ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

Identification and Prevention of Code Open Source Quotation and Plagiarism — Innovative Solutions to Enhance Code Plagiarism Detection Tools

Wenya Yang

Beijing University of Technology, Beijing, 100124, China
1265461775@qq.com

Abstract: *This research focuses on the identification and obfuscation prevention of code open source quotation and plagiarism. Through a comprehensive analysis of existing code plagiarism detection tools and open source code features, and starting from the perspective of Alibaba's code management, this study considers measures for code quality management. Furthermore, innovative solutions are proposed to enhance the effectiveness of plagiarism detection tools. These methods aim to strengthen the distinction between open source citation and plagiarism, improve the accuracy of code plagiarism detection tools, and provide support for protecting intellectual property rights, maintaining software engineering ethics and industry ethics, ensuring balanced development of technological innovation, and promoting the development of open source communities.*

Keywords: *Software Engineering Ethics, Plagiarism Detection, Code Open Source, Code Analysis, Code Optimization*

1. Introduction

In today's booming software development field, plagiarism is becoming increasingly prominent, which has a serious negative impact on the entire industry and has aroused widespread concern. Especially in the wide application of open source software, the code of open source projects often becomes one of the main sources of plagiarism. In addition, the blurring of the line between plagiarism and legitimate citation poses challenges to existing detection tools. This research delves into the current status of plagiarism and its threats to software engineering, with a focus on the characteristics of open source code and existing code detection tools. The aim is to propose innovative solutions to enhance detection accuracy and efficiency. Through these efforts, we hope to create a more innovative, and secure environment for the software industry, deepen understanding of plagiarism and open source, strengthen developers' ability to prevent plagiarism, uphold intellectual property rights and software engineering ethics, and promote the positive development of the open-source community.

2. Plagiarism and Open Source

2.1. Current State of Code Plagiarism

Code plagiarism has emerged as a serious problem in the current computer industry. Joy et al. found that student plagiarism persists in educational institutions and shows a trend of increasing. [1] This has triggered profound reflections on the prevalent phenomenon of code plagiarism in today's society. The diversity of code plagiarism is manifested in six aspects. The first is self-plagiarism and reuse of source code, which is an improper borrowing of previous work. Second, copying text directly from books and online resources highlights inappropriate means of obtaining information. The third is theft or paid ghostwriting, which not only infringes intellectual property rights but also undermines the value of independent practice. In addition, inappropriate collaboration, code conversion into another programming language, and forgery also constitute typical manifestations of plagiarism. Miscollaboration and transcoding reflect disregard for collaboration and programming norms, while forgery poses a direct threat to honest academic conduct. Each of these plagiarism behaviors fully reflects the harmfulness of code plagiarism. Additionally, Cosma and Joy suggested that source code can be obtained through various means, including the internet, source code repositories, and textbooks. [2] The increased opportunities for access make plagiarism easier, and the resulting plagiarism issues are not only

ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

present among students but also widespread in other fields. Therefore, the improper use of open source code has become a common phenomenon for achieving the purpose of plagiarism. In the context, this research will thoroughly analyze the specific manifestations of these plagiarism behaviors, aiming to propose innovative methods and tools to more effectively identify and prevent plagiarism involving the improper use of open source code.

2.2. Characteristics of Open Source Code

Open source code, with its unique advantages, is gradually becoming a significant driver of innovation in software engineering. [3] Its explicit licenses ensure the legitimate use of intellectual property, while the active community environment fosters continuous improvement in code quality. Detailed documentation and comments enhance the transparency and comprehensibility of the code, and extensive testing and application validate its stability and reliability. These features not only provide a solid foundation for the legitimate citation of open source projects but also offer strong grounds for preventing plagiarism. Advocating knowledge sharing, we should deeply understand and respect the spirit of open source while accurately distinguishing between open source referencing and plagiarism. This is crucial for ensuring the healthy development of software engineering and the prosperity of the innovation ecosystem. Through in-depth research into the characteristics of open source code, we can more effectively promote high-quality, reproducible research, injecting more vitality into the field of software engineering.

