From Correctness to Collaboration: A Human-Centered Taxonomy of AI Agent Behavior in Software Engineering

Tao Dong
Google
Sunnyvale, CA, USA
taodong@google.com

Sherry Shi
Google
Kirkland, WA, USA
sherryyshi@google.com

Harini Sampath
Google
Kirkland, WA, USA
harinis@google.com

Andrew Macvean
Google
Kirkland, WA, USA
amacvean@google.com

Abstract

The ongoing transition of Large Language Models in software engineering from code generators into autonomous agents requires a shift in how we define and measure success. While models are becoming more capable, the industry lacks a clear understanding of the behavioral norms that make an agent effective in collaborative software development in the enterprise. This work addresses this gap by presenting a taxonomy of desirable agent behaviors, synthesized from 91 sets of developer-defined rules for coding agents. We identify four core expectations: *Adhere to Standards and Processes*, *Ensure Code Quality and Reliability*, *Solve Problems Effectively*, and *Collaborate with the Developer*. These findings offer a concrete vocabulary for agent behavior, enabling researchers to move beyond correctness-only benchmarks and start designing evaluations that reflect the socio-technical nature of professional software development in enterprises.

CCS Concepts

• **Human-centered computing → Empirical studies in HCI**.

Keywords

Human-AI Collaboration, AI Agents, LLM Evaluation, Software Engineering

ACM Reference Format:
Tao Dong, Sherry Shi, Harini Sampath, and Andrew Macvean. 2026. From Correctness to Collaboration: A Human-Centered Taxonomy of AI Agent Behavior in Software Engineering. In *Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems (CHI EA '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3772363.3798733

1 Introduction

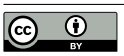
Powered by recent advancements in Large Language Models (LLMs), the nature of Generative AI tools for software engineering has been undergoing a rapid transformation over the past few years, shifting from passive code completers [21] to increasingly autonomous agents capable of tackling complex tasks including testing [10], migrations [17], bug fixing [15, 19] , and more. This evolution marks a critical paradigm shift: from AI as a tool to be wielded by a developer to AI as a collaborative partner integrated into a software team's workflow [11, 13].

As agents assume this role of "partner," the definition of success becomes increasingly ambiguous. While current models are becoming more capable of generating correct code, the industry lacks a clear understanding of the behavioral norms that make an agent an effective contributor within a software development team. Unlike a simple tool, an autonomous partner must demonstrate sound judgment, adhere to team values and norms, and communicate effectively—qualities that are often implicit and difficult to define.

This ambiguity creates a significant gap in how we measure progress. The field has been heavily guided by benchmarks like SWE-bench [8] and LiveCodeBench [6], which define success primarily in terms of functional correctness. By fixating on the final code output, they are fundamentally unequipped to assess the collaborative behaviors and process adherence that enables software development in a team setting.

To address this challenge and move beyond a monolithic view of agent performance, we ask the overarching research question: *What are the core behaviors an intelligent agent should be evaluated for in software engineering?*

To answer this question, we derived a taxonomy of four core behavioral expectations and 15 specific attributes through a qualitative analysis of 91 real-world agent rule sets deployed within Google, a large technology company. Agent rules are often defined in project-level configuration files[1], such as `CLAUDE.md` or `GEMINI.md`, that allow developers to provide custom instructions and behavioral guidelines to steer how an AI agent approach tasks within a given codebase. By characterizing these behavioral expectations of AI agents as partners, we provide the missing vocabulary needed to close the current evaluation gap, enabling the field to move beyond functional correctness and start assessing an agent's capacity for effective teamwork.



CHI EA '26, Barcelona, Spain

ACM ISBN 979-8-4007-2281-3/26/04
https://doi.org/10.1145/3772363.3798733

[1] An introduction to agent rule files: https://cursor.com/docs/context/rules.

