Human-Agent versus Human Pull Requests: A Testing-Focused Characterization and Comparison

Human-Agent versus Human Pull Requests: A Testing-Focused Characterization and Comparison

Roberto Milanese*
Department of Control and Computer Engineering
Politecnico di Torino
Turin, Italy
roberto.milanese@polito.it

Francesco Salzano
Department of Biosciences and Territory
University of Molise
Campobasso, Italy
francesco.salzano@unimol.it

Angelica Spina
Department of Biosciences and Territory
University of Molise
Campobasso, Italy
angelica.spina@unimol.it

Antonio Vitale*
Department of Control and Computer Engineering
Politecnico di Torino
Turin, Italy
antonio.vitale@polito.it

Remo Pareschi
Department of Biosciences and Territory
University of Molise
Campobasso, Italy
remo.pareschi@unimol.it

Fausto Fasano
Department of Biosciences and Territory
University of Molise
Campobasso, Italy
fausto.fasano@unimol.it

Mattia Fazzini
Department of Computer Science & Engineering
University of Minnesota
Minneapolis, USA
mfazzini@umn.edu

Abstract

AI-based coding agents are increasingly integrated into software development workflows, collaborating with developers to create pull requests (PRs). Despite their growing adoption, the role of human-agent collaboration in software testing remains poorly understood. This paper presents an empirical study of 6,582 human-agent PRs (HAPRs) and 3,122 human PRs (HPRs) from the AIDev dataset. We compare HAPRs and HPRs along three dimensions: (i) testing frequency and extent, (ii) types of testing-related changes (code-and-test co-evolution vs. test-focused), and (iii) testing quality, measured by test smells. Our findings reveal that, although the likelihood of including tests is comparable (42.9% for HAPRs vs. 40.0% for HPRs), HAPRs exhibit a larger extent of testing, nearly doubling the test-to-source line ratio found in HPRs. While test-focused task distributions are comparable, HAPRs are more likely to add new tests during co-evolution ($OR = 1.79$), whereas HPRs prioritize modifying existing tests. Finally, although some test smell categories differ statistically, negligible effect sizes suggest no meaningful differences in quality. These insights provide the first characterization of how human-agent collaboration shapes testing practices.

CCS Concepts

• **Software and its engineering → Software testing and analysis**; **Collaboration in software development**; *Empirical software engineering*; *Software evolution.*

Keywords

Software Testing, Mining Software Repositories, Empirical Study, Pull Requests, Test Smells, Coding Agents

ACM Reference Format:
Roberto Milanese, Francesco Salzano, Angelica Spina, Antonio Vitale, Remo Pareschi, Fausto Fasano, and Mattia Fazzini. 2026. Human-Agent versus Human Pull Requests: A Testing-Focused Characterization and Comparison. In *23rd International Conference on Mining Software Repositories (MSR '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3793302.3793617

1 Introduction

AI-based coding agents are transforming software engineering by enabling human-agent collaboration throughout the development lifecycle [3, 9, 10, 14, 26, 29]. These agents are increasingly integrated at the pull request (PR) level to perform complex tasks, including feature implementation, debugging, and testing [23]. As agents become core participants in development, characterizing their testing contributions is crucial for ensuring software quality and reliability. However, current research on agent-based testing often focuses on isolated code generation tasks or specific programming languages [2, 18], overlooking organic development contexts where production and test code co-evolve. While broader studies examine agent adoption across diverse development tasks [6, 25, 30],

*Also with University of Molise.

MSR '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2474-9/26/04
https://doi.org/10.1145/3793302.3793617

they rarely prioritize testing, leaving a fragmented understanding of how human-agent collaboration shapes testing practices.