3. Code Inspection Case Analysis

Code plagiarism touches the bottom line of ethics and morality in software engineering, and it is directly related to the quality and sustainability of software projects. Taking industry giants such as Alibaba as an example, its measures to deal with code plagiarism and maintain code quality have important reference value for the whole industry. Alibaba knows that code quality is the lifeline of the software project, in terms of code detection, Alibaba invested a lot of resources and technical force. Through the introduction of advanced automatic detection tools, combined with manual review, a multi-level code quality control system is formed to detect plagiarism in the code in time, and to dig out potential design defects and logic errors to ensure that each line of code meets Alibaba's high standards. In addition, Alibaba also pays great attention to the readability and maintainability of the code. In the process of code review, in addition to checking whether the function implementation is correct, it will also evaluate whether the structure of the code is reasonable, whether the comments are clear, whether the naming is standard, etc.

3.1. Java Code Convention Detection

Alibaba's Java Development Handbook, as a comprehensive and detailed development guideline, provides Java engineers with clear programming guidance. This handbook, derived from the practical experience of numerous technical elites within Alibaba, has undergone multiple validations and refinements through frontline projects. It has now become the standard convention for Java development at Alibaba.

The goals of the Java code convention are clear and explicit: to enhance efficiency, ensure quality, and pursue excellence. Based on unified standards, the convention aims to eliminate communication barriers, reduce redundant work, and thereby improve overall development efficiency. It also emphasizes the prevention of potential issues by utilizing standardized code writing, enhancing system maintainability, and reducing the probability of faults. Additionally, the convention embodies a pursuit of craftsmanship, encouraging developers to polish every line of code with an utmost attitude, creating high-quality works.

To implement the code convention, Alibaba employs various tools and methods for assistance. IDE detection plugins can instantly identify convention violations as developers write code, prompting them to make timely corrections. Pipeline integrated testing conducts automated checks before code submission to ensure that the code quality meets the standards. Integrated code reviews provide more comprehensive checks during the code review phase, helping teams discover potential issues and make improvements. In the Codeup code hosting platform on Alibaba Cloud, the Java code convention detection capability is further integrated and applied. This provides developers with a more convenient and quick inspection method during code submission and review stages, making the execution of code

ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

conventions more efficient and effective.

3.2. Intelligent Code Patch Recommendation

When handling code repositories to identify corrective commits, it is essential to follow a series of rigorous steps, making full use of developers' good commit practices. Firstly, filter out commits with keywords indicating fixes in their commit messages. For the sake of analysis accuracy, focus only on commits involving fewer than 5 files to avoid submissions that involve too many files, potentially masking the actual corrective actions. Extract file-level deletion and addition content from the filtered commits, forming Defect and Patch pairs (DP Pair). To reduce noise and enhance the quality of analysis, an improved DBSCAN method is employed to cluster defective and patch code (Figure 1). This clustering method can group similar defective and patch code together at both the file and code snippet levels. This approach not only reduces noise but also identifies common error patterns from historical code submissions. Utilizing a self-developed template extraction method, the characteristics of defective and patch code are summarized. Code repair templates with generality and reusability are generated by adapting to different variables in the context.

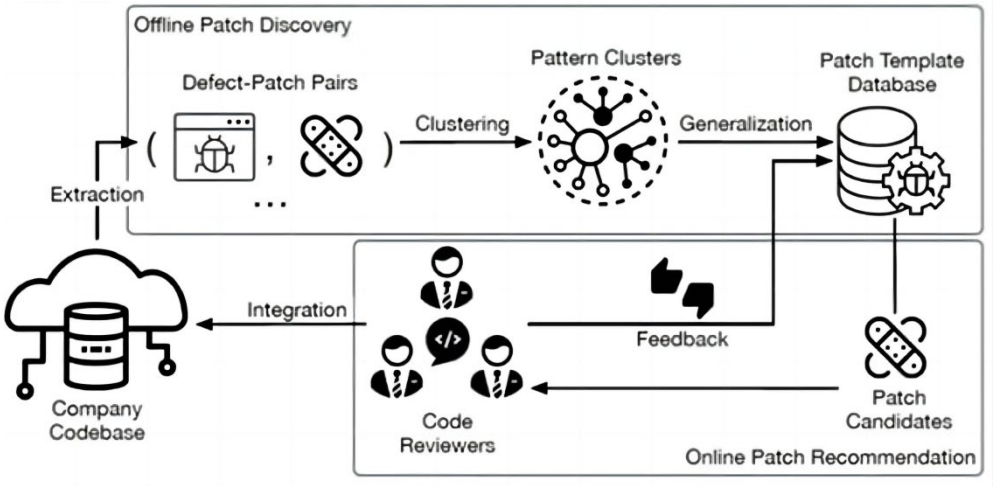

Figure 1: Code Intelligent Patch.

