A Study on Code Clone Lifecycles in Pull Requests Created by AI Agents

Italo Uchoa
State University of Ceará
Fortaleza, Brazil
italo.uchoa@aluno.uece.br

Denis Sousa
State University of Ceará
Fortaleza, Brazil
denis.sousa@aluno.uece.br

Henrique Chuvas
State University of Ceará
Fortaleza, Brazil
henrique.chuvas@aluno.uece.br

Matheus Paixao
State University of Ceará
Fortaleza, Brazil
matheus.paixao@uece.br

Chaiyong Ragkhitwetsagul
Faculty of ICT, Mahidol University
Nakhon Pathom, Thailand
chaiyong.rag@mahidol.ac.th

Thiago Lima
State University of Ceará
Fortaleza, Brazil
thiaguinho.lima@aluno.uece.br

Abstract

Code clones are fragments of code that are copied and reused within the same or across different codebases, often with minor modifications. Their presence poses significant challenges, as defects or changes in one cloned fragment may require consistent updates across all related clones, negatively affecting software maintainability. Code Clone Lifecycle analysis provides valuable insights into when code clones are introduced and how they evolve during the code review process. Recent advances in Large Language Models (LLMs) have enabled Coding Agents that autonomously create branches, modify code, and submit Pull Requests (PRs). While these agents improve productivity, they also introduce new challenges for managing code clones within PRs. This paper presents an analysis of the Code Clone Lifecycle in agentic PRs hosted on GitHub. Using the NiCad clone detection tool, we analyzed 7,851 PRs created by AI agents from the AiDev dataset. Our results identify 28,425 clones across 497 PRs. Manual validation of a representative sample shows a predominance of Type I (29%) and Type III (46.26%) clones. Among the affected PRs, 93 contain clones restricted to a single commit, 320 exhibit clones recurring across multiple commits, and 84 present both single and recurring occurrences. Overall, the findings indicate that clones tend to persist once introduced, progressing through the PR lifecycle and ultimately being merged into the codebase.

CCS Concepts

• **Software and its engineering → Maintaining software**.

Keywords

Code Clones, AI-Agents Pull Requests, Clone Detection

ACM Reference Format:
Italo Uchoa, Denis Sousa, Henrique Chuvas, Matheus Paixao, Chaiyong Ragkhitwetsagul, and Thiago Lima. 2026. A Study on Code Clone Lifecycles in Pull Requests Created by AI Agents. In *23ª Conferência Internacional sobre Mineração de Repositórios de Software (MSR '26), 13 a 14 de abril de 2026, Rio de Janeiro, Brasil*. ACM, Rio de Janeiro, RJ, BR, 5 pages. https://doi.org/10.1145/3793302.3793608

MSR '26, Rio de Janeiro, Brasil

ACM ISBN 979-8-4007-2474-9/2026/04
https://doi.org/10.1145/3793302.3793608

1 Introduction

The growth of open-source software development has reshaped how software systems are built and maintained, enabling large-scale collaboration among developers [7]. Platforms such as GitHub play a central role by supporting collaboration, contribution tracking, and discussion around code changes [14]. On GitHub, collaboration is mainly conducted through pull requests (PRs), which enable proposing, reviewing, and refining changes before integration, helping ensure code quality and alignment with project standards [38].

Pull requests embody the principles of Modern Code Review (MCR), a widely adopted practice in contemporary software development. Through iterative and asynchronous reviews, MCR supports improvements in system quality, maintainability, and knowledge sharing [15, 30]. While large technology companies have long integrated MCR into their workflows [2, 5, 24], it has also become the primary mechanism for code review on GitHub, where pull requests are used to evaluate changes before merging [38].

Code clones are fragments of code copied and reused within the same or across different codebases, often with minor modifications [1, 23, 39]. Their presence poses significant challenges, as defects or changes in one cloned fragment may require consistent updates across all related clones, negatively affecting software maintainability [11, 16, 22]. Code clones are typically detected using static analysis tools, called clone detectors [39].