To address this gap, we present a large-scale empirical study of software testing in PRs featuring collaboration between human developers and AI-based coding agents (HAPRs) versus PRs primarily contributed by human developers (HPRs). We structure our study around three Research Questions (RQs):

- **RQ$_1$**: *Are there differences between HAPRs and HPRs in the frequency and extent of testing contributions?*
- **RQ$_2$**: *How do the development contexts in which testing contributions appear compare between HAPRs and HPRs?*
- **RQ$_3$**: *How does test quality, as reflected by test smells, compare between HAPRs and HPRs?*

To answer these questions, we analyze 6,582 HAPRs and 3,122 HPRs from the AIDev dataset [23], spanning four programming languages (Java, JavaScript, Python, and TypeScript) and five major coding agents (Codex, Claude Code, Copilot, Devin, and Cursor). We characterize these PRs along three interconnected dimensions that provide a holistic view of human-agent collaborative testing. First, we evaluate testing frequency and extent to determine if agents contribute a volume of test code comparable to HPRs. Second, we examine the development contexts, distinguishing between co-evolution and test-focused tasks, to determine whether the collaboration is primarily focused on test additions or if it also extends to more complex maintenance tasks. Third, we assess testing quality via test smells to ensure that the collaborative effort does not introduce low-quality test code that could hinder long-term maintainability.

Our findings suggest that while human-agent testing practices are largely aligned with traditional workflows, the synergy between developers and agents enables more extensive testing contributions without a corresponding drop in quality. These results provide a foundation for developers to optimize agent integration and for researchers to explore the potential of human-agent collaborative testing. To support the validation and extension of our work, we provide a publicly available replication package [24].

2 Data Preparation

For our study, we used the AIDev dataset (version 3) [22], which provides 932,791 HAPRs and 6,618 HPRs from 116,211 GitHub repositories. Since the dataset includes HPRs only for repositories with at least 500 stars, we applied the same filter to HAPRs to ensure consistent comparison. We also excluded PRs without changes, leaving us with 12,319 HAPRs and 6,521 HPRs from 1,473 repositories.

Language Filtering. We narrowed our scope to four widely used programming languages: Java, JavaScript, Python, and TypeScript. These languages were selected for three reasons: (i) their popularity in industry [7] and open-source communities [15]; (ii) the availability of standard testing frameworks [11], which facilitates the identification of test files; and (iii) the availability of test smell detection tools to support our analysis. We used `unidiff` [5] to analyze the raw PR diffs retrieved from GitHub (e.g., [8]) and identify all changed files. For each PR, we retrieved file contents from the PR head commit or, in the case of removed files, from the parent of the deletion commit. We then used Linguist [13] to detect the programming language of each file. After this filtering step, our derived *dataset* included 6,582 HAPRs and 3,122 HPRs.

Table 1: Testing frequency and extent in HAPRs and HPRs.

Group	Count			Ratio			
	TPR	*TF*	*TLoC*	Setting	*TPR/PR*	*TF/F*	*TLoC/LoC*
HAPRs	2,821	9,574	1,158,302	*All PRs*	0.429	0.204	0.327
				Test PRs	-	0.301	0.400
HPRs	1,248	8,365	471,409	*All PRs*	0.400	0.194	0.185
				Test PRs	-	0.255	0.232

Test File Identification. We developed language-specific heuristics to distinguish between test and source code files. Unlike most studies [20, 32], which primarily check for the "test" keyword in filenames, Gonzalez et al. [16] proposed using content-specific constructs as more reliable indicators. Inspired by this, we manually analyzed a sample of 370 PRs, stratified by language (95% confidence level, CL, and 5% margin of error, ME), from our dataset to derive language-specific heuristics. Each PR was independently reviewed by two authors who are PhD students with at least 3 years of experience in software engineering tasks. They identified test markers based on three aspects: (i) *File Paths* to capture directory and naming conventions, (ii) *Import Statements* to identify testing frameworks imported in test files, and (iii) *Code Keywords* to consider syntactic markers such as annotations, methods, and assertions. We implemented the heuristics and evaluated them on a second stratified sample of 370 PRs. Two authors independently labeled it, achieving a Cohen's kappa of 0.81, indicating almost perfect agreement [21], while disagreements were resolved through discussion. The results show high accuracy in test file identification, with a precision of 0.988, a recall of 0.982, and an F-score of 0.985. Qualitative analysis revealed that the few misclassifications occurred in borderline cases. False positives involved source code used for testing other codebases (e.g., [4]), while false negatives were limited to custom tests that lacked standard patterns (e.g., [27]).