3.3. Dependency Package Vulnerability Detection

In the field of security management for open-source components, Alibaba has established a practical detection and management mechanism for developers, namely the Dependency Package Vulnerability Detection Service and the Dependency Package Security Issue Report. However, in practical operations, many developers find dealing with dependency package vulnerabilities challenging. The cost of fixing these vulnerabilities is often higher than addressing vulnerabilities in their own code. The main reason behind this lies in the complexity of dependency relationships and the uncertainty in version selection.

To simplify this process, a more in-depth identification and analysis of dependencies is required, clearly distinguishing between direct and transitive dependencies. It's crucial to directly link to the specific dependency package introduction files, enabling developers to quickly pinpoint the issues without spending significant time searching in the code ocean. Additionally, aggregating and processing vulnerability data provides intelligent version upgrade suggestions for developers. When a dependency has multiple vulnerabilities, developers can evaluate whether to upgrade based on the recommendations. Furthermore, analyzing API changes and code call chains between different versions helps developers assess the cost and risk of version upgrades. The core of this mechanism is to provide comprehensive information support for developers, enabling them to make informed decisions when dealing with dependency package vulnerabilities. By automating the creation of repair reviews, developers are provided with an efficient platform, allowing them to focus more on the security and stability of their code rather than being troubled by complex dependency relationships.

3.4. Source Code Vulnerability Detection

Alibaba utilizes the Sourcebrella Pinpoint source umbrella detection engine in its source code security detection, bringing revolutionary improvements to the detection of injection and security policy risks.

ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

The Source Umbrella detection engine is built on nearly a decade of in-depth research by the Prism research group at the Hong Kong University of Science and Technology. It not only incorporates the essence of software verification technologies worldwide in the past decade but also innovates and improves upon them, successfully designing and implementing a leading software verification system. The core of this verification system lies in its unique verification approach, transforming programming languages into mathematical forms such as first-order logic and linear algebra. Through formal verification techniques, it deduces the causes of defects. This approach allows the Source Umbrella detection engine to uncover defects hidden for over 10 years in large and active open-source projects—defects often beyond the reach of other detection tools in the market. When detecting a 2 million-line large open-source project, the Source Umbrella detection engine can complete the process in just 1.5 hours while maintaining a false positive rate of around 15%. This achievement sets a benchmark in the industry for complex and extensive analysis projects.

Alibaba's "Source Code Vulnerability Detection" is built upon the outstanding security analysis capabilities of the Source Umbrella detection engine. It demonstrates balanced and excellent performance across multiple dimensions, including analysis accuracy, speed, and depth. The core advantages are highlighted in the following aspects: Firstly, it supports bytecode analysis, ensuring that the logic of third-party package code is thoroughly examined without missing any details. Secondly, it excels in logic analysis across long function call chains, delving into every corner of the code. Moreover, it can handle indirect data modification issues introduced by references, pointers, and similar constructs, which is not commonly found in similar tools. Additionally, compared to tools like Clang and Infer, it outperforms in accuracy and effective issue recognition. Lastly, it exhibits remarkable performance, with the average analysis time for a single application taking only about 5 minutes.

The Source Umbrella detection engine stands out in the field of code analysis due to its precise data flow tracking capability and high-depth, high-accuracy function call chain analysis. This enables it to discover deep issues spanning multiple layers of functions. While uncovering defects, it also provides detailed insights into the triggering process, as well as the relevant control flow and data flow. This offers developers great convenience, assisting them in quickly understanding and resolving issues. Moreover, it enables the improvement of software quality at a lower cost in the early stages of software development, significantly reducing production costs and enhancing development efficiency.

It is evident that Alibaba has achieved significant success in source code security detection by introducing the Sourcebrella Pinpoint source umbrella detection engine. The scientific application of this engine has enhanced the precision and efficiency of code analysis, providing developers with more convenient and effective issue resolution solutions.

4. Analysis of Existing Code Plagiarism Detection Tools

Plagiarism can be detected manually or with plagiarism detection software. Manual inspection requires more effort and time, and is not very useful when there are too many documents to examine or when the original work cannot be compared. Therefore, using plagiarism detection tools is a more useful and efficient method. [4] In addition to the measures taken by enterprises, in order to meet the challenge of code plagiarism, a variety of plagiarism detection tools have emerged. These tools have their own advantages and disadvantages in detecting code similarity, which provides strong support for identifying plagiarism.