It is important to understand when code clones are introduced into the system during code review, and to observe whether they are identified and reported by the reviewers. Uchoa et al. [33] introduces a taxonomy that enables the categorization of *Code Clones Lifecycle*, analyzing the commits that include the introduction of code clones in a PR. The *Single Lifecycle* category represents the cases in which clones appear in only one specific commit of the PR. Differently, the *Recurring Lifecycle* category represents the cases in which clones persist or reappear across multiple commits of the PR.

Recent advances in Large Language Models (LLMs) have enabled *Coding Agents* [10, 29, 36] that autonomously modify code and submit pull requests (PRs), referred to as *Agentic-PRs* [13]. While these agents can significantly improve developer productivity [3], their risks and limitations remain underexplored, including the generation of incorrect or insecure code and reduced developer awareness [20, 34]. Additionally, training on large code corpora may lead agents to generate code similar to existing implementations, potentially introducing code clones into software projects.

To the best of our knowledge, no prior study has examined code clones introduced by coding agents. Thus, this study fills in the gap. We analyze the *Code Clones Lifecycle* in *Agentic-PRs* registered on GitHub using the AIDev dataset [13]. Inspired by the study of Uchoa et al. [33], we analyze the commits at which clones emerge in *Agentic PRs* and examine which of these clones are eventually incorporated into the codebase. We conduct this investigation using NiCad [4], a multi-language code clone detection tool.

The main contributions of this work are twofold. First, this study constitutes the first exploratory investigation of the *Code Clones Lifecycle* in *Agentic Pull Requests*. Second, we provide a publicly available *replication package* that supports the analysis of clone lifecycles in pull requests created by AI agents on GitHub [32].

2 Background

Pull Requests (PRs) are the primary mechanism used in modern collaborative software development platforms, such as GitHub, to propose, discuss, and integrate changes into a shared codebase [12]. Within a PR-based workflow, reviewers assess aspects such as correctness, performance, and adherence to project standards, while authors respond by submitting revisions that address the raised concerns [18, 19, 31]. As a result, a single PR may comprise multiple commits, reflecting successive refinements of the proposed changes until they are approved and incorporated into the main branch [8].

Code Clones are code snippets copied and reused within or across codebases, often with minor modifications [1, 23, 39]. In this study, we consider Type-I to Type-III clones [39]: Type-I are identical except for whitespace and comments; Type-II preserve syntactic structure with variations in identifiers and types; and Type-III allow additional statement-level modifications. We exclude Type-IV clones, which are semantically equivalent but syntactically dissimilar and whose identification is highly subjective [26]. To detect clones in *Agentic-PRs*, we use NiCad [4], a text-based clone detection tool that supports multiple languages and accurately identifies Type-I to Type-III clones [25].

Code Clone Lifecycle describes the evolution of clones across PR commits and can be classified into two categories [33].
Single Lifecycle represents the cases in which clones appear in only one commit of the PR, being classified into three subcategories: The **Early Stage** category occurs when clones appear exclusively in the initial commit of a PR and are removed in later commits. The **Mid Stage** category occurs when clones are present in a single commit in the middle of the PR and are removed in subsequent commits. The **Late Stage** category occurs when clones are present only in the last commit of the PR, being merged into the codebase. **Recurring Lifecycle** captures clones that persist across multiple PR commits, with four subcategories: The **Emerging Cycle** category occurs when clones are present from the beginning to the middle of the PR, being removed in the final commits. The **Central Cycle** category occurs when clones appear repeatedly during the middle of the PR, but do not persist to the end. The **Closing Cycle** category occurs when clones are present from the middle to the end of the PR, being merged into the codebase. The **Full Cycle** category occurs when clones are present throughout all commits of the PR, being merged into the codebase.