Final Dataset. This data preparation process led to a dataset [24] containing 6,582 HAPRs and 3,122 HPRs. *2,821 out of 6,582* HAPRs and *1,248 out of 3,122* HPRs contain test files. The total number of test files is 17,939, and the total number of changed files across all the PRs in the dataset is 64,542.

3 Empirical Study

In this section, we present the methodology and findings for our three research questions, exploring the frequency, context, and quality of testing contributions in HAPRs and HPRs.

3.1 RQ$_1$: Frequency and Extent of Testing

In RQ$_1$, we compare the contributions appearing in HAPRs and HPRs to assess whether human-agent collaboration affects the likelihood of including tests and the extent of testing-related changes.

Methodology. We organized our analysis across three levels: PRs, changed files, and changed lines of code. The PR level captures

testing frequency by assessing *how often* test code is included, while the file and line levels capture testing extent by measuring *how much* test code is included. To quantify these aspects, we defined two metrics: *Count* and *Ratio*. At the PR level, we measured the number of test PRs (TPR) and their ratio to all PRs (TPR/PR). At the file and line levels, we measured the number of test files (TF) and test lines of code (TLoC), along with their ratios to total changed files and lines (TF/F and TLoC/LoC). In the last two cases, we calculated the ratios against both all PRs (*All PRs* setting) and the subset of test PRs (*Test PRs* setting). This allowed us to understand the extent of testing in general and whether it differs between HAPRs and HPRs when tests are included. To assess statistical significance, we applied different tests based on the type of data. At the PR level, we used the chi-squared (χ^2) test at a significance level of $\alpha = 0.05$ [12] to compare the two groups. Then, we used the odds ratio (*OR*) to measure the effect size [1]. At the file and lines of code levels, we first computed the ratio of test files and test lines to the total number of changed files and lines for each PR. This resulted in a distribution of values for HAPRs and HPRs, rather than aggregate ratios. Since these distributions were non-normal, as confirmed by the Anderson-Darling test [34], we used the Mann-Whitney U test to compare them at a significance level of $\alpha = 0.05$. Then, we used Cliff's delta (δ) to measure the magnitude [19] and interpreted it according to the thresholds proposed by Vargha and Delaney [33].

Findings. Table 1 summarizes the results. In terms of testing frequency, HAPRs include test code slightly more often than HPRs (42.9% versus 40.0%). Although this difference is statistically significant ($p = 0.008$), the odds ratio ($OR = 1.13$) is close to one, suggesting that the difference is minimal. When considering the extent of testing, HAPRs demonstrate higher ratios at both the file (0.204 versus 0.194) and the line (0.327 versus 0.185) levels. These differences are statistically significant ($p < 0.001$), though the magnitude is negligible ($\delta_{file} = 0.054, \delta_{line} = 0.053$). In contrast, the differences become more evident in the *Test PRs* setting, where HAPRs demonstrate a greater extent of testing at both the file (0.301 versus 0.255) and the line (0.400 versus 0.232) levels. These differences are statistically significant ($p < 0.001$) with small effect sizes ($\delta_{file} = 0.147, \delta_{line} = 0.142$). While the line ratio is nearly double for HAPRs, similar Cliff's delta values indicate that the likelihood of an HAPR having a higher test ratio than an HPR remains consistent across the file and line levels.

Answer to RQ$_1$: *HAPRs and HPRs are similarly likely to include test-related changes, indicating comparable testing frequency. However, when tests are present, HAPRs tend to include a larger proportion of test-related files and lines of code, reflecting more extensive testing contributions than those observed in HPRs.*

3.2 RQ$_2$: Type and Context of Testing

In RQ$_2$, we examine the development contexts in which testing contributions appear to understand whether human-agent collaboration influences where testing work is concentrated within PRs.