2 Related Work

The optimization targets of AI in software engineering have been heavily expressed in and codified by benchmarks. Early benchmarks such as HumanEval [4] and MBPP [1] established functional correctness as the primary metric, treating the agent as a black-box code generator. Success was measured by pass@k rates on unit tests, a suitable method for evaluating single-shot code completion. As tasks grew more complex, benchmarks like SWE-Bench [8] and SWE-Lancer [16] maintained this focus on final outcomes, albeit for more challenging, multi-file problems. Most recently, with the rise of autonomous agents in coding, the research focus has shifted from the final output to the process of generating it. Frameworks like MAST [18] and TRAIL [5] provide valuable perspectives and common failure modes to look out for.

One important capability of coding agents that many existing evaluation techniques do not assess is the quality of the agent's interactions with the developer. In contrast, in domains such as medicine and education, human-centric metrics have been adopted when assessing LLMs' capabilities. For instance, studies in medicine evaluate AI not just for diagnostic accuracy but also for the quality and empathy of its communication with patients [2, 12]. Similarly, in education, AI tutors are assessed on their ability to provide scaffolding and encouragement, not just correct answers [9, 14, 20].

Within the AI for SWE domain, researchers are beginning to acknowledge this need. In an observational study, Kumar et al. [11] found a direct correlation between successful task completion and the amount of human-agent communication, highlighting the importance of interaction quality. However, the specific characteristics that constitute high-quality human-AI interaction within this domain remain unclear.

One source of developers' expectations for agent behavior is agent rules, a mechanism for developers to provide project-level context and directives that has recently seen growing adoption across the industry. In a study by Jiang et al. [7], 36% of repos studied have agent rule files that contain behavior-related guidelines. Similarly, in Chatlatanagulchai et al.'s study, 24.4% of agent context files contained specific instructions on the desired behavior and roles of agentic coding, and methods for integrating other AI tools [3]. While these studies take a descriptive approach to analyzing the content of agent rules in open source software projects, this work leverages data from one of the largest enterprise codebases, and focuses primarily on the expected behaviors expressed in these rules to establish a normative taxonomy for the evaluation of agent behavior.

3 Methods

To create a taxonomy of desirable agent behaviors in enterprise software engineering, we analyzed 91 project-level rule files for coding agents at Google, a global technology company. Collected in late July 2025, about a month after support for agent rules was introduced to software developers at the company, this dataset represents perspectives from early adopters of agent rules in this organization.

Two of the authors performed iterative open coding on 15 rule files, which led to a codebook consisting of 15 behavioral attributes grouped by four themes. To scale the analysis, we employed an LLM-based annotator to code the remaining corpus. We instructed the model to first segment each file into a set of coherent rules and then annotate each rule with up to three codes from the codebook. On a validation set of 5 files (95 rules), the annotator achieved 94.4% precision and 91.2% recall with zero hallucinations, a performance we deemed acceptable for our qualitative analysis. The prompt template for the annotator and the full codebook are available in the supplemental material.

4 Findings

Our analysis of the agent rules defined by software engineers reveals four core expectations and 15 specific behaviors for AI agents, summarized in Table 1. In the rest of this section, we describe these expectations in detail. To protect proprietary information, we replaced specific instances of such with descriptive labels enclosed in angle brackets.

4.1 Adhering to Standards and Processes

A predominant expectation was that the agent must strictly adhere to established standards and project-specific processes. This expectation highlights the "brownfield" nature of software development in a large enterprise where AI-generated code is expected to integrate seamlessly into a mature, existing codebase.

Developers frequently instructed the agent to "follow," "adhere to," "consult," and "use" pre-existing, widely-accepted guidelines for specific programming languages and frameworks within the company. These included documents like "CSS Best Practices," "efficient java guide," "Go Style," and the "TypeScript style guide." The specificity of these rules can vary across projects. While some rules simply link to online documentation, others underscore key steps directly in the rules file itself, indicating differing degrees of confidence in the agent's capacity to identify and implement relevant guidelines.

