repository). We also identify 18 types of software artifacts affected by the unethical behavior. The diverse types of unethical behavior identified in our study (1) call for attentions of developers and researchers when making contributions in GitHub, and (2) point to future research on automated detection of unethical behavior in OSS projects. From our study, we propose Etor, an approach that can automatically detect six types of unethical behavior by using ontological engineering and Semantic Web Rule Language (SWRL) rules to model GitHub attributes and software artifacts. Our evaluation on 195,621 GitHub issues (1,765 GitHub repositories) shows that Etor can automatically detect 548 unethical behavior with 74.8% average true positive rate (up to 100% true positive rate). This shows the feasibility of automated detection of unethical behavior in OSS projects.

## CCS CONCEPTS

• **Social and professional topics** → **Codes of ethics**; • **Software and its engineering** → **Software maintenance tools**; **Open source model**.

## KEYWORDS

Ethics in Software Engineering, Open-source software projects

*corresponding author


*ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA*

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3616314

**ACM Reference Format:**
Hsu Myat Win, Haibo Wang, and Shin Hwei Tan. 2023. Towards Automated Detection of Unethical Behavior in Open-Source Software Projects. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616314

## 1 INTRODUCTION

With the increasing popularity of Open-Source Software (OSS) development, ethical considerations have become an important yet often neglected topic within the research community. For example, the incident where researchers investigated the feasibility of stealthily introducing vulnerabilities in OSS by making *hypocrite commits* (commits that deliberately introduce critical bugs into code), has provoked active discussion among the Linux community, researchers, and other OSS developers [66]. The Linux developers argued that making "hypocrite commits" is "not ethical", and wastes developers' time in reviewing invalid patches [20]. More importantly, this incident [66] has revealed an attack on the basic premise of OSS itself (i.e., the fact that anyone can contribute to the code and any OSS project is susceptible to a similar incident). Indeed, unethical behavior committed by OSS contributors might lead to broken trust between the OSS community and the contributor, whereas unethical software development might lead to loss of funding, reputation, or other resources of the OSS organization involved. Despite the importance of understanding the unethical issues by *stakeholders* (individuals who participated or are interested in the OSS project, and can either affect or be affected by the OSS project), most studies on unethical behavior in OSS projects mainly focus on the common types of unethical behavior, such as gender bias [42, 62], fairness in the code review process [36], and software licensing [47, 64, 65]. *There is little to no study that investigates the important question: "What kind of behavior is considered unethical by stakeholders in OSS projects?".* Without understanding the definition of unethical behavior from the perspective of the stakeholders of OSS projects, incidents similar to the "hypocrite commits" experiments are bound to reoccur.

Prior studies stress the importance of considering ethical issues in OSS projects by using various examples and referring to ethical principles [37, 58**?** ]. Unfortunately, a study revealed that instructing participants to consider the ACM code of ethics does not affect their ethical decision-making in software engineering tasks [51]. A similar argument has been made in AI ethics, which calls for practical methods to translate principles into practice [55]. Hence, we argue that it is *not enough to merely observe the occurrence of unethical behavior via several examples in OSS projects*, it is much

more important to *systematically study their characteristics, and design practical tools that can automatically detect unethical behavior by presenting evidences using data in OSS projects to stakeholders.*

To bridge the gaps between general ethical principles and OSS practices, we present the first study of the types of unethical behavior in OSS projects from stakeholders' perspectives. Note that we conduct a cross-domain study because it allows us to capture the dynamics of collaboration and community interactions, which are essential considerations in ethical evaluations. Specifically, our study aims to answer the research questions below:

**(RQ1) How do stakeholders in OSS projects define unethical behavior, and what are the types of unethical behavior?**

To understand the definition of unethical behavior from the perspective of the stakeholders of OSS projects, by referring to ethical principles, we study the diverse types of unethical behavior, their characteristics, and the corresponding ethical principles that drive these unethical behavior in OSS projects.

**(RQ2) Given the type of unethical behavior, what are the OSS project artifacts that are affected by the unethical behavior?**

To guide our design of a tool that can automatically identify unethical behavior in OSS projects, we study the affected software artifacts (i.e., artifacts in which the stakeholders claimed to violate ethical principles) for each type of unethical behavior.

Our study leads to a taxonomy of 15 types of unethical behavior in OSS projects, including S1: No attribution to the author in code, S2: Soft forking, S3: Plagiarism, S4: License incompatibility, S5: No license provided in public repository, S6: Uninformed license change, S7: Depending on proprietary software, S8: Self-promotion, S9: Unmaintained project with paid service, S10: Vulnerable code/API, S11: Naming confusion, S12: Closing issue/PR without explanation, S13: Offensive language, S14: No opt-in or no option allowed, and S15: Privacy Violation. Six of them have not been studied (S2, S6, S8, S9, S11, S12). Inspired by our study, we propose Ethic detector (Etor), an automatic detection tool based on *ontological engineering* (a description of entities and their properties, relationships, and behaviors) and Semantic Web Rule Language (SWRL) rules to model software artifacts. In short, we made the following contributions:

- **Study.** To the best of our knowledge, we conducted the first study of unethical behavior in OSS projects from the stakeholders' perspective. Our study of 316 GitHub issues/PRs from 301 projects revealed 15 types of unethical behavior with six new ones. Our study also revealed the diversity of the affected software artifacts. Our benchmark containing 316 issues with various types of unethical behavior lays the foundation for future automated approaches for detecting unethical behavior.
- **Technique.** We propose Etor, a novel ontology-based tool that automatically detects unethical behavior in OSS projects. We model GitHub attributes using ontologies, and design SWRL rules to check for unethical behavior in various artifacts.
- **Evaluation.** Our evaluation on 195,621 GitHub issues/ PRs from 1,765 repos shows that Etor can automatically detect 548 issues with 74.8% true positive rate on average.

## 2 BACKGROUND AND RELATED WORK

***Ethical Principles.*** Prior work on ethical principles in OSS projects mainly studied six aspects: (1) accountability, (2) attribution, (3) autonomy, (4) informed consent, (5) privacy, and (6) trust [29, 34, 46, 63]. **Accountability** means that an individual is accountable for their actions. **Attribution** (e.g., copyright) means giving credit to authors when the credit is due. **Autonomy** allows an individual to decide, plan, and act to achieve their goals. In OSS projects, individuals inherently have autonomy because they can choose which tasks to perform but may gain or lose autonomy once they agree to participate. **Informed Consent** is an agreement between the individual and the institution maintaining ethical values, such as autonomy. **Privacy** is a right of a stakeholder on what information another stakeholder can obtain and communicate to others. **Trust** refers to expectations between people through goodwill.

***Web Ontology Language (OWL)*** is a standard ontology language endorsed by the W3C to construct an OWL knowledge model [1, 22, 49]. It is a semantic web language designed to model rich and complex knowledge about things, groups of things, and relations between things. Knowledge expressed in OWL can be exploited by computer programs, e.g., to verify the consistency of that knowledge or to make implicit knowledge explicit. Thus, we design our tool based on ontology engineering.

***Semantic Web Rule Language (SWRL)*** is a language that combines OWL and Rule Markup Language (RuleML), which can be used to express Horn-like rules and logic [2]. SWRL rules are used to infer new knowledge regarding the individual (instance) by chains of properties. We choose to model the unethical behavior in OSS projects using SWRL because (1) its expressiveness [39] is well-suited for modeling unethical behavior that involves different GitHub attributes and diverse types of software artifacts, and (2) it has been widely used to model concepts such as privacy for medical data [28] and access control policy [26, 45].

***Related Work.*** Prior work studies focus on multiple aspects of ethical concerns for several domains.

*Ethical concerns in Software Engineering research.* Several studies focus on ethical concerns for empirical studies in software engineering. Badampudi conducted a study about the reports of the ethical considerations in Software Engineering publications [23]. Andrews et al. [21] illustrated some of the common approaches to encourage ethical behavior and their limits for demanding ethical behavior between researchers' duty and their publishing as well as the companies' and individuals' integrity. Singer et al. [60] introduced their work as a practical guide to ethical research involving humans in software engineering. Our study is complementary to these studies as the types of unethical behavior discovered in our study point to potential violations of ethical principles that software engineering researchers should consider when assessing automated tools involving OSS projects.

*Studies on ethical concerns in OSS.* Mikkonen et al. [53] highlight the persistence of Protestant ethic values in Free OSS (FOSS) communities. Yet, their emphasis is on FOSS motivation, with less attention to reducing unethical behavior acts like license violations. On the other hand, some existing studies of OSS projects focus on issues related to gender bias [42, 62], fairness of the code review process [36], similar code in Stack Overflow and GitHub [24, 68], and software licensing [47] [65] [64]. Studies relating to gender bias in GitHub [42, 62] aim to address the obstacles in improving gender diversity. Meanwhile, a study of a large industrial open source ecosystem (OpenStack) shows that unfairness is "starting

to be perceived as an issue" in OSS [36]. Several studies investigated code clones between code snippets from Stack Overflow and projects on GitHub and found a considerable number of non-trivial clones [24, 68]. Although these studies also explored how GitHub stakeholder's reference code was copied or adapted from Stack Overflow answers without giving proper credit to the authors (who wrote the code), they did not consider the scenario where the stakeholder of the code snippets used in GitHub is the same as the owner of the code in Stack Overflow (in this case, credit is not needed). Several techniques have been proposed for the automated detection of license incompatibility [35, 43, 67]. While our study identifies license incompatibility as an unethical behavior, it includes more diverse types of issues related to licensing (e.g., missing license, and uninformed license change). Nevertheless, all existing studies on ethical concerns in OSS projects only focus on a few aspects of ethical principles, and they did not conduct analysis of the diverse types of ethical violations in OSS projects in GitHub.