Methodology. We manually analyzed a statistically significant stratified sample of 339 test HAPRs and 294 test HPRs (95% CL, 5% ME). We categorized test development contexts following Zaidman et al. [35]: the co-evolution context (*COC*) involves changes to both production and test code, while the test-focused context (*TFC*) includes changes to test code only. We also considered the test evolution tasks proposed by Pinto et al. [28] based on the type of change applied to test files: addition (*Add*), modification (*Mod*), and deletion (*Del*). Combining these contexts and tasks yields six distinct categories. Following a double-coding process, two authors independently reviewed and categorized each PR, assigning all applicable labels and resolving any disagreements through discussion. We use the *Count* and *Ratio* metrics to quantify how frequently the contributions in each category are. To assess the statistical significance, we used the chi-squared (χ^2) test [1] at a significance level of $\alpha = 0.05$ [12] to compare the two groups in each scenario. Since we performed multiple comparisons with the same sample, we adjusted the p-values using the Holm-Bonferroni correction [17]. Finally, we used the odds ratio (*OR*) to measure the effect size [1].

Table 2: Testing type metrics in HAPRs and HPRs under co-evolution (COC) and test-focused (TFC) contexts.

Group		*COC*				*TFC*			
		Add	*Mod*	*Del*	Total	*Add*	*Mod*	*Del*	Total
HAPRs	*Count*	232	70	3	269	31	43	3	69
HPRs	*Count*	161	128	14	237	14	47	1	59
HAPRs	*Ratio*	0.684	0.206	0.009	0.794	0.091	0.127	0.009	0.204
HPRs	*Ratio*	0.548	0.435	0.048	0.806	0.048	0.160	0.003	0.201

Findings. Table 2 summarizes the metrics. The distribution of testing contexts is remarkably consistent between HAPRs and HPRs. Specifically, test-and-code co-evolution is significantly more frequent than test-focused tasks, with an 80/20 distribution. However, specific testing tasks reveal a different trend. During co-evolution, HAPRs are more likely to include new tests (68.4% versus 54.8%). This difference is significant ($p = 0.003$), with a moderate effect size ($OR = 1.79$). Conversely, HPRs are significantly more involved in maintaining and refactoring existing tests (43.5% versus 20.6% for modifications and 4.8% versus 0.9% for deletions). Both differences are statistically significant ($p_{COCMod} < 0.001$, $p_{COCDel} = 0.023$), and the odds ratios indicate substantial and strong effects for modifications and deletions ($OR_{COCMod} = 0.34$, $OR_{COCDel} = 0.18$), respectively. In contrast, no statistically significant differences were observed in test-focused contexts ($p > 0.05$ for all tasks).

Answer to RQ$_2$: *Testing contributions in HAPRs and HPRs appear in similar development contexts, with both predominantly occurring in code-and-test co-evolution rather than test-focused settings. However, within co-evolutionary contexts, HAPRs more frequently introduce new tests, whereas HPRs more often involve the modification or deletion of existing test code.*

3.3 RQ$_3$: Test Smells

In RQ$_3$, we compare the testing quality of test code introduced in HAPRs and HPRs, as reflected by the presence of test smells, to assess if human-agent collaboration influences the quality of tests.

Table 3: Smell Delta (head vs. base) in HAPRs and HPRs.