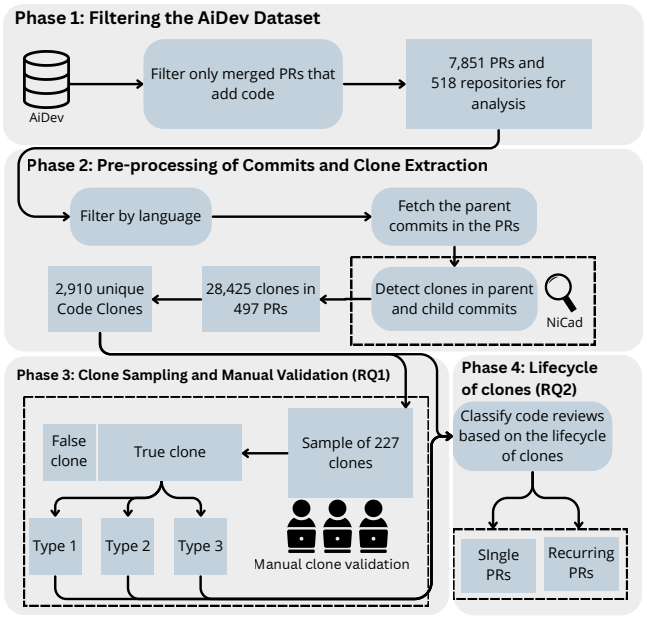

Figure 1: Overview of the methodology (4 phases)

3 Methodology

The empirical basis of this study is the *AiDev* dataset [13], AiDev maps PRs authored by coding agents, capturing their interactions within collaborative development workflows. The dataset comprises 932,791 *Agentic Pull Requests* created by five widely adopted coding agents, *OpenAI Codex* [6], *Devin* [13], *GitHub Copilot* [37], *Cursor* [9], and *Claude Code* [27], and spans 116,211 GitHub repositories. We aim to answer the following research questions:

- *RQ1: What is the distribution of code clone types in* Agentic-PRs*?*
- *RQ2: What is the Code Clones Lifecycle in* Agentic-PRs*?*

To answer the proposed RQs, we carried a four phases methodology, as illustrated in Figure 1. We explain each phase below.

3.1 Phase 1: Filtering the AiDev Dataset

The first phase of the methodology aims to identify merged Agentic-PRs, as only these pull requests effectively modify the target repository and therefore have the potential to introduce code clones into the codebase. Although unmerged PRs may contain code clones, they were not considered in this study because they do not impact the final codebase.

First, we identified the PRs that contained at least one commit that added new lines of source code. Second, we filtered the PRs by retaining only those written in programming languages supported by NiCad: Python, C#, Java, and Ruby. This filtering resulted in a set of 7,851 PRs distributed across 518 repositories, where repositories were selected based on the criterion of containing at least one PR that satisfied the study conditions, as shown in Table 1.

These results form the foundation for the rest of our methodology, in which the identified repositories undergo mining and clone detection processes. The project's dataset, alongside all other

Table 1: Summary of the dataset employed in this study.

Data	Python	C#	Java	Ruby	Total
Repositories	310	134	46	28	518
Pull Requests	5,381	1,136	1,003	331	7,851
Clones	7,533	5,581	8,369	6,942	28,425
Unique Clones	617	316	1,006	971	2,910
Sample	48	25	78	76	227

artifacts created in this study, is available in our replication package [32].

3.2 Phase 2: Pre-processing of Commits and Clone Extraction

The second phase focuses on preparing and analyzing commits from the filtered PRs (Phase 1) to identify clone sets introduced by AI agents. For each programming language, we extracted all projects and the corresponding *child* commits of the filtered PRs. Since the AiDev dataset does not provide the associated *parent* commits, we recovered them directly from the GitHub repositories through a preprocessing step. This process is essential to avoid the rebasing problem [17].

After reconstructing the *parent–child* commit pairs, we ran NiCad on both versions of each commit using parameters for detecting Type I–III clones. We adopted a similarity threshold of 0.3, ensuring at least 70% structural similarity [4], and set the minimum clone size to six lines of code (`minsize=6`) based on prior work [21]. Clone pairs were then clustered into clone classes (`cluster=yes`) to identify groups of related clones.