JPlag is a system that focuses on finding similarities in multiple sets of source code files, standing out with its token-based analysis approach and the Greedy String Tiling algorithm. [5] This algorithm allows JPlag to deeply detect structural similarities in code, going beyond surface-level character matching. [6] Additionally, JPlag generates detailed reports, providing users with comprehensive detection results, leading to its widespread use in the academic and educational fields. However, in certain situations, JPlag may result in relatively coarse matches due to algorithm limitations, especially when dealing with significant variations in code or substantial structural changes.

MOSS adopts more advanced algorithms compared to JPlag, combining hash functions and tree structure matching, making MOSS more accurate in detecting code similarity, with a particular emphasis on structural similarity. It supports multiple programming languages and can detect similar code that has undergone modifications and variations. However, MOSS has a relatively complex usage and may require a certain learning curve for beginners. Additionally, due to its emphasis on structural similarity, it may struggle to accurately detect code that has undergone significant changes but remains logically similar.

ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

Turnitin, initially designed for detecting plagiarism in academic papers, later expanded into the field of code plagiarism detection. It employs a sophisticated similarity matching method that not only focuses on code similarity but also emphasizes the logic and structure of the code, providing Turnitin with a more comprehensive perspective in detecting code plagiarism. Since its primary design intent was to detect text plagiarism, it might be relatively simpler in code detection, potentially overlooking highly variant code or lacking sufficient support for specific programming languages.

Codequiry is a tool specifically focused on source code detection, using advanced plagiarism detection solutions to find similarities in non-original code and software. The tool supports various mainstream programming languages and emphasizes the detection of modified and transformed code. This makes Codequiry highly accurate in detecting similar code that has undergone modifications and transformations, although it may face challenges in accurately distinguishing structurally similar but logically different code. Additionally, dealing with complex code structures may lead to higher computational complexity.

In conclusion, each of these four plagiarism detection tools has its strengths and weaknesses. When choosing to use them, it is necessary to balance based on specific requirements and scenarios. But they all lack an understanding of the motivation behind the use of open source code, and they can't tell definitively whether code similarity is based on legitimate citations or plagiarism. And these detection systems only automatically detect program similarity, but there is no way to know why the code is similar, so plagiarism still needs to be determined by humans.

5. Innovative Proposal

Faced with the shortcomings of code detection software in distinguishing between reasonable quotations and plagiarism, a series of innovative methods are indeed needed to improve the accuracy and usability of detection tools. This is related to not only the technical level, but also the healthy development of the entire software development industry and the maintenance of software engineering ethics. The author's proposal aims to provide a more comprehensive perspective for plagiarism detection tools through multi-dimensional analysis.

Firstly, it is recommended to enhance the functionality of the code plagiarism detection system by checking the license and authorization information in the code. Authentic open-source projects typically include clear licenses, while instances of plagiarism may not adhere to these license requirements. By analyzing the project's licenses and authorization information, plagiarism detection tools can make preliminary assessments of the code's legitimacy.

Secondly, combining the approach of examining social network data and version control history, plagiarism detection software can incorporate in-depth analysis of social network data and version control history. Genuine open-source projects on platforms like GitHub and GitLab usually have rich social network activities and well-defined version control history, including contributor lists and discussion records. [7] Instances of plagiarism may lack such rich records. Additionally, the version control history of authentic projects should demonstrate a gradual evolution process, while plagiarism often lacks this evolutionary history.

The quality of comments and documentation is also a crucial factor in determining code authenticity. By introducing natural language processing techniques, plagiarism tools can analyze comments and documentation in the code more accurately. Genuine open-source projects typically have comprehensive documentation, while instances of plagiarism may lack detailed explanations and comments. Through in-depth evaluation of the quality of comments and documentation, plagiarism tools can better assess the authenticity of the code.

In terms of technical implementation, it is worth considering the introduction of more advanced code structure and style analysis techniques. Genuine open-source projects typically adhere to a consistent code style, while instances of plagiarism may manifest as a combination of code with different styles. Additionally, behavioral analysis is a crucial step, involving the analysis of code input-output processing, exception handling, and other aspects. Genuine open-source projects usually exhibit clear behavioral patterns, while instances of plagiarism may show inconsistent or chaotic behavior. Through in-depth analysis of the code's structure and style, plagiarism tools can identify differences in similar code more precisely, thereby improving detection accuracy.