Test Smell	Group	Mean Δ	Min Δ	Max Δ	StdDev
Assertion Roulette	HAPRs	8.043	-2353	1525	72.928
	HPRs	7.698	-330	1232	53.333
Conditional Test	HAPRs	1.846	-459	2686	52.996
	HPRs	0.534	-58	64	4.459
Duplicate Assert	HAPRs	0.388	-18	86	3.149
	HPRs	0.372	-60	91	4.784
Empty Test	HAPRs	-0.038	-168	33	3.309
	HPRs	0.002	-32	20	1.116
Exception Handling	HAPRs	0.504	-188	639	13.031
	HPRs	0.271	-12	29	1.771
Ignored Test	HAPRs	1.607	-917	467	23.124
	HPRs	0.153	-8	66	2.441
Magic Number Test	HAPRs	1.607	-917	467	23.124
	HPRs	1.650	-76	336	13.269
Redundant Print	HAPRs	0.062	-153	61	3.463
	HPRs	0.063	-7	9	0.623
Sleepy Test	HAPRs	0.076	-62	51	1.879
	HPRs	0.008	-13	10	0.682
Unknown Test	HAPRs	4.148	-301	7624	148.252
	HPRs	1.926	-4	9	2.306

Methodology. We analyzed the presence of test smells in both HAPRs and HPRs. For each test PR in our dataset, we analyzed test files in the states before and after each PR using AromaDr [31], a language-independent tool capable of detecting ten common test smells reported in Table 3. To quantify the impact of these changes, we introduced the *Smell Delta* metric, which measures the difference in the total count of test smells between the pre- and post-PR states. This allowed us to assess the extent to which agents and human authors affect the quality of the test code. To assess statistical significance, we compared the distributions of smell deltas for both groups using the Mann-Whitney U test [19] at a significance level of $\alpha = 0.05$ [12]. Finally, we used Cliff's delta (δ) to measure the magnitude of the difference [19] and interpreted it according to the thresholds proposed by Vargha and Delaney [33].

Findings. Table 3 summarizes the smell distribution across the considered categories for both HAPRs and HPRs. Across nearly all test smell categories, HAPRs show substantially larger standard deviations and more extreme minimum and maximum values compared to HPRs. This suggests that changes derived from human-agent collaboration tend to be less stable and more heterogeneous: while some HAPRs introduce few or no smells, others introduce or remove a very large number of smells. For several categories, most notably *Assertion Roulette* and *Magic Number Test*, both HAPRs and HPRs show positive mean deltas. This suggests that these smells are structural issues in test writing, rather than being specific to coding agents. When considering statistical significance, a few smell categories (*Assertion Roulette*, *Conditional Test*, *Magic Number Test* and *UnknownTest*) appear statistically different between HAPRs and HPRs; however, in all cases, the effect sizes are negligible, indicating that these differences are not practically meaningful.

Answer to RQ3: *Test quality, as reflected by test smells, is largely comparable between HAPRs and HPRs. Although a few test smell categories show statistically significant differences, the associated effect sizes are negligible, indicating no practically meaningful differences attributable to human-agent collaboration.*

4 Threats to Validity

Construct Validity. The heuristic-based identification of test files may result in misclassifications. We mitigate this risk by developing the heuristics through extensive manual analysis of a statistically significant sample of PRs.

Internal Validity. Observed differences between HAPRs and HPRs may be influenced by uncontrolled factors, such as developer expertise or the nature of the addressed issues. While these factors cannot be fully controlled in an observational study, this threat is partially mitigated by analyzing real-world PRs that reflect authentic software development practices rather than artificial settings.

External Validity. The findings of this study may not fully generalize to all programming languages or coding agents. We mitigate this threat by focusing on widely used programming languages and widely adopted coding agents, as well as on real-world GitHub projects with substantial community engagement.

5 Related Work

A few studies have focused on the quality of tests generated by coding assistants. Alves et al. [2] analyzed test smells in Python test code generated by Copilot, assessing the maintainability of generated tests independently of source code changes. Joshi and Band [18] explored the use of Copilot to generate tests for existing projects and compared the generated tests against manually written tests. While these studies provide valuable insights into the characteristics of agent-generated tests, they primarily consider testing as a standalone activity applied to already implemented functionality, rather than as part of an integrated development process.