Using the clone sets extracted from the *parent* and *child* versions, we conducted a direct comparison between them. A clone is considered to have been introduced by an AI agent when: (i) a clone set appears exclusively in the *child* version; or (ii) a clone set already present in the *parent* version is extended in the *child* version through the addition of new cloned code fragments.

We identified 28,425 clone instances across 497 pull requests, with recurrence across commits reflecting expected clone evolution patterns [33]. To reduce bias in manual validation, we deduplicated clones by hashing their code content, grouping identical fragments across commits into single representatives. This process reduced the dataset from 28,425 detected instances to 2,910 unique clones. Table 1 summarizes the analyzed PRs and clone counts.

3.3 Phase 3: Clone Sampling and Manual Validation

In the third phase, we focused on selecting a statistically representative sample of the previously identified unique code clones for manual validation. Using a confidence level of 95%, a margin of error of 5%, and a population of 2,910 unique clones, we determined a final sample size of 227 clones. To preserve representativeness across programming languages, we employed uniform stratified sampling, as presented in Table 1.

With the representative sample defined, we performed a detailed manual analysis of the clones reported by NiCad. Although modern clone detectors exhibit strong performance, they are still susceptible to false positives [35]. Additionally, while NiCad is capable of detecting Type I, Type II, and Type III clones, it does not automatically classify the clone type, making human inspection necessary.

The manual validation process aimed to answer two key questions for each sampled clone: (i) whether the reported clone was a true clone or a false positive, and (ii) for true clones, what its type was. The evaluation was conducted by three researchers with experience in clone analysis. Initially, two evaluators independently inspected each clone. In cases of disagreement, either regarding validity or clone type, a third evaluator provided the final decision.

3.4 Phase 4: Clone Lifecycle Analysis

Following manual validation, we proceeded with the investigation of the lifecycle of code clones introduced in Agentic-PRs. For this step, we used the 497 PRs and the 28,425 code clones identified within these PRs. Given the high precision of Nicad for detect true clones (see Section 4.1), we conducted the analysis across the entire population to obtain a more comprehensive view of the data as performed in previous work [33]. Based on this analysis, we categorized the PRs according to the behavior of their clones, as discussed in Section 2.

4 Results and Discussions

4.1 RQ1: What is the distribution of code clone types in Agentic-PRs?

To address *RQ1*, we conducted a manual validation of a statistically representative sample of 227 clones (see Section 3.3). From this sample, 214 instances were confirmed as true code clones, corresponding to a precision rate of 94.27%. This result indicates that NiCad is a suitable tool for detecting code clones in Agentic-PRs.

Based on this precision rate, we extrapolated the results to the entire population of clones detected by NiCad. Out of the 28,425 reported clone instances, we estimate that approximately 26,796 represent true code clones. These clones were identified across 497 merged *Agentic-PRs*, demonstrating that code cloning is a common phenomenon during the creation of pull requests by AI agents.

Next, we analyzed the distribution of clone types among the 214 validated clones. Table 2 summarizes the distribution of true clone types, as manually identified and classified, across programming languages.. Overall, the most prevalent clones were Type-III (46.2%), followed by Type-I (29%) and Type-II (24.8%), suggesting that AI agents frequently reuse code either verbatim or with substantial modifications. A language-specific analysis reveals distinct distributions of clone types. While all languages exhibit a predominance of Type III clones, Ruby and Java present higher absolute numbers of such clones, whereas C# shows a more balanced distribution among the three clone types. Python, in turn, exhibits a relatively higher proportion of Type II and Type III clones, indicating moderate structural reuse with minor modifications.

Our findings align with prior large-scale studies showing that Type I and Type III clones are more frequent than Type II clones [28, 33]. While AI agents largely follow clone creation patterns similar to human developers, we observe a higher proportion of Type II clones in agent-generated code, indicating a stronger tendency toward systematic renaming and lightweight syntactic adaptations.

Table 2: Distribution of clone types per language.