## 3 STUDY OF UNETHICAL BEHAVIOR IN OSS

To address the two research questions introduced in Section 1, we conducted a study of unethical behavior in OSS projects. Figure 1 gives an overview of our study.

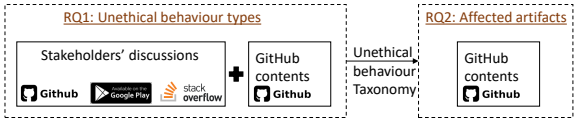


**Figure 1: Overview of our study on unethical behavior.**

### 3.1 Study Methodology

*3.1.1 Data Collection.* We built a crawler that searches GitHub issues using the keyword "ethic", concepts related to unethics, and synonyms for "un/ethical" (i.e., "unprofessional", "unfair", "right", "proper", and "principle") via the GitHub API. Our crawler identifies 1235 issues/PRs of 842 projects submitted by stakeholders. Due to GitHub API's 1000-results restriction, we used PyGitHub [3] and github-api [4] to perform targeted searches using keywords along with issue creation year (i.e., results for each keyword-year combination stayed below 1000 records since 2008). We then manually checked the results to exclude issues that do not have a clear description or are unrelated to ethical behavior (e.g., issues that mentioned "ethic"' but only updated terms and conditions[1]). After manually filtering 919 irrelevant issues, we obtained 316 issues.

*3.1.2 Deriving Initial Keywords Based on Ethical Principles.* Before analyzing the GitHub issues, we reviewed ethical principles from prior studies [29, 34, 46, 63], and identified six ethical principles guiding the action of stakeholders in OSS projects, including: (1) accountability, (2) attribution, (3) autonomy, (4) informed consent, (5) privacy, and (6) trust (i.e., we exclude "welfare" because it is related to fair wages which is generally not discussed in our studied issues). We use these six underlying ethical principles and their corresponding ethical guidelines as guidance for merging relevant themes. We first refer to the six principles and their corresponding guidelines. For example, keywords such as "copy" and "plagiarism" belong to the same ethical guideline ("To respect copyright") for "(S1) No attribution to the author in code", "(S2) Soft forking", and "(S3) Plagiarism" as they are all related to giving proper credits to the authors but we separate them into different types as they involve different degrees of copying (copying entire repository in "Soft forking" versus copying texts in "Plagiarism"). Based on the ethical principles and guidelines, we obtain the initial 23 keywords (e.g., "copy", and "plagiarism") shown in Table 1.

**Table 1: Ethical guidelines and keywords**

| Ethical guideline | Keywords |
|---|---|
| To respect copyright | copy, plagiarism, credit, fork |
| To help individuals make informed consent decisions easier via licensing | license |
| To avoid license violation | proprietary, closed source |
| To respect expectations between people through goodwill | promote, advertise, promotion |
| To be responsible for the project maintenance | maintain, support, pay, purchase |
| To avoid fraudulent activities | malware, vulnerable, hacking |
| To be responsible for naming | name |
| To be responsible for explaining public actions | unfair |
| To avoid offensive language | offensive |
| To allow individuals to choose which tasks to perform | opt-in, option |
| To protect the right of an individual of personal information | privacy |

*3.1.3 Data Analysis.* After getting the initial keywords, we manually analyzed the discussions using thematic analysis [31], an approach for identifying patterns (or "themes") within data. Specifically, the first two authors of the paper perform five steps: (1) We carefully read and analyzed all discussions in the issue to understand what stakeholders discussed about and how they described unethical behavior; (2) We coded the key sentences and phrases in each issue by highlighting sections of text, and coming up with shorthand labels or "codes" (e.g., licenses) to describe their content. We reread the related key sentences, and their surrounding context discussions to generate initial codes. New codes can be added when reading the discussions. After reading the discussions, we collate together the key phrases into groups identified by codes. These codes allow us to have a condensed overview of the main points and common recurring phrases. (3) From initial codes, we reviewed and aggregated codes with similar meanings into groups (e.g., importance of informed consent) to derive themes. Themes are generally broader than codes. (4) With the initial set of themes in the prior step, we reviewed all themes to merge similar themes (e.g., promoting informed consent through licensing and mitigating license violations); (5) We finalized the themes by providing clear definitions. To reduce research bias, steps (1) to (4) were conducted independently by the first two authors of the paper. Both authors are Ph.D. students with more than two years of research experience and over 4 years of coding experience. The first author had taken a computer ethics course, while the second author had experience in OSS development. After independent analysis, the two authors meet to resolve conflicts and finalize themes in step (5). Specifically,

[1] https://github.com/Pryaxis/handbook/issues/3

there are 39 cases (12%) with divergent themes for types of unethical behavior. Subsequently, our 6-month study resulted in identifying 15 types of unethical behavior with 11 ethical guidelines.

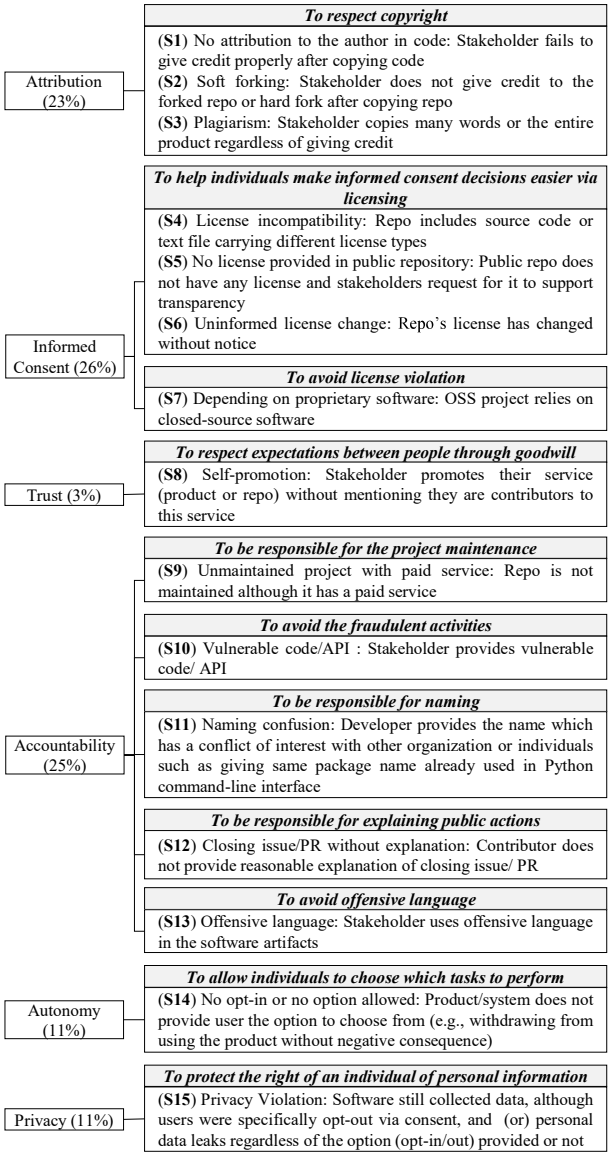


**Figure 2: Taxonomy of unethical behavior in OSS projects.**

## 3.2 RQ1: Types of Unethical Behavior

Figure 2 shows the 15 types of unethical behavior in our study. Boxes on the left (e.g., "Attribution") describes the ethical principle behind each type, whereas the grey heading for the boxes on the right (e.g., "To respect copyright") includes the 11 ethical guidelines, and the contents present the related types of unethical behavior. Six of the 15 types have not been previously studied (i.e., S2, S6, S8, S9, S11, S12).

**Table 2: Types and affected artifacts for unethical behavior**

| Type | Issues (#) | Affected Software Artifacts |
|---|---|---|
| S1 | 49 | 41 Source code, 5 Configuration files, 1 API, 1 project, 1 script |
| S2 | 19 | 19 Projects |
| S3 | 6 | 2 Source code, 2 Data, 1 UI, 1 Project |
| S4 | 26 | 9 Legalese, 7 Source code, 4 `README`/ `CONTRIBUTING.md`, 3 Configuration files, 1 Image, 1 OS, 1 Website |
| S5 | 31 | 31 Legalese |
| S6 | 9 | 9 `CHANGELOGs` |
| S7 | 16 | 16 APIs |
| S8 | 8 | 8 PR/Issue comments |
| S9 | 10 | 10 Release histories |
| S10 | 27 | 23 Source code, 4 APIs |
| S11 | 21 | 10 Product names, 8 Source code, 1 UI, 1 Data, 1 Script |
| S12 | 15 | 10 PR/Issue comments, 5 PR/Issue code reviews |
| S13 | 7 | 2 UIs, 2 Product names, 1 Source code, 1 `README`/ `CONTRIBUTING.md`, 1 Website |
| S14 | 36 | 15 UIs, 11 Software features, 6 Source code, 4 Configuration files |
| S15 | 36 | 12 Source code, 10 APIs, 5 UIs, 5 Software features, 3 Configuration files, 1 Website |
| **Total** | **316** | **316** |

**Finding 1:** The types of unethical behavior in OSS projects are diverse (15 types identified in our study, with six new ones)

We explain the 11 ethical guidelines and the corresponding types of unethical behavior below:

(1) *To respect copyright.* There are three types of unethical behavior related to copyright, described below:

**S1: No attribution to the author in code.** This issue occurs when the stakeholders fail to give proper credit after copying code [25]. An example for **S1** is: "it is **unethical** not to credit or at the least, point out that these features are inspired by...".[2]

**S2: Soft forking.** This issue occurs if the copied item is a *repository* and the copied repository has not been forked. Although GitHub encourages forking for social coding, a copied repository should acknowledge the original repository by creating an official fork [52]. An example discussion for **S2** is: "Unauthorised copy of... **unethical**... You must delete this repo and fork from the original...".[3]

**S3: Plagiarism.** Plagiarism occurs if the stakeholders copy texts (non-source code) or the entire product regardless of giving credit or not [19]. An example discussion for **S3** is: "Interactive book should be free of plagiarism. By replicating the content used by...**unethical**.".[4]

In this example, the repository of an interactive book is unethical because the book uses copied texts from several websites.