Beyond test-specific studies, growing work explores coding agents across broader software development tasks. Sergeyuk et al. [30] examined how coding agents are adopted in practice, focusing on developer perceptions, usage patterns, and organizational implications. Nikolaidis et al. [25] compared the effectiveness of ChatGPT and Copilot in generating Python code, evaluating quality and correctness. Butler et al. [6] conducted a controlled trial to assess the impact of generative AI coding tools in real workplace settings, highlighting productivity and behavioral effects. These studies do not focus primarily on testing, nor do they analyze how testing is distributed or shaped in collaborative human-agent workflows.

6 Conclusion

This paper compares testing contributions in 6,582 human-agent PRs (HAPRs) and 3,122 human PRs (HPRs). Our analysis reveals that while both groups incorporate testing at similar rates, HAPRs nearly double the ratio of test code added. However, the nature of these contributions differs: HAPRs predominantly introduce new tests during code-and-test co-evolution, whereas HPRs focus more on modifying and maintaining existing test suites. Despite the significantly higher volume of test code in HAPRs, we find no meaningful difference in quality as measured by test smells.

These findings have several implications for the future of AI-assisted software engineering. First, they provide empirical evidence that human-agent synergy can significantly scale testing volume while maintaining a level of quality comparable to that of human-authored tests. Second, the systematic shift toward "addition over maintenance" in co-evolutionary contexts highlights a potential risk in collaboration workflows: a bias toward expanding test suites rather than evolving them. This suggests that while agents are effective for generating new tests, human developers must remain vigilant in directing agents toward refactoring and updating legacy tests to prevent suite bloating and ensure long-term maintainability. Future work should investigate whether human-agent collaboration would benefit from shifting emphasis toward maintaining and evolving existing tests rather than primarily adding new tests. In addition, future studies should assess the effectiveness of tests produced in HAPRs using test coverage and fault-detection capability. Together, these directions can help refine the role of coding agents and inform best practices for AI-assisted software development.

Acknowledgments

This publication is part of the project PNRR-NGEU, which has received funding from the MUR – DM 118/2023.

References

[1] Alan Agresti. 2006. *An Introduction to Categorical Data Analysis* (2nd ed.). Wiley. doi:10.1002/0470114754

[2] Victor Anthony Alves, Cristiano Santos, Carla Bezerra, and Ivan Machado. 2024. Detecting Test Smells in Python Test Code Generated by LLM: An Empirical Study with GitHub Copilot. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 581–587.

[3] Anthropic. 2025. *Claude Code*. Retrieved December 23, 2025 from https://www.claude.com/product/claude-code

[4] benfdking. 2025. *Implement vscode test controller*. Retrieved December 23, 2025 from https://github.com/TobikoData/sqlmesh/pull/5042

[5] Matias Bordese. 2023. *Unidiff version 0.7.5*. https://github.com/matiasb/python-unidiff

[6] Jenna Butler, Jina Suh, Sankeerti Haniyur, and Constance Hadley. 2025. Dear Diary: A randomized controlled trial of Generative AI coding tools in the workplace. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 319–329.

[7] Stephen Cass. 2025. *The Top Programming Languages 2025*. Retrieved December 20, 2025 from https://spectrum.ieee.org/top-programming-languages-2025

[8] claudecodebyia. 2025. *Fix Discord presence module error handling (Fixes #5535)*. Retrieved December 23, 2025 from https://patch-diff.githubusercontent.com/raw/MeteorDevelopment/meteor-client/pull/5568.diff

[9] Cognition. 2025. *Devin*. Retrieved December 23, 2025 from https://devin.ai

[10] Cursor. 2025. *Cursor*. Retrieved December 23, 2025 from https://cursor.com

[11] Rômulo Martins da Silva, Cafer Cruz, Heleno de S. Campos, Leonardo G. P. Murta, and Vânia de Oliveira Neves. 2019. What is the adoption level of automated support for testing in open-source ecosystems?. In *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing* (Salvador, Brazil) *(SAST '19)*. Association for Computing Machinery, New York, NY, USA, 80–89. doi:10.1145/3356317.3356325

[12] R. A. Fisher. 1992. *Statistical Methods for Research Workers*. Springer New York, New York, NY, 66–70. doi:10.1007/978-1-4612-4380-9_6