Language	Type I	Type II	Type III	Total
Python	11 (5%)	13 (6%)	19 (9.2%)	43 (21%)
C#	8 (4%)	6 (3%)	11 (5%)	25 (12%)
Java	27 (13%)	16 (7%)	31 (14%)	74 (34%)
Ruby	16 (7%)	18 (8%)	38 (18%)	72 (33%)
Total	**62 (29%)**	**53 (24.8%)**	**99 (46.2%)**	**214 (100%)**

Table 3: Distribution of clone lifecycle categories in Agentic-PRs per Language

Category	Python	C#	Java	Ruby	Total
Single – Early Stage	28	12	9	2	51
Single – Mid Stage	26	14	8	6	54
Single – Late Stage	40	17	63	1	121
Recurring – Emerging Cycle	15	4	4	9	32
Recurring – Central Cycle	29	9	4	4	46
Recurring – Closing Cycle	73	23	14	11	121
Recurring – Full Cycle	81	34	160	12	287

4.2 RQ2: What is the Code Clones Lifecycle in Agentic-PRs?

According to Table 3, a single PR may be associated with multiple lifecycle categories (e.g., Single–Early Stage, Single–Mid Stage, and Recurring–Emerging Cycle). To avoid double counting and accurately reflect the number of PRs affected by clones, each PR was counted only once based on its unique occurrence, resulting in a total of 497 PRs analyzed. For PRs classified as *Single*, the proportion of clones that are incorporated into the codebase (i.e., Single and Late stages) is consistently lower across most programming languages. Specifically, when PRs are classified as Single, the merge rates of clones are 42.5% for Python, 39.5% for C#, and only 11.0% for Ruby. In contrast, PRs classified as *Recurring* exhibit substantially higher clone merge rates. In these cases, clones are merged into the codebase in 77.8% of Python PRs, 81.4% of C# PRs, and 63.8% of Ruby PRs.

These findings suggest that when code clones are not identified and addressed shortly after their introduction, particularly during the early stages of a PR, they are likely to persist throughout the review process and ultimately be merged into the codebase. These observations are consistent with the findings of Uchoa et al. [33]. Moreover, AI-generated clones show lifecycle and type distributions comparable to those reported for human developers [33].

Java shows the highest proportion of recurring clones that persist throughout the entire PR (*Full Cycle*), with 160 out of 287 cases (55.7%), indicating that once introduced, clones are often reused across commits. In Java, this recurrence may partially reflect language constraints and framework-driven patterns, such as boilerplate code and interface implementations, where duplication can be unavoidable or intentional rather than harmful. In contrast, Python and C# present a more balanced distribution between *Single* and *Recurring* lifecycles, likely due to greater syntactic flexibility and more frequent refactoring practices. Ruby shows lower frequencies across all categories, suggesting that clone recurrence is less common and typically confined to more specific development scenarios.

5 Threats to the Validity

Conclusion and Internal threats: A potential internal threat to validity stems from NiCad's limited language support. To mitigate this issue, we restricted the analysis to languages supported by NiCad and evaluated alternative tools, which either supported fewer languages or detected only Type I clones.

Construct threats: Threats to construct validity are related to the classification of clone types. NiCad reports clones at the granularity of methods, even when only a portion of the code was added or modified. This can make manual validation and clone type classification potentially imprecise. To increase reliability, two researchers independently analyzed each sampled clone. In cases of disagreement regarding the validity or type of the clone, a third researcher was consulted to make the final decision. All researchers involved in this process have prior experience in code clone analysis.

External threats: The findings from this study are limited to only the four languages and their respective GitHub repositories and may not be generalized to other programming languages.

6 Ethical Implications

This study does not involve human subjects or direct interaction with individuals. All analyses were conducted exclusively on publicly available software repositories and pull requests, and no information regarding the identity, background, or personal characteristics of developers was collected, analyzed, or inferred at any stage of the research. Hence, this study did not require approval from an institutional ethics committee, as the collected data were used solely for research purposes and focused only on technical artifacts. Furthermore, the analyses were designed to avoid reputational harm or biased interpretations, and the results are reported in aggregate form rather than evaluating or judging individual developers or projects. By adhering to these principles, this work follows established ethical guidelines for Mining Software Repositories research.