(2) *To help individuals make informed consent decisions easier via licensing.* There are three types of unethical behavior related to licensing, described below:

**S4: License incompatibility.** It occurs if the repository includes source code or text files carrying different license types than the project's license because stakeholders must ensure license compatibility of the repository. Example for **S4** is: "To continue distributing when we know they have incompatible licenses is **unethical**.".[5]

**S5: No license provided in public repository.** This issue occurs if the public repository does not have any license and the

[2] https://github.com/novus-package-manager/novus/issues/3
[3] https://github.com/biddyweb/yes-cart/issues/33
[4] https://github.com/CircuitVerse/Interactive-Book/issues/80
[5] https://github.com/mpdf/mpdf/issues/15

stakeholders request for it because licenses state the official permissions to use a repository, and project owners should provide them if the OSS is public for greater transparency. An example comment for **S5** is: "The repository is public which implies an intent of being open-source but no license is specified making review of the code an issue...People get...at the end of the day, but they are funding this stuff instead of the... developers. That's **unethical** but legal.".[6]

**S6: Uninformed license change.** Due to transparency concerns, OSS developers should inform the stakeholders about the license change (via `CHANGELOG` or PR) prior to changing the license. **S6** occurs if the contributors fail to do so. An example for **S6** is: "Normally license change are announced in some form of PR or announcement or discussion and none of that has taken place...I find this silent change **unethical**.".[7]

(3) *To avoid license violations.* Stakeholders must obey the OSS license agreement and avoid integrating prohibited licenses that cause violations in license dependency chains [44].

**S7: Depending on proprietary software.** This issue occurs if the OSS project relies on closed-source software because OSS projects should be fully open-sourced. An example comment for **S7** is: "Since ... is fully open source software, I believe depending on closed source software is **unethical**".[8]

(4) *To respect expectations between people through goodwill.* Trust is an ethical principle that refers to respecting expectations between people through goodwill. The following type of unethical behavior may lead to broken trust among stakeholders in OSS projects:

**S8: Self-promotion.** This issue occurs when the stakeholder advertises their repository by suggesting to incorporate it into another repository *without mentioning that they are contributors or owners of the artifact.* This goal of the stakeholder is to attract attention to their less well-known repository to increase its popularity. An example comment for **S8** is: "I strongly advise against migrating to lib2...Seeing him leverage his notability and following to promote and increase the adoption of lib2 ..., which he just released a few days ago, is **unethical**...failing to disclose that you are promoting your own package here is a bad".[9]

In this example, an external (not affiliated with the ESLint repository) user who is the owner of the `lib2` library opens a PR in the ESlint repository to suggest replacing the `chalk` library with `lib2` to promote their own library. To mitigate this issue, a contributor of ESLint later suggested the user to disclose the fact that they are promoting their own library.

(5) *To be responsible for the project maintenance.* Project owners, especially those who offer paid services should actively maintain their projects. If the project owners would to discontinue their technical support, they should inform the users before asking them to pay for the service.

**S9: Unmaintained project with paid service.** This issue occurs if the project repository is not actively maintained when it has a paid service. It is unethical because the project owner is responsible for providing support to paid users who reported the bugs, and fixing the bugs within a reasonable time. An example for **S9** is: "I just bought the pro version, and now I'm having this same problem...definitely **unethical**.".[10]

In this example, the user who has paid for the open-source app reported the failure in using themes (a functionality that is only available for paid users) but the app is no longer maintained.

(6) *To avoid fraudulent activities.* As a code of conduct in OSS, stakeholders should be aware of malicious activities.

**S10: Vulnerable code/API.** The issue occurs when stakeholders or a project is involved in malicious activities (e.g., contributing malicious code/API or leaving an unfixed vulnerability in the code). An example comment for **S10** is: "Given that iText 2.1.7 has...unfixed security vulnerability, ...continuing to release it is **unethical**. In my opinion, iText 2.1.7 should be replaced by OpenPDF.".[11]

In this example, the user suggested replacing `iText` which has unfixed vulnerability with another library (`OpenPDF`) where the vulnerability has been fixed. Another example for **S10** is the "hypocrite commits" incident mentioned in Section 1.

(7) *To be responsible for naming.* Stakeholders are responsible for all software artifacts that they own, including the selected names.

**S11: Naming confusion.** This issue occurs when it involves the stakeholders' duty to give unique names for their artifacts (e.g., packages, variables, and libraries). Project owners should identify unique names before using the names. An example for **S11** is: "There is already a package 'click' for creating command-line interfaces. I am using `coreapi` package which ... import `click` package:... your library does not have a style component and python throws an error...this kind of behavior for a company... **unethical**".[12]

In the above example, a user complained that the developers of the `click-integration-django` library select the same package name as the `Click` package, causing an error when using the package due to naming conflicts.

(8) *To be responsible for explaining public actions.* Owners of OSS projects should explain each decision made for supporting users.

**S12: Closing issue/PR without explanation.** This occurs when an issue/PR has been closed without providing any explanation because all stakeholders are expected to receive reasonable explanations for informational fairness [30]. An example for **S12** is: "It's a bit **unfair** to just close something without explaining why?...I don't understand why this (despite several closed issues all saying the same thing) isn't being implemented".[13]

(9) *To avoid offensive language.* Stakeholders should encourage respectful environment in OSS projects by avoiding offensive language because words with offensive language might represent unethical behavior [32]. Prior study stated that hate speech (offensive words) might not be a criminal offense but can still be harmful [56].

**S13: Offensive language.** This occurs if the stakeholders or part of the project uses offensive language, including but not limited to insults, arrogance, entitlement, unprofessional, and similar forms of offensive communication [54]. An example for **S13** is: "Rename the Scroll of Genocide to something else...It was never a good or **ethical** name...It is not "merely" systemic and deliberate mass-murder...but state-enacted systemic destruction, neglect and

[6] https://github.com/pkalogiros/AudioMass/issues/1
[7] https://github.com/minio/minio/issues/12143
[8] https://github.com/wger-project/wger/issues/266
[9] https://github.com/eslint/eslint/pull/15102
[10] https://github.com/tranleduy2000/javaide/issues/236
[11] https://github.com/flyingsaucerproject/flyingsaucer/pull/123
[12] https://github.com/click-llc/click-integration-django/issues/1
[13] https://github.com/twbs/bootstrap/issues/5632

suppression of entire schools of culture, science, literature, truth, of everything that makes us human".[14]

In this example, the stakeholder thinks that using the word "Genocide" to name a scroll in the game is unethical because the word promotes intentional destruction of human beings.

(10) *To allow individuals to choose which tasks to perform.* Based on the "Autonomy" ethical principle, stakeholders of OSS should have the freedom to choose the tasks to perform.

**S14: No opt-in or no option allowed.** This occurs if the system does not provide options (e.g., withdrawing from using the product). For example, no option is available for uninstalling the third-party library. We focus on issues with "no option" or "no opt-in" because they provide stronger protections than opt-out [27]. An example comment for **S14** is: "There should be an option if someone wants to completely remove ... from the system...I think it's **unethical** to not provide an easy way for a program to be uninstalled".[15]

(11) *To protect the right of an individual of personal information.* The privacy of stakeholders of OSS should be protected.

**S15: Privacy Violation.** This occurs in OSS projects under two common scenarios: (1) if the software still collects data despite opting-out via consent, and (2) if there exist personal data leaks regardless of the options (opt-in/out). Example for **S15** is: "Form submitted even if opt-in checkbox is unchecked...Signing people up when they haven't opted in is a major enough bug that it renders the plugin useless (or at least **unethical**)".[16]

Table 2 presents the number of issues we found for each type of unethical behavior. The "Type" and "Issues (#)" columns represent the types of unethical behavior and the number of issues we found in GitHub, respectively. Overall, our study identifies 15 types of unethical behavior where *the most common types of unethical behavior are related to copyright (**S1**, **S2**, and **S3**) and licensing (**S4**, **S5**, **S6**, and **S7**).* As our study shows that illegal copying of code (**S1**) or copying the entire repository (**S2**), or copying texts (**S3**) are common in OSS projects, we hope to raise awareness to stakeholders of OSS projects that such behavior is considered unethical.

**Finding 2:** The most common types of unethical behavior in OSS are issues related to copyright (23%) and licensing (26%).