[13] GitHub. 2025. *Linguist version 9.2.0*. https://github.com/github-linguist/linguist

[14] GitHub. 2025. *Copilot*. Retrieved December 23, 2025 from https://github.com/features/copilot

[15] GitHub. 2025. *Octoverse: A new developer joins GitHub every second as AI leads TypeScript to #1*. Retrieved December 20, 2025 from https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/

[16] Danielle Gonzalez, Joanna C.S. Santos, Andrew Popovich, Mehdi Mirakhorli, and Mei Nagappan. 2017. A Large-Scale Study on the Usage of Testing Patterns That Address Maintainability Attributes: Patterns for Ease of Modification, Diagnoses, and Comprehension. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 391–401. doi:10.1109/MSR.2017.8

[17] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. http://www.jstor.org/stable/4615733

[18] Vijay Joshi and Iver Band. 2025. Disrupting test development with ai assistants. In *2025 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 421–425.

[19] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (2017), 579–630. doi:10.1007/s10664-016-9437-5

[20] Pavneet Singh Kochhar, Tegawendé F. Bissyandé, David Lo, and Lingxiao Jiang. 2013. An Empirical Study of Adoption of Software Testing in Open Source Projects. In *2013 13th International Conference on Quality Software*. 103–112. doi:10.1109/QSIC.2013.57

[21] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. doi:10.2307/2529310

[22] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. *AIDev: Studying AI Coding Agents on GitHub*. doi:10.5281/zenodo.16919272

[23] 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).

[24] Roberto Milanese, Francesco Salzano, Angelica Spina, Antonio Vitale, Remo Pareschi, Fausto Fasano, and Mattia Fazzini. 2025. *Replication Package for Human-Agent versus Human Pull Requests: A Testing-Focused Characterization and Comparison*. doi:10.5281/zenodo.18098186

[25] Nikolaos Nikolaidis, Karolos Flamos, Khanak Gulati, Daniel Feitosa, Apostolos Ampatzoglou, and Alexander Chatzigeorgiou. 2024. A Comparison of the Effectiveness of ChatGPT and Co-Pilot for Generating Quality Python Code Solutions. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering-Companion (SANER-C)*. IEEE, 93–101.

[26] OpenAI. 2025. *Codex*. Retrieved December 23, 2025 from https://openai.com/codex

[27] pelikhan and Copilot. 2025. *Add mdtranslator script with comprehensive link validation QA step*. Retrieved December 23, 2025 from https://github.com/microsoft/genaiscript/pull/1681

[28] Leandro Sales Pinto, Saurabh Sinha, and Alessandro Orso. 2012. Understanding myths and realities of test-suite evolution. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (Cary, North Carolina) *(FSE '12)*. Association for Computing Machinery, New York, NY, USA, Article 33, 11 pages. doi:10.1145/2393596.2393634

[29] Ketai Qiu, Niccolò Puccinelli, Matteo Ciniselli, and Luca Di Grazia. 2025. From today's code to tomorrow's symphony: The AI transformation of developer's routine by 2030. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–17.

[30] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. 2025. Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology* 178 (2025), 107610.

[31] Publio Silva, Carla Bezerra, Ivan Machado, and Márcio Ribeiro. 2025. AromaDr: A Language-Independent Tool for Detecting Test Smells. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 914–920.

[32] Fabian Trautsch and Jens Grabowski. 2017. Are There Any Unit Tests? An Empirical Study on Unit Testing in Open Source Python Projects. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 207–218. doi:10.1109/ICST.2017.26

[33] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. doi:10.3102/10769986025002101

[34] Berna Yazici and Senay Yolacan. 2007. A comparison of various tests of normality. *Journal of Statistical Computation and Simulation* 77, 2 (2007), 175–183. doi:10.1080/10629360600678310

[35] Andy Zaidman, Bart Van Rompaey, Arie van Deursen, and Serge Demeyer. 2011. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering* 16, 3 (01 06 2011), 325–364.