7 Conclusion

This paper presented a study of the code clones lifecycle in pull request generated by coding agents. Using the AiDev dataset, we focused on 7,851 merged Agentic-PRs written in Python, C#, Java, and Ruby. By applying the Nicad clone detection tool, we identified 28,425 clone instances in 497 PRs.

The findings show that coding agents largely mirror human developers, with Type-I (29.0%) and Type-III (46.2%) clones dominating, and clones persisting across multiple commits being more likely to be merged. At the same time, although Type-II(24.8%) clones remain less frequent than Type-I and Type-III clones, their proportion is relatively higher in Agentic-PRs than what has been reported for human-authored code. This study underscores the need for clone-aware checks in AI-generated PR workflows, through clone detection in continuous integration pipelines or explicit review guidelines.

ACKNOWLEDGMENTS

This work received partial funding from CNPq-Brazil, Universal grant 404406/2023-8

References

[1] Qurat Ul Ain, Wasi Haider Butt, Muhammad Waseem Anwar, Farooque Azam, and Bilal Maqbool. 2019. A systematic review on code clone detection. *IEEE access* 7 (2019), 86121–86144.

[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

[3] Federico Cirett-Galán, Raquel Torres-Peralta, René Francisco Navarro Hernández, José Luis Ochoa Hernández, San Contreras-Rivera, Luis Arturo Estrada-Ríos, and Germán Machado-Encinas. 2024. Assessing the Use of GitHub Copilot on Students of Engineering of Information Systems. *International Journal of Software Engineering and Knowledge Engineering* 34, 11 (2024), 1717–1734. doi:10.1142/S0218194024500335

[4] James R Cordy and Chanchal K Roy. 2011. The NiCad clone detector. In *2011 IEEE 19th international conference on program comprehension*. IEEE, 219–220.

[5] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. 2013. Development and deployment at facebook. *IEEE Internet Computing* 17, 4 (2013), 8–17.

[6] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference (ACE '22)*. 10–19. doi:10.1145/3511861.3511863

[7] Armstrong Foundjem, Eleni Constantinou, Tom Mens, and Bram Adams. 2022. A mixed-methods analysis of micro-collaborative coding practices in OpenStack. *Empirical Software Engineering* (2022).

[8] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 345–355. doi:10.1145/2568225.2568260

[9] Kosei Horikawa, Hao Li, Yutaro Kashiwa, Bram Adams, Hajimu Iida, and Ahmed E. Hassan. 2025. Agentic Refactoring: An Empirical Study of AI Coding Agents. *arXiv preprint* (2025). https://arxiv.org/abs/2511.04824

[10] Yoichi Ishibashi and Yoshimasa Nishimura. 2024. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183* (2024).

[11] Judith F Islam, Manishankar Mondal, and Chanchal K Roy. 2016. Bug replication in code clones: An empirical study. In *2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER)*, Vol. 1. IEEE, 68–78.

[12] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, and Davide Taibi. 2021. Does code quality affect pull request acceptance? An empirical study. *Journal of Systems and Software* 171 (2021), 110806. doi:10.1016/j.jss.2020.110806

[13] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv preprint arXiv:2507.15003* (2025).

[14] Antonio Lima, Luca Rossi, and Mirco Musolesi. 2014. Coding Together at Scale: GitHub as a Collaborative Social Network. arXiv:1407.2535 [cs.SI] https://arxiv.org/abs/1407.2535

[15] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21 (2016), 2146–2189.

[16] Manishankar Mondal, Banani Roy, Chanchal K Roy, and Kevin A Schneider. 2019. Investigating context adaptation bugs in code clones. In *2019 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 157–168.

[17] Matheus Paixao and Paulo Maia. 2019. Rebasing in Code Review Considered Harmful: A Large-Scale Empirical Investigation. In *Proceedings of the 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 45–55. doi:10.1109/SCAM.2019.00014

[18] Matheus Paixão, Anderson Uchôa, Ana Carla Bibiano, Daniel Oliveira, Alessandro Garcia, Jens Krinke, and Emilio Arvonio. 2020. Behind the intents: An in-depth empirical study on software refactoring in modern code review. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 125–136.