## 3.3 RQ2: Affected Software Artifacts

*3.3.1 Identifying Software Artifacts.* We define *affected software artifacts* as objects in software repositories that violate ethical principles. To derive the set of affected software artifacts, we started with the 19 categories from the taxonomy of prior study [59]. Then, we categorized the artifacts we found in our study based on the 19 categories. After removing 11 categories with no artifact found, we obtained eight categories: (1) source code, (2) script, (3) configuration, (4) database (data), (5) image, (6) prose, (7) legalese, and (8) other. For the *prose* category (i.e., plain text files), we only found two concrete types (i.e., README/CONTRIBUTING.md, and CHANGELOG) so we separated them into two categories. As the category "other" in prior study [59] is too broad, we split it into 10 new categories based on aforementioned steps in thematic analysis: (1) external application programming interface (API), (2) user interface (UI), (3) project, (4) release history, (5) software feature, (6) product name, (7) operating system (OS), (8) website, (9) PR/Issue code review, (10) PR/Issue comment. We derive "PR/Issue code review" and "PR/Issue comment" based on prior work [41]. Our newly introduced categories aim to preserve the hierarchy of artifacts (Project > Software feature [33] > Source code). For 28 cases (8.9%), both authors meet to discuss the issues labeled with different categories to resolve any disagreement. Finally, we obtained 18 affected software artifacts: (1) project, (2) software feature, (3) source code, (4) external API, (5) legalese, (6) product name, (7) release history, (8) UI, (9) configuration file, (10) PR/Issue code review, (11) PR/Issue comment, (12) README / CONTRIBUTING.md, (13) CHANGELOG, (14) data, (15) image, (16) OS, (17) website, and (18) script (i.e., source code in languages executed by an interpreter). As several artifacts are more difficult to understand, we clarify below:

- **Project**: The affected artifacts involve more than one type of artifact within the entire repository.
- **Software feature**: Functional or non-functional system requirements [33, 40]. An example is the ability to unsubscribe a service.
- **Source code**: Source files (excluding scripts, binary code, and build code) that belong to the current repository (internal).
- **External API**: API from third party (external) library or service.
- **Legalese**: Licenses, copyright notes, or patents.
- **Product name**: The product, project, or app name.

**Finding 3:** The unethical behavior in OSS projects affects many different types of software artifacts (our study found 18 types).

The third column in Table 2 presents the affected artifacts for each unethical behavior. Each number in the column denotes the number of GitHub issues with a certain type of artifact (e.g., "19 Projects" means that there are 19 issues where **S2** is affected by projects). Theoretically, one issue might discuss multiple artifacts but we found that each issue only discusses one artifact because (1) developers prefer discussing ethical concerns for one type of artifact in one issue, and (2) some categories are hierarchical (e.g., "project" includes multiple types of artifacts). Overall, Table 2 shows that *source code is still the most common type of artifact for unethical behavior* (i.e., it affects eight types of unethical behavior).

**Finding 4:** Source code is the most common type of affected artifacts (affects eight types of unethical behavior).

# 4 TOOL DESIGN

Our study reveals the existence of diverse types of unethical behavior in OSS projects, and they usually involve diverse software artifacts. The diversity and the complexity of the rules governing the ethics-related activities in GitHub motivate the need for a modeling approach that can abstract this complexity and facilitate its automatic detection. In Section 4.1, we describe how we model unethical behavior using SWRL rules. In Section 4.2, we explain the architecture of Etor that uses SWRL rules for automatic detection.

## 4.1 Modeling via SWRL Rules

We propose using SWRL rules to represent unethical behavior in an OSS project together with the publicly available data in GitHub. SWRL rules allow us to model affected software artifacts

[14] https://github.com/NetHack/NetHack/issues/359
[15] https://github.com/EasyEngine/easyengine/issues/488
[16] https://github.com/katzwebservices/Contact-Form-7-Newsletter/issues/79

**Table 3: GitHub attributes and types for auto-detection**

| Attribute | Type | Description |
|---|---|---|
| `GHRepository`: main class | | |
| licenseFile | GHContent | repo's license file |
| readmeFile | GHContent | readme file |
| fileCount | int | # of files in repo |
| fileContent | GHContent | file's content |
| commitCountByPath | int | # of commits for specific file path |
| commitByPath | GHCommit | commit for file path |
| fork | GHRepository | fork of a repo |
| forkCount | int | # of forks of repo |
| contributor | GHUser | stakeholder taking part in GitHub repo |
| pullRequestCountByCommit | int | # of PRs which contain specific commit |
| latestRelease | GHRelease | the last release in GitHub history |
| `GHUser`: A GitHub user identified by username | | |
| user | String | GitHub username |
| `GHIssue`: A GitHub issue that describes a bug or a feature. | | |
| issueMessageBody | String | description of issue |
| issueOwner | GHUser | stakeholder who reports an issue |
| `GHCommit`: The code changes in a commit. | | |
| commitCodeChange | String | code change in commit |
| `GHContent`: The content (including source code) of a file and its location (file path). | | |
| contentCount | int | # of contents stored in file's content |
| content | String | content |
| path | String | file path |
| pathCount | int | # of file paths |
| `GHRelease`: A latest release is represented by the published date of the release | | |
| publishedDate | Date | date of release in GitHub history |

as hierarchies of classes and properties, capturing the relationships between affected software artifacts and stakeholders. Table 3 shows GitHub attributes used in our modeling. The columns under "Attribute", and "Type" explain each attribute and its type. We model each OSS project as `GHRepository`. By referring to the GitHub Repositories API [5] , we selected 11 data properties (e.g., `latestRelease` and `licenseFile`) that belong to a `GHRepository` by excluding properties that are irrelevant for unethical behavior (e.g., `avatar_url` that points to the icon for a repository). Apart from `GHRepository`, we introduce six classes to model data properties of a repository: (1) `GHUser`, (2) `GHCommit`, (3) `GHContent`, (4) `GHIssue`, (5) `GHPullRequest`), (6) `GHRelease`. While GitHub users (`GHUser`) usually play different roles in OSS projects, we only model: (1) contributors (users who are official contributors of a repository) and (2) issue owners (users who report an issue). For modeling `GHIssue`, we reuse the same convention in GitHub by modeling a PR (`GHPullRequest`) as a subclass of `GHIssue` (i.e., GitHub Issue Search API will search for issues and PRs, essentially treating a PR as a type of GitHub issue). Figure 3 shows the OWL ontology for our model where `GHRepository` is the main class, and the arrows denote the relationships between the classes. Specifically, `GHIssue` $\multimap\!\triangleright$ `GHPullRequest` represents the subclass relations, whereas other arrows denote hasA relations (e.g., `GHIssue` $\multimap$ `GHUser` means that each issue has a user who reports the issue).

## 4.2 Automatic Detection of Unethical Behavior

We designed Etor to auto-detect six types. We excluded nine types because (1) they involve artifacts (e.g., product names, software features) that are difficult to automatically isolate from other artifacts (i.e., "No opt-in or no option allowed", "Privacy Violation", "Naming confusion", and "Offensive language"), (2) they require sophisticated analysis of configuration files, API or source code (i.e., "Plagiarism", "Depending on proprietary software", and "Vulnerable code/API"), (3) their detection requires advanced natural language processing (i.e., "Closing issue/PR without explanation" as it requires automatically checking if the explanation for closing the PR/issue exists), and (4) approaches for "License incompatibility" [35, 43, 67] exist so we exclude it not to duplicate effort.