Alibaba's detection engine used in its source code security inspection is also a good practice case that can be used as a way to improve code plagiarism detection software. The engine assists detection through

ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

a variety of means, including IDE detection plug-ins, pipeline integration testing and code review integration, which can provide comprehensive inspections at different development stages to help developers find and correct non-compliance issues in a timely manner. IDE plug-ins can detect problems immediately when writing code and remind developers to correct them in time. This is very helpful for improving code quality and reducing post-maintenance costs. Pipeline integration testing provides automated checks to ensure code quality meets standards before it is submitted. This prevents non-compliant code from entering the version control system, thus ensuring the quality of the entire project. Code review integration provides a more thorough review during the code review phase, helping the team identify potential problems and make improvements, which improves team collaboration and code quality.

Combining these methods, our innovative approach aims to provide a more comprehensive perspective for plagiarism detection tools, enhancing detection accuracy and enabling a more precise determination of whether the detected code uses open-source code or involves plagiarism. These improvements not only benefit technical advancements but also contribute to the healthy development of the entire software development ecosystem and uphold software engineering ethics.

6. Summary

In summary, this article provides a comprehensive description of the current state of code plagiarism and the characteristics of open-source code. It makes a detailed analysis of the four existing code plagiarism detection tools: JPlag, MOSS, Turnitin and Codequiry, and Alibaba's code detection management. Building upon the identified shortcomings of these tools, the article proposes innovative methods by considering various factors such as licenses, social networks, version control history, comments, and documentation. The aim is to strengthen the ability of plagiarism detection tools to differentiate between open-source references and actual plagiarism, enhancing accuracy and efficiency. The article offers a broader perspective for the software development field, addressing the challenges of plagiarism in the context of widespread use of open-source software. However, it is acknowledged that the research has certain limitations. Firstly, the proposed methods have not undergone experimental validation, necessitating further verification for their applicability in diverse scenarios and projects to ensure practical effectiveness and operability. Secondly, detecting plagiarism in specific domains or complex projects may still pose challenges, requiring continuous optimization and improvement. Additionally, the methods may demand specialized knowledge and experience in software engineering, prompting exploration of ways to make them more accessible to a wider audience. Future research is expected to optimize the proposed innovative methods further, incorporating advanced technologies like deep learning to improve the performance of plagiarism detection tools in handling large-scale and complex projects. Collaboration with open-source communities, leveraging big data analytics, could facilitate the establishment of more comprehensive open-source code repositories, enabling more precise identification of legitimate references and instances of plagiarism. Overall, this study contributes innovative approaches to the recognition and prevention of code plagiarism in the context of open-source development, offering valuable insights for the evolution of plagiarism detection technology in software engineering. Continued efforts are anticipated to balance the spirit of open source with the need for intellectual property protection, advancing the progress of plagiarism detection technology. By addressing code plagiarism, we can establish a more reliable development environment and drive the software engineering field towards a healthier, more innovative, and sustainable direction.

References

[1] M. Joy, G. Cosma, J. Y. -K. Yau, J. Sinclair. Source Code Plagiarism — A Student Perspective. IEEE Transactions on Education, 2011, 54(1), 125-132.

[2] G. Cosma, M. Joy. Towards a Definition of Source-Code Plagiarism. IEEE Transactions on Education, 2008, 51(2), 195-200.

[3] R. T. Marszalek, L. Flintoft. Being open: our policy on source code. Genome Biology, 2016, 17(1), 172-172.

[4] Vandana. A Comparative Study of Plagiarism Detection Software. In: 2018 5th International Symposium on Emerging Trends and Technologies in Libraries and Information Services (ETTLIS), Noida, India, 2018, 344-347.

[5] Li Guofan, Zhang Feng, Liu Cong. CLPDetector: A Cross-Language Code Plagiarism Detection Tool Based on Pseudo-Twin Network. Journal of Small and Microcomputers, 2022, 43(07), 1562-1568.

ISSN 2616-5775 Vol. 7, Issue 1: 65-71, DOI: 10.25236/AJCIS.2024.070110

[6] L. Prechelt, G. Malpohl, M. Philippsen. Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science, 2002, 8(11), 1016-1038.
[7] Jiang Chenwei. Research and Implementation of Code Plagiarism Detection Technology Based on Version Evolution in Git. Nanjing: Nanjing University of Posts and Telecommunications, 2019.