[19] Luca Pascarella, Davide Spadini, Fabio Palomba, and Alberto Bacchelli. 2019. On the effect of code review on code smells. *arXiv preprint arXiv:1912.10098* (2019).

[20] Henry Pearce, Benjamin Tan, Baleegh Ahmad, Riad S. Wahby, and Shwetak N. Patel. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 754–768. doi:10.1109/SP46214.2022.9833576

[21] Chaiyong Ragkhitwetsagul and Jens Krinke. 2019. Siamese: scalable and incremental code clone search via multiple code representations. *Empirical Software Engineering* 24, 4 (2019), 2236–2284.

[22] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. 2019. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering* 47, 3 (2019), 560–581.

[23] Chanchal K Roy, Minhaz F Zibran, and Rainer Koschke. 2014. The vision of software clone management: Past, present, and future (keynote paper). In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 18–33.

[24] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*. 181–190.

[25] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, and Cristina V. Lopes. 2016. SourcererCC: scaling code clone detection to big-code. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. 1157–1168.

[26] Soily Ghosh Sneha, Sadia Niha, and Md. Manzurul Hasan. 2025. Evaluating Code Clone Detection and Management: A Comprehensive Comparison among Different Techniques and Tools along with Some Effective Future Directions. In *ICCA '24: Proceedings of the 3rd International Conference on Computing Advancements*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3723178.3723206

[27] Andrei Sobo, Awes Mubarak, Almas Baimagambetov, and Nikolaos Polatidis. 2025. Evaluating LLMs for Code Generation in HRI: A Comparative Study of ChatGPT, Gemini, and Claude. *Applied Artificial Intelligence* 39, 1 (2025), 2439610.

[28] Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K Roy, and Mohammad Mamun Mia. 2014. Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 476–480.

[29] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901* (2024).

[30] Anderson Uchôa, Caio Barbosa, Daniel Coutinho, Willian Oizumi, Wesley KG Assunçao, Silvia Regina Vergilio, Juliana Alves Pereira, Anderson Oliveira, and Alessandro Garcia. 2021. Predicting design impactful changes in modern code review: A large-scale empirical study. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 471–482.

[31] Anderson Uchôa, Caio Barbosa, Willian Oizumi, Publio Blenílio, Rafael Lima, Alessandro Garcia, and Carla Bezerra. 2020. How does modern code review impact software design degradation? an in-depth empirical study. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 511–522.

[32] Italo Uchoa, Denis Sousa, Henrique Donato, Matheus Paixao, Chaiyong Ragkhitwetsagul, and Thiago Lima. 2026. Replication Package for the paper: "A Study on Code Clone Lifecycles in Pull Requests Created by AI Agents". https://zenodo.org/records/18344200

[33] Italo Uchoa, Denis Sousa, Matheus Paixao, Pedro Maia, Anderson Uchôa, and Chaiyong Ragkhitwetsagul. 2025. An Exploratory Study on the Lifecycle of Code Clones During Code Review. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 293–303.

[34] Pranav Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. ACM, 1–15. doi:10.1145/3491102.3501879

[35] Tiantian Wang, Mark Harman, Yue Jia, and Jens Krinke. 2013. Searching for better configurations: a rigorous approach to clone evaluation. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 455–465.

[36] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

[37] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE '23)*. 172–178. doi:10.1145/3545945.3569830

[38] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A. Gerosa. 2022. Quality Gatekeepers: Investigating the Effects of Code Review Bots on Pull Request Activities. *Empirical Software Engineering* 27, 5 (2022), 108. doi:10.1007/s10664-022-10130-9

[39] Morteza Zakeri-Nasrabadi, Saeed Parsa, Mohammad Ramezani, Chanchal Roy, and Masoud Ekhtiarzadeh. 2023. A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges. *Journal of Systems and Software* 204 (10 2023), 111796. doi:10.1016/j.jss.2023.111796

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009