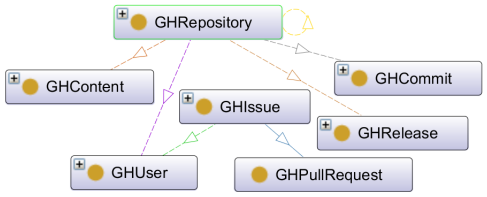


**Figure 3: Our ontology of unethical behavior in OSS projects**

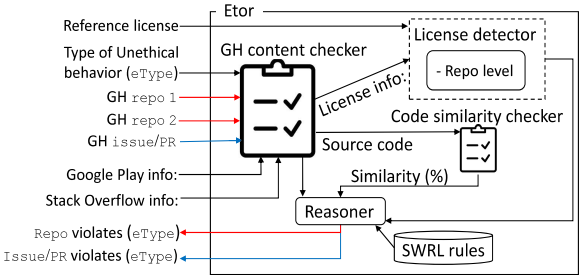


**Figure 4: Overall architecture of Etor (GH denotes GitHub).**

**Table 4: SWRL rules**

| |
|---|
| GHIssue(?i)∧isIssueMessageBody(?i,?b)∧ swrlb:contains(?b,"https://stackoverflow.com/")∧isWebLink(?b,?w)∧ isIssueOwner(?i,?u1)∧openStream(?w,?s)∧ swrlb:booleanNot(contains(?s,?u1))∧isFileCount(?r,?fc)∧ swrlb:greaterThan(?fc,0)∧isFileContent(?r,?ffct)∧ isContentCount(?ffct,?ffctc)∧swrlb:greaterThan(?ffctc,0)∧ isContent(?ffct,?ffcc)∧isSimilarCode(?ffcc,?s)∧ swrlb:booleanNot(contains(?ffcc,?w))∧→**S1**(?i) |
| GHRepository(?r1)∧GHRepository(?r2)∧swrlb:isSimilarCode(?r1,?r2)∧ isForkCount(?r1,?fc)∧swrlb:greaterThan(?fc,0)∧isFork(?r2,null)→ **S2**(?r2) |
| GHRepository(?r)∧isFileCount(?r,?fc)∧swrlb:greaterThan(?fc,0)∧ isLicenseFile(?r,null)∧isReadmeFile(?r,?rmf)∧isContentCount(?rmf, ?rmcc)∧swrlb:greaterThan(?rmcc,0)∧isContent(?rmf,?c)∧ swrlb:booleanNot(contains(?c,[License]))→**S5**(?r) |
| GHRepository(?r)∧isFileCount(?r,?fc)∧swrlb:greaterThan(?fc,1)∧ isLicenseFile(?r,?lf)∧isContentCount(?lf,?fctc)∧ swrlb:greaterThan(?fctc,0)∧isContent(?lf,?c)∧isPathCount(?lf,?pc)∧ swrlb:greaterThan(?pc,0)∧isPath(?lf,?p)∧isCommitCountByPath(?r, ?ctp)∧swrlb:greaterThan(?ctp,1)∧isCommitByPath(?r,?cl)∧ isCommitCodeChange(?cl,?cc)∧swrlb:contains(?cc,[License])∧ swrlb:booleanNot(contains(CHANGELOG,?cc))∧ isPullRequestCountByCommit(?r,?prc)∧swrlb:lessThan(?prc,1)→**S6**(?r) |
| GHIssue(?i)∧isIssueMessageBody(?i,?b)∧swrlb:contains(?b, "https://github.com/")∧isWebLink(?b,?r2)∧swrlb:notEqual(?r1,?r2)∧ isIssueOwner(?i,?u)∧swrlb:booleanNot(isContributor(?r1,?u))∧ swrlb:booleanNot(contains(?b,[DEMO]))∧isContributor(?r2,?u)→**S8**(?i) |
| GHRepository(?r)∧isLatestRelease(?r,?lr)∧isPublishedDate(?lr,?d)∧ durationGreaterThan(0.5,?d,"Years")∧isFork(?r,null)∧isFileCount(?r, ?fc)∧swrlb:greaterThan(?fc,0)∧isReadmeFile(?r,?rm)∧ isContentCount(?rm,?rmcc)∧swrlb:greaterThan(?rmcc,0)∧isContent(?rm, ?c)∧swrlb:contains(?c,"https://play.google.com")∧isWebLink(?c,?l)∧ openStream(?l,?s)∧swrlb:contains(?s,"in-app purchase")→**S9**(?r) |

***Overview of Etor.*** Figure 4 presents the overall architecture of our automatic detection tool, Etor. Etor supports detection of unethical behavior for two levels, including: (1) repository (denoted as `repo`),

and (2) GitHub issue/pull request (we denote an issue as `issue` and a pull request as `PR`). Given a `repo` or an `issue/PR`, and the type of unethical behavior `eType` to be checked, the Etor relies on its set of SWRL rules for its detection, and produces as output whether there is a violation of `eType` in the given input. Apart from GitHub attributes in Table 3 that can be detected using the GitHub API, our SWRL rule reasoner uses two additional components for its detection: (1) license detector that checks for licenses at the repository level, and (2) code similarity checker that identifies similar code.

***Supported Types.*** Etor supports six types of unethical behavior. We describe the SWRL rules for all supported types in Table 4. We next describe how Etor checks each supported type.

***(S1)*** *No attribution to the author in code.* Etor checks if an issue or a PR has a Stack Overflow link representing a reference code, and the code snippet copied from Stack Overflow cites the reference link. Although there can be many resources from which stakeholders copy the reference code, Etor only check for Stack Overflow links because (1) we learned from our study and from existing work [25] that contributors are required to give credit to copied code snippets in Stack Overflows as they are protected by the CC-BY-SA Creative Commons license, and (2) to support other online resources (e.g., GitHub links), we need to automatically extract the original reference code (requires parsing Web pages of different formats), and identify the appropriate license for the code snippet (requires detecting the license for partial code, which is beyond the scope of this paper). Given an `issue/PR`, Etor checks if a comment `b` in the `issue/PR` posted by a stakeholder `u1` contains the Stack Overflow link (`w`) (we use regular expression to extract `w`). Etor reports a potential violation if: (1) `u1` is not the owner of the Stack Overflow comment, (2) the code snippets from Stack Overflow are found in one of the files in the repository (`F`) with at least 10% similarity (copyright law permits the use of up to 10% of work without permission [6]), and (3) `w` is not found in `F`.

***(S2)*** *Soft forking.* Given two repositories `r1` and `r2`, Etor compares the contents of all source files in the two repositories to check if one repository is a *soft-fork* (the repository has the same content but it is not listed as an official fork of another repository) of another repository. Specifically, we use `AC2` [7] to detect the similarities between files. AC2 is a source code plagiarism detection tool that has been widely used by graders to detect plagiarism within a group of assignments. We select AC2 because (1) it supports many programming languages (e.g., C, C++, Java, and PHP), (2) it can be run in a local environment without connection to remote servers, and (3) it is quite robust as it incorporates multiple algorithms found in the literature. Etor reports a violation if it detects: (1) 100% similarity between `r1` and `r2`, and (2) `r2` is not in the fork list of `r1`.

***(S5)*** *No license provided in public repository.* Given a repository `r`, Etor detects the repo-level license by checking if it exists in the: (1) `LICENSE` file [8] in the main directory of `r`, (we check only in the main directory to avoid mistakenly finding API license or package license) or (2) `README.md` file with license information (we use the list of licenses provided by GitHub [9] for repo-level license detection). Etor reports a potential violation if no license is found after searching for the two files.

***(S6)*** *Uninformed license change.* We consider a change to be *uninformed* if (1) it is not announced in the `CHANGELOG.md` or (2) the license change is not done via PR. Given a repository `r`, Etor checks if the repo-level license has been changed by: (1) extracting commit lists of the license file, and (2) checking if commit changes include license updates. If the license changes occur in more than one commit (we ignore the first commit as it is the initial license creation), Etor checks whether the changes have been announced in the `CHANGELOG.md` by checking whether the `CHANGELOG.md` mentions license information. If license information is not found, Etor checks the PR count for the commit (`pullRequestCountByCommit`). If the count is less than one, Etor marks it as a potential violation.

***(S8)*** *Self-promotion.* We consider *self-promotion* to be the scenario where a contributor `u` opens a GitHub issue/PR where the content of the issue/PR includes links to another repository in GitHub to promote their own repository. Given an `issue/PR` for `r1` as input, Etor first (1) checks that the `issue/PR` includes a link `L` to another repository `r2`, and (2) identifies the stakeholder `u` who opens the `issue/PR`. Then, it reports a violation if: (1) `r1` is not `r2`, (2) `u` is not a contributor of `r1` (i.e., `u` is an outsider for `r1`), and (3) `u` is a contributor of `r2`. To reduce false positives, Etor also checks if `L` includes specific keywords that usually indicate that the contributor is sharing the link `L` for demonstration purposes (e.g., `[DEMO]`) instead of promoting a repository/library ("\issues\", "\pull\", "\commit\", "\tree\", "\releases\", "\blob\", and "\runs\").

***(S9)*** *Unmaintained Android Project with Paid Service.* This type checks whether an Android project offered paid service in Google Play, but stopped actively maintaining the GitHub repository. On average, 115 APIs are updated per month [48], and 49% of app updates have at least one update within 47 days [50]. Based on this frequency of app updates, we define an *unmaintained Android project* to be an Android project where the latest update is released less than 0.5 year. Given a repository `r` as input, Etor first checks for unmaintained Android projects by examining whether (1) the latest release date (`D`) of `r` is less than 0.5 year, and (2) `r` is an *original repository* (not forked from other repositories). Then, it checks whether the app offers a paid service by (1) identifying the Google Play link `l` from `r`, and (2) searching for the "in-app purchase".

## 5 EVALUATION

We applied Etor on 195,621 GitHub issues and PRs of 1,765 GitHub repositories to address the following research questions:

**RQ3:** How many unethical issues can Etor detect in OSS projects?
**RQ4:** What are the accuracy and efficiency of Etor in its detection?

By counting the number of unethical issues in OSS projects, RQ3 provides a rough estimation of the prevalence of each type of unethical behavior in OSS projects. For RQ4, we measure the accuracy and efficiency of Etor using the following metrics:

**True Positive (TP):** Etor classifies an unethical behavior as a potential violation, and it is a true violation.

**False Positive (FP):** Etor incorrectly classifies an unethical behavior as a potential violation, and it is a false violation.

**Time:** The average time taken (in seconds) to detect a type of unethical behavior across all the evaluated repositories/issues.

**Selection of projects/issues.** As there is no prior benchmark for evaluating the detection of unethical behavior, we construct a dataset by crawling GitHub. Our goal is to select the most recent popular (most stars and most forks) OSS projects and the GitHub issues/PRs from OSS projects for evaluation. We first obtain the list

**Table 5: Number of issues detected and TP/FP rate**

| Type | # Unethical Issues | True Positive | | False Positive | | Time (s) |
|---|---|---|---|---|---|---|
| | # repos or issues / Total | # repos or issues / Total | % | # repos or issues / Total | % | |
| (S1) No attribution to the author in code | 80 / 195,621 issues | 59 / 80 issues | 74 | 21 / 80 issues | 26 | 5.4 |
| (S2) Soft forking | 10 / 100 repos | 10 / 10 repos | **100** | 0 / 10 repos | **0** | 343.1 |
| (S5) No license provided in public repository | 476 / 1,765 repos | 424 / 476 repos | 89 | 52 / 476 repos | 11 | 3.1 |
| (S6) Uninformed license change | 18 / 1,765 repos | 16 / 18 repos | 88 | 2 / 18 repos | 11 | 9.2 |
| (S8) Self-promotion | 116 / 195,621 issues | 37 / 116 issues | 32 | 79 / 116 issues | 68 | 4.3 |
| (S9) Unmaintained Android Project with Paid Service | 3 / 1,765 repos | 2 / 3 repos | **66** | 1 / 3 repos | **33** | 5.3 |
| **Average** | - | - | **74.8** | - | **24.8** | - |

of the top 2,000 OSS projects (we first get the top 1,000 projects with the greatest number of stars, and then the top 1,000 projects with the greatest number of forks) created last year (2021). After eliminating duplicated projects, there are 195,621 GitHub issues/PRs of 1,765 projects in our evaluation set. As soft forking requires two repositories as input, we obtain the pair of repositories ($repo1, repo2$) by getting $repo1$ from the top 200 projects (first 100 from most stars, subsequent 100 from most forks) from the initial list of 2,000 projects. From these 200 projects, our crawler automatically identifies $repo2$ by searching GitHub for projects with similar names using the name of $repo1$ as the query. At this step, our crawler found only 10 out of the 200 projects that have repositories with similar names. For each of these 10 projects, our crawler retrieves the first 10 repositories from the search results as $repo2$, leading to a total of 10*10=100 projects for evaluating soft forking.

Example 1. *We are researchers conducting research on mining software repositories. May we know the license for the repo? GitHub also encourages us to have a license for public repos (under "Licensing a repository - GitHub Docs"). We would like to highlight this as a transparency concern.*

**Ethical Considerations.** Before getting feedback from stakeholders, we obtained IRB approval from our institution to maintain the ethical integrity of research. Moreover, calling out stakeholders for violations of unethical behavior could potentially lead to similar ethical concerns in prior work [66]. Therefore, we choose to evaluate Etor by (1) manually inspecting the reported issues, and (2) reporting only the types of unethical behavior with high accuracy (based on our manual analysis). Additionally, to avoid violating ethical principles as in the "hypocrite commits" incident, as shown in Example 1, we explicitly mentioned in each reported issue that we are researchers conducting research on mining software repositories. Furthermore, to reduce author bias in manual TP/FP classification, a non-author labels each issue independently.

All experiments are conducted on a machine with Intel(R) Core (TM) i7-7500 CPU @2.7 GHz and 16 GB RAM.

**Implementation.** We use Protégé 5.5.0 [57] for the ontology model. Our crawler uses PyGitHub, and github-api for querying GitHub.

## 5.1 RQ3: Number of Detected Issues

Table 5 summarizes the results of our evaluation. The "Type" column denotes the types of unethical behavior detected by Etor, whereas the second column is of the form $x$ / $y$ where $x$ represents the number of repositories/issues with the unethical behavior detected and $y$ denotes the total number of repositories/issues in our evaluation dataset. Overall, Etor has successfully detected at least one violation for all types of unethical behavior that we studied. As our evaluation dataset is different from the study dataset, and we have observed the occurrences of unethical behavior in both datasets, this indicates that *different types of unethical behavior are prevalent in OSS projects*. Table 5 also shows that "No license provided in public repository" is the most common type among the six types of detected issues. This means that *a relatively high percentage of the evaluated repositories are missing license files* (around 24% of the evaluated repositories if we exclude the false positives). For the issue-level detection, we observe that "No attribution to the author in code" and "Self-promotion" are the most common ones among all evaluated issues/PRs. This indicates that *contributors of OSS projects tend to (1) forget to give credit to the author in their copied code snippets, or (2) promote their own repositories without mentioning they are contributors to the repositories.*

## 5.2 RQ4: Accuracy and Efficiency of Etor

**Accuracy.** To evaluate the effectiveness of Etor, two raters (one author, and one non-author who is an undergraduate CS student working as a part-time student assistant) independently inspect its output. Specifically, for each violation reported by Etor, each rater determines if the violation is a true positive (TP) or a false positive (FP). The initial Cohen's Kappa was 0.82, which indicates a high level of agreement. The two raters then meet to resolve any disagreement to reach Cohen's Kappa of 1.0. The "True positive" and "False positive" columns in Table 5 show the results for the inspection. On average, the TP rate is 74.8%, and the FP rate is 24.8%. For repository-level detection, although Etor can only detect a small number of violations for "Soft forking", it can detect these unethical issues with high accuracy (0% FP rate). As we consider a repository a *soft-fork* only if all the contents of the two repositories are the same (100% similarity), this design decision may lead to fewer violations being found but increase the accuracy of its detection. In future, it is worthwhile to study the effect of the similarity threshold on the accuracy of its detection. For issue-level detection, Etor can detect S1 with reasonable accuracy (26% FP rate).

**Efficiency.** The "Time" column in Table 5 shows the average time taken to detect an unethical behavior. Overall, the average time to analyze a repository is 3.1–343.1 seconds and the average time taken to analyze an issue is 4.3–5.4 seconds. This indicates that *Etor can detect a type of unethical behavior relatively fast.* We also observe that "Soft forking" is the most time-consuming type to

detect because Etor needs to check for code similarities for all source files within the repository.

**Reasons behind Inaccurate Detection.** We review Etor's reported FPs manually. Etor reports the highest FP rate for "Self-promotion". Recall that Etor checks that a stakeholder `St` opens an issue/PR `I` at repository `R1`, and includes the other repository (`R2`) link (`L`). A true "Self-promotion" only occurs if `St` did not mention about being a contributor of `R2`. We need to manually verify this condition by reading the comments written in natural language. Hence, FPs may occur if (1) `St` mentioned that they are contributors of `R2` (e.g., "I am working on a project called the ..." in comment [10]) or (2) `St` wanted to ask for suggestion in using `R1` for `R2` (e.g., "I'd like to try your ... module in a non-mmdetection repo" [15]).

In "No attribution to the author in code", three reasons exist : (1) no actual copying occurs but a link exists (e.g., the Stack Overflow link was mentioned as references [17]), (2) Etor checks the exact link and fails to detect if the citation uses the short link of Stack Overflow, and (3) Etor matches the exact GitHub user name with the Stack Overflow user name, and fails to detect if the user name is different (e.g., GitHub user name is `usera2` and Stack Overflow user name is `User A` [14]). For "No license provided in public repository", FPs occur because the repository (1) has a license file that is not in the main directory (e.g., `LICENSE` file in the inner folder [13]), (2) has a disclaimer in `README.md` (e.g., "This repository is for personal study and research purposes only. Please DO NOT USE IT FOR COMMERCIAL PURPOSES." [18]), (3) is used for education purposes (we need to manually exclude repositories for the public schools where the license is not required), (4) has no source code or data, and (5) is under an organization license and no separate license is defined for the repository [16]. For "Uninformed license change", FPs occur because the scenario where the repository has restored the old license should not be considered a violation (e.g., the stakeholder changed the license from "Apache License Version 2.0" to "GNU GENERAL PUBLIC LICENSE Version 3", and they restored back to "Apache License Version 2.0". For "Unmaintained Android Project with Paid Service," we found one FP because the unmaintained project is a library used by an app instead of the app itself but the app is actively maintained (with a new release).

**Stakeholders' Feedback.** Apart from manually labeling the unethical issues, we also obtained qualitative feedback by reporting them to stakeholders of OSS projects. To avoid spamming OSS developers with inaccurate results, we only reported the types of unethical behavior with >=80% TP rate in our manual analysis (i.e., **S2**, **S5**, **S6**). For each of these reported types, we opened a GitHub issue to developers when both raters labeled it as TP. We excluded 39 issues because the project owners have disabled GitHub issues (this usually indicates that they do not accept contributions or bug reports [11]). For example, the repository [12] violates the "No license provided in public repository" rule but we cannot report this to the project owner as GitHub issues have been disabled. We also found 19 issues that were previously reported and fixed the issues before we filed a bug report. In total, we have reported 392 issues, and received 83 replies (a response rate of 21.17%) from stakeholders. We carefully looked through all those responses and identified 68 (81.93%) replies as valid and 15 (18.07%) responses as invalid. Among these 68 valid replies, 39 (57.35%) have been fixed, and 29 (42.65%) have been accepted by the stakeholders of the OSS. An example valid feedback that we received is "Thank you very much for the warning. I have already added the license to the repos that didn't have it.". For the 15 responses that we considered as invalid, developers (1) directly deleted or closed our submitted issues without any explanations (7/15), (2) thought that the issue reporter was a software bot although we have created the issue manually and explicitly mentioned in the issue that we are researchers (5/15), (3) are not interested in getting any GitHub issues (e.g., claiming that the repository is personal) (2/15), and (4) explained that "Software is not open source but everyone or you can use my soft without license. thank you for support my soft" (1/15).

## 6 DISCUSSION AND IMPLICATIONS

We discuss practical takeaways and suggestions below:

**Implications for Stakeholders of OSS Projects.** By reading issues that stakeholders in OSS considered as "unethical behavior", our study revealed that the types of unethical behavior in OSS projects are diverse (Finding 1), suggesting that stakeholders of OSS projects should be better educated to create awareness of the different types of unethical behavior when contributing to OSS projects to avoid violating ethical principles. Apart from general types of unethical behavior, our study also pinpoints six new types of unethical behavior in OSS projects (i.e., (S2) Soft forking, (S6) Uninformed license change, (S8) Self-promotion, (S9) Unmaintained project with paid service, (S11) Naming confusion, and (S12) Closing issue/PR without explanation). Some of them are related to the unique features of GitHub (e.g., "Soft forking" represents ethical concerns when forking, "Closing issue/PR without explanation" are related to closing GitHub issues/PRs, "Self-promotion" occurs due to the need to promote the popularity of one's new repository, whereas "Unmaintained project with paid service" denotes the responsibility of an OSS project owner to actively maintain the project to support paid users). The identified new types call for consideration of the unique context of OSS projects when studying unethical behavior. Novice contributors (e.g., students) should also be aware of unethical behavior when contributing to open-source projects [61]. Meanwhile, although most software development efforts focus on source code maintenance, our study urges OSS project owners to be responsible for the product name selection to avoid violating "Naming confusion". As issues related to copyright and licensing are the most common ones (Finding 2), contributors of OSS projects should pay more attentions in giving appropriate credits, and selecting suitable software licenses when copying software artifacts or using library. Meanwhile, although source code is still the most common affected software artifact (Finding 4), our study urges OSS stakeholders to be responsible for various types of software artifacts (Finding 3) to avoid violating ethical principles when uploading them to GitHub.

**Implications for Researchers and Tool Designers.** As many of the identified types of unethical behavior (Finding 1) are ethical issues that frequently occur in daily life, our study provides empirical evidence that there exists some overlaps between the types that occur under general setting (e.g., "Plagiarism" and "Offensive language") and those that are deemed as unethical behavior by stakeholders in OSS projects. Indeed, the prevalence of plagiarism is inline with prior study which reported the prevalence of the

code borrowing practices in GitHub [38]. Due to the diverse types of unethical behavior, future empirical research should advance beyond the general types of unethical behavior.

While existing work mostly focuses on license incompatibility [35, 43, 67], our study found new issues related to licensing. e.g., "Uninformed license change". As these issues still occur frequently (Finding 2) and our study identified new types of issues, our study provides empirical motivations for improving techniques related to copyright and software licenses. For the newly identified types of unethical behavior, we foresee a huge potential for future research direction in: (1) conducting more in-depth study in the motivations and the common solutions behind each type of unethical behavior, and (2) introducing automated techniques that can detect and possibly resolve these issues. We believe that our taxonomy of 316 GitHub issues and our tool that uses software artifacts and data available in GitHub API lay the foundation for future approaches on automated detection of unethical behavior.

Although source code remains the most common affected artifact (Finding 4), other artifacts in natural language (e.g., PR/Issue comments, product names, and website) are also common in our study (Finding 3). A promising research direction is to apply natural language processing techniques to accurately detect affected software artifacts in natural language. For example, techniques can be designed to automatically extract and recommend descriptive yet non-conflicting names (e.g., package names) to avoid "Naming confusion". Another future direction is to design techniques that can automatically identify disclaimer-like statements to accurately detect "Closing issue/PR without explanation" (to detect the explanation for the PR/issue), and "Self-promotion" (to extract statement where the stakeholder has mentioned being a contributor).

**Challenges in Automated Detection of Unethical Behavior.** To provide guidelines for future research on the automated detection of unethical behavior, we discuss several challenges identified in our study and evaluation:

- As shown in our study in Section 3.3, the types of artifacts affected by the unethical behavior are quite diverse. An accurate detection technique needs to support analysis of various types of artifacts, including source code, data, and websites.
- Within GitHub, we notice that discussions and announcements in GitHub spread across multiple web pages (issues, PRs, wikis, discussions, and commit logs). With the rapid growth of different web page types in GitHub, it poses additional challenges for automated approaches to exhaustively analyze all relevant web pages.
- Some discussions of unethical behavior occur outside of GitHub (e.g., emails, slack channel). For example, for "Self-promotion", we cannot check whether the stakeholder has communicated with the developers in advance through emails. Without complete information about the discussion, the detection is bound to be inaccurate.
- The scope for the detection can be too broad for some types of unethical behavior (e.g., "Naming confusion"). Without a predefined scope of detection (package name collision versus app name collision), we cannot accurately detect the behavior.
- There exist ambiguities for certain unethical behavior, which makes it difficult even for human beings to reach consensus (e.g., whether the language used is offensive). In this case, an automated tool can present all relevant information to help stakeholders in making more grounded ethical decisions.

# 7 THREATS TO VALIDITY

**External.** Our study mainly establishes the ethical principles based on four prior works (i.e., [29, 34, 46, 63]). Additional literature might have exposed more ethical dimensions in OSS projects. Our findings of unethical behavior may not generalize beyond the studied OSS projects and issues/PRs. There could be unethical behavior that is not reported to the issue tracker. Unfortunately, there is no conceivable way to study these unreported issues. As some issues may not have the ethics-related keywords that we used for searching, we could have also missed some unethical behavior. Nevertheless, our selected keywords already help us in discovering many types of unethical behavior. Hence, we believe the issues in our study provide a representative sample of the reported and resolved unethical issues in our studied repositories. While other types of unethical behavior discovered in our study are important, Etor can only detect six of them, and our evaluation is limited to these six types. Nevertheless, our experiments show that Etor can detect unethical behavior with relatively high accuracy. For "No attribution to the author in code", Etor checks Stack Overflow links but can also establish a basis for code attribution verification.

**Internal.** Our code and scripts may have bugs that can affect the reported results so we made them publicly available for inspection.

# 8 CONCLUSION

To better understand unethical behavior in OSS projects, we conduct a study of the types of unethical behavior in OSS projects. By reading and analyzing the discussion of stakeholders in OSS projects, our study of 316 GitHub issues identifies 15 types of unethical behavior. These unethical behaviors are affected by various types of software artifacts. Inspired by our study, we propose Etor, an ontology-based approach that can automatically detect unethical behavior. Our evaluation of Etor on 195,621 issues (1,765 repositories) shows that Etor can automatically detect 548 issues with 74.8% TP rate on average (up to 100% TP rate). As the first study that investigates the types of unethical behavior in OSS projects, we hope to raise awareness among OSS stakeholders regarding the importance of understanding ethical issues in OSS projects. While Etor shows promising results in automated detection of unethical behavior in OSS projects, we plan to enhance Etor in the future to detect more types (e.g., gender bias) and reduce false positives using machine learning techniques.

# 9 DATA AVAILABILITY

The dataset and the tool are publicly available at the link[17].

# ACKNOWLEDGMENT

The authors would like to thank the reviewers for their insightful feedback and Xinyu Yao for helping with the evaluation process.

# REFERENCES

[1] [n.d.]. https://www.w3.org/2001/sw/#owl
[2] [n.d.]. http://www.w3.org/Submission/SWRL/
[3] [n.d.]. https://github.com/PyGithub/PyGithub
[4] [n.d.]. https://github-api.kohsuke.org/
[5] [n.d.]. https://docs.github.com/en/rest/repos
[6] [n.d.]. https://www.legislation.gov.au/Details/C2017C00180

[17] https://github.com/EtorChecker/Etor

[7] [n.d.]. https://github.com/manuel-freire/ac2
[8] [n.d.]. https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions/adding-a-license-to-a-repository
[9] [n.d.]. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository
[10] [n.d.]. https://github.com/Anarios/return-youtube-dislike/issues/401
[11] [n.d.]. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/enabling-features-for-your-repository/disabling-issues
[12] [n.d.]. https://github.com/rydercalmdown/package_theft_preventor
[13] [n.d.]. *ailab.* https://github.com/bilibili/ailab
[14] [n.d.]. *Are we correctly handling console.Console in node objectKeys(console)?* https://github.com/sindresorhus/ts-extras/issues/50
[15] [n.d.]. *CUDA vs Naive Speedup?* https://github.com/d-li14/involution/issues/1
[16] [n.d.]. *DogeBot2.* https://github.com/DGXeon/DogeBot2
[17] [n.d.]. *Squeeze tooltip in the sections panel.* https://github.com/livebook-dev/livebook/pull/536
[18] [n.d.]. *VIP.* https://github.com/Oreomeow/VIP
[19] [n.d.]. What is Plagiarism? ([n. d.]). https://www.plagiarism.org/article/what-is-plagiarism
[20] 2021. , Report on University of Minnesota Breach-of-Trust Incident pages. https://lwn.net/ml/linux-kernel/202105051005.49BFABCE@keescook/
[21] Anneliese Amschler Andrews and Arundeep S Pradhan. 2001. Ethical issues in empirical software engineering: the limits of policy. *Empirical Software Engineering* 6 (2001), 105–110. https://doi.org/10.1023/A:1011442319273
[22] Grigoris Antoniou and Frank van Harmelen. 2004. Web ontology language: Owl. In *Handbook on ontologies*. Springer, 67–92.
[23] Deepika Badampudi. 2017. Reporting Ethics Considerations in Software Engineering Publications. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 205–210. https://doi.org/10.1109/ESEM.2017.32
[24] Sebastian Baltes and Stephan Diehl. 2019. Usage and Attribution of Stack Overflow Code Snippets in GitHub Projects. *Empirical Softw. Engg.* 24, 3 (jun 2019), 1259–1295. https://doi.org/10.1007/s10664-018-9650-5
[25] S. Baltes, R. Kiefer, and S. Diehl. 2017. Attribution Required: Stack Overflow Code Snippets in GitHub Projects. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 161–163. https://doi.org/10.1109/ICSE-C.2017.99
[26] Dizza Beimel and Mor Peleg. 2011. Using OWL and SWRL to represent and reason with situation-based access control policies. *Data Knowledge Engineering* 70, 6 (2011), 596–615. https://doi.org/10.1016/j.datak.2011.03.006
[27] Stephen R Bergerson. 2000. E-commerce Privacy and the Black Hole of Cyberspace. *Wm. Mitchell L. Rev.* 27 (2000), 1527.
[28] Hanene Boussi Rahmouni, Tony Solomonides, Marco Casassa Mont, and Simon Shiu. 2009. Modelling and Enforcing Privacy for Medical Data Disclosure across Europe. *Studies in health technology and informatics* 150 (02 2009), 695–9. https://doi.org/10.3233/978-1-60750-044-5-695
[29] Mark Cenite, Benjamin H. Detenber, Andy W.K. Koh, Alvin L.H. Lim, and Ng Ee Soon. 2009. Doing the right thing online: a survey of bloggers' ethical beliefs and practices. *New Media & Society* 11, 4 (2009), 575–597. https://doi.org/10.1177/1461444809102961 arXiv:https://doi.org/10.1177/1461444809102961
[30] Jason A Colquitt. 2001. On the dimensionality of organizational justice: a construct validation of a measure. *Journal of applied psychology* 86, 3 (2001), 386. https://doi.org/10.1037/0021-9010.86.3.386
[31] Daniela S. Cruzes and Tore Dyba. 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*. 275–284. https://doi.org/10.1109/ESEM.2011.36
[32] Daniela America da Silva, Henrique Duarte Borges Louro, Gildarcio Sousa Goncalves, Johnny Cardoso Marques, Luiz Alberto Vieira Dias, Adilson Marques da Cunha, and Paulo Marcelo Tasinaffo. 2021. Could a Conversational AI Identify Offensive Language? *Information* 12, 10 (2021). https://doi.org/10.3390/info12100418
[33] T. Eisenbarth, R. Koschke, and D. Simon. 2003. Locating features in source code. *IEEE Transactions on Software Engineering* 29, 3 (2003), 210–224. https://doi.org/10.1109/TSE.2003.1183929
[34] Batya Friedman, Peter H. Kahn, Alan Borning, and Alina Huldtgren. 2013. *Value Sensitive Design and Information Systems*. Springer Netherlands, Dordrecht, 55–95. https://doi.org/10.1007/978-94-007-7844-3_4
[35] Daniel M. German, Yuki Manabe, and Katsuro Inoue. 2010. A Sentence-Matching Method for Automatic License Identification of Source Code Files. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering* (Antwerp, Belgium) *(ASE '10)*. Association for Computing Machinery, New York, NY, USA, 437–446. https://doi.org/10.1145/1858996.1859088
[36] Daniel M. German, Gregorio Robles, Germán Poo-Caamaño, Xin Yang, Hajimu Iida, and Katsuro Inoue. 2018. "Was My Contribution Fairly Reviewed?" A Framework to Study the Perception of Fairness in Modern Code Reviews. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 523–534. https://doi.org/10.1145/3180155.3180217
[37] Nicolas E. Gold and Jens Krinke. 2020. Ethical Mining - A Case Study on MSR Mining Challenges. In *2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR)*. 265–276. https://doi.org/10.1145/3379597.3387462
[38] Yaroslav Golubev, Maria Eliseeva, Nikita Povarov, and Timofey Bryksin. 2020. A Study of Potential Code Borrowing and License Violations in Java Projects on GitHub. In *2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR)*. 54–64. https://doi.org/10.1145/3379597.3387455
[39] Asunción Gómez-Pérez. 2004. Ontology Evaluation. In *Handbook on Ontologies*, Steffen Staab and Rudi Studer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 251–273. https://doi.org/10.1007/978-3-540-24750-0_13
[40] Hsi and Potts. 2000. Studying the evolution and enhancement of software features. In *Proceedings 2000 International Conference on Software Maintenance*. 143–151. https://doi.org/10.1109/ICSM.2000.883033
[41] Syed Fatiul Huq, Ali Zafar Sadiq, and Kazi Sakib. 2019. Understanding the Effect of Developer Sentiment on Fix-Inducing Changes: An Exploratory Study on GitHub Pull Requests. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. 514–521. https://doi.org/10.1109/APSEC48747.2019.00075
[42] Nasif Imtiaz, Justin Middleton, Joymallya Chakraborty, Neill Robson, Gina Bai, and Emerson Murphy-Hill. 2019. Investigating the Effects of Gender Bias on GitHub. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 700–711. https://doi.org/10.1109/ICSE.2019.00079
[43] Georgia M. Kapitsaki, Frederik Kramer, and Nikolaos D. Tselikas. 2017. Automating the License Compatibility Process in Open Source Software with SPDX. *J. Syst. Softw.* 131, C (sep 2017), 386–401. https://doi.org/10.1016/j.jss.2016.06.064
[44] Georgia M. Kapitsaki, Nikolaos D. Tselikas, and Ioannis E. Foukarakis. 2015. An Insight into License Tools for Open Source Software Systems. *J. Syst. Softw.* 102, C (apr 2015), 72–87. https://doi.org/10.1016/j.jss.2014.12.050
[45] A. S. M. Kayes, Wenny Rahayu, Tharam Dillon, and Elizabeth Chang. 2018. Accessing Data from Multiple Sources Through Context-Aware Access Control. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 551–559. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00084
[46] David Kocsis and Gert Jan De Vreede. 2016. Towards a taxonomy of ethical considerations in crowdsourcing. In *Americas Conference on Information Systems*. https://api.semanticscholar.org/CorpusID:12263532
[47] Josh Lerner. 2005. The Scope of Open Source Licensing. *Journal of Law, Economics, and Organization* 21 (04 2005), 20–56. https://doi.org/10.1093/jleo/ewi002
[48] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. 2013. An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *2013 IEEE International Conference on Software Maintenance*. 70–79. https://doi.org/10.1109/ICSM.2013.18
[49] Deborah L McGuinness, Frank Van Harmelen, et al. 2004. OWL web ontology language overview. *W3C recommendation* 10, 10 (2004), 2004.
[50] Stuart Mcilroy, Nasir Ali, and Ahmed E. Hassan. 2016. Fresh Apps: An Empirical Study of Frequently-Updated Mobile Apps in the Google Play Store. *Empirical Softw. Engg.* 21, 3 (jun 2016), 1346–1370. https://doi.org/10.1007/s10664-015-9388-2
[51] Andrew McNamara, Justin Smith, and Emerson Murphy-Hill. 2018. Does ACM's Code of Ethics Change Ethical Decision Making in Software Development?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 729–733. https://doi.org/10.1145/3236024.3264833
[52] Tommi Mikkonen and Linus Nyman. 2011. To Fork or Not to Fork: Fork Motivations in SourceForge Projects. *Int. J. Open Source Softw. Process.* 3, 3 (jul 2011), 1–9. https://doi.org/10.4018/jossp.2011070101
[53] Teemu Mikkonen, Tere Vadén, and Niklas Vainio. 2007. The Protestant ethic strikes back: Open source developers and the ethic of capitalism. *First Monday* 12 (2007). https://api.semanticscholar.org/CorpusID:6880570
[54] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian Kästner. 2022. "Did You Miss My Comment or What?" Understanding Toxicity in Open Source Discussions. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 710–722. https://doi.org/10.1145/3510003.3510111
[55] Brent Mittelstadt. 2019. Principles alone cannot guarantee ethical AI. *Nature Machine Intelligence* 1 (11 2019). https://doi.org/10.1038/s42256-019-0114-4
[56] Mainack Mondal, Leandro Araújo Silva, and Fabrício Benevenuto. 2017. A Measurement Study of Hate Speech in Social Media. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media* (Prague, Czech Republic) *(HT '17)*. Association for Computing Machinery, New York, NY, USA, 85–94. https://doi.org/10.1145/3078714.3078723
[57] Mark A. Musen. 2015. The Protégé Project: A Look Back and a Look Forward. *AI Matters* 1, 4 (jun 2015), 4–12. https://doi.org/10.1145/2757001.2757003
[58] Christopher Oezbek and Freie Universität. 2010. Research Ethics for Studying Open Source Projects. http://hdl.handle.net/20.500.12424/135960
[59] Rolf-Helge Pfeiffer. 2020. What Constitutes Software? An Empirical, Descriptive Study of Artifacts. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for

Computing Machinery, New York, NY, USA, 481–491. https://doi.org/10.1145/3379597.3387442

[60] J. Singer and N.G. Vinson. 2002. Ethical issues in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 28, 12 (2002), 1171–1180. https://doi.org/10.1109/TSE.2002.1158289

[61] Shin Hwei Tan, Chunfeng Hu, Ziqiang Li, Xiaowen Zhang, and Ying Zhou. 2021. GitHub-OSS Fixit: Fixing Bugs at Scale in a Software Engineering Course. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 1–10. https://doi.org/10.1109/ICSE-SEET52601.2021.00009

[62] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, and Chris Parnin. 2016. Gender bias in open source: Pull request acceptance of women versus men. *PeerJ Prepr.* 4 (02 2016), e1733. https://doi.org/10.7287/PEERJ.PREPRINTS.1733V1

[63] Matteo Turilli and L. Floridi. 2009. The ethics of information transparency. *Ethics and Information Technology* 11 (2009), 105–112. https://api.semanticscholar.org/CorpusID:3043200

[64] Christopher Vendome, Mario Linares-Vasquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. 2015. License Usage and Changes: A Large-Scale Study of Java Projects on GitHub. In *2015 IEEE 23rd International Conference on Program Comprehension*. 218–228. https://doi.org/10.1109/ICPC.2015.32

[65] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. 2017. Machine Learning-Based Detection of Open Source License Exceptions. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 118–129. https://doi.org/10.1109/ICSE.2017.19

[66] Qiushi Wu and Kangjie Lu. 2021. On the feasibility of stealthily introducing vulnerabilities in open-source software via hypocrite commits. In *Proc. Oakland*.

[67] Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. 2023. LiDetector: License Incompatibility Detection for Open Source Software. *ACM Trans. Softw. Eng. Methodol.* 32, 1, Article 22 (feb 2023), 28 pages. https://doi.org/10.1145/3518994

[68] Di Yang, Pedro Martins, Vaibhav Saini, and Cristina Lopes. 2017. Stack Overflow in Github: Any Snippets There?. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 280–290. https://doi.org/10.1109/MSR.2017.13

Received 2023-02-02; accepted 2023-07-27