Beyond general best practices, developers specified a multitude of *project-specific* procedures and conventions. These rules governed local practices such as dependency management, interaction with external systems, test generation and execution, API usage patterns, naming conventions, and code organization. By providing these rules, developers effectively mandated an "orientation" for the AI agent to ensure the agent's contributions are not only functional but also conformant.

4.2 Ensure Code Quality and Reliability

Our analysis of agent rules highlights developers' deep concern for code quality, extending far beyond functional correctness. Developers instructed the agent to produce code consistent in style, maintainable, reliable, and performant.

First, a significant portion of the rules centered on enforcing stylistic consistency, a critical factor for maintaining a large-scale, multi-author codebase. These instructions fell into three categories:

- **Guideline Application:** Developers frequently directed the agent to follow links of official style guides. These references were often supplemented with specific rules for naming conventions, import ordering, or syntax.

Table 1: Human-Centered Taxonomy of AI Agent Behavior: Core Expectations, Behaviors, and Example Rules. % Files represents the percentage of files in our dataset that contained rules pertaining to each behavior.

Expectation	Behavior	Example Agent Rule from Corpus	% Files
1. Adhere to Standards and Processes	Following Established Best Practices	"Follow the [Angular Style Guide](<URL>) for more details"	45.05
	Following Project Workflows and Conventions	"After creating your new `_test.go` file, add a `go_test` target to the `test/<package>` file."	92.31
2. Ensure Code Quality and Reliability	Maintain Code Style	"Follow Local Style: Adhere to existing patterns and naming conventions... This takes precedence over general style guides."	57.14
	Write Readable and Maintainable Code	"Write for Others: Code should be written with the assumption that someone else will read, understand, and maintain it."	36.26
	Build Robust and Performant Software	"Unsubscribe from observables using `take(1)` or other lifecycle-aware operators to prevent memory leaks."	46.15
3. Solve Problems Effectively	Understand Project Context before Acting	"Before creating the first SQL query, *you must* read the '.proto' files linked from the `<path>/sql.md` to understand the fields that are very likely to come up in queries."	90.11
	Work Incrementally and Iteratively	"Incremental Changes: Make the smallest possible code change, then run tests. Fix failures before making further changes."	11.00
	Validate Work Proactively	"After finishing any changes, always run our unit tests by running `<command>` and iterating until these tests pass."	37.36
	Maintain Task Focus	"Defer Unrelated Tasks: If you identify a necessary but out-of-scope task (e.g., a needed data model refactoring), leave a TODO comment."	9.89
	Infer Intent from Context	"If the user does not write any test description, the AI agent should still try inferencing the test case from the name."	19.78
	Learn by Example	"Inspect other tools in the `tools/` dir and copy their approaches."	32.97
4. Collaborate with the Developer	Communicate Effectively	"STRICTLY FORBIDDEN from starting messages with 'Great' or 'Certainly'... It is important you be clear and technical in your messages."	30.77
	Seek Help and Clarification	"You are allowed to ask the user questions. Be sure to use a clear and concise question that will help you move forward with the task."	25.27
	Plan Collaboratively and Analyze Trade-offs	"Critically evaluate all requests. If a prompt is ambiguous, please challenge it and propose a better alternative, explaining the trade-offs..."	8.79
	Learn from Feedback and Past Experiences	"At the end of every task, or upon making an error, you MUST update the `lessons_learned.md` file...This process is essential for self-correction and knowledge retention."	14.29

- **Tool Execution:** Adherence was often automated by instructing the agent to execute standard formatting and linting tools as part of its workflow.
- **Contextual Adaptation:** Some developers expected the agent to infer and adopt the style of the surrounding code and instructed the agent to prioritize local conventions over global ones when they are in conflict.

Second, developers emphasized that AI-generated code should be easy for humans to understand, modify, and debug. This expectation frames code not as a set of instructions for a machine, but as a form of communication within a development team. Specifically, developers provided several types of guidance about code maintainability:

- **High-Level Principles:** Adherence to established software engineering philosophies like DRY ("Don't Repeat Yourself"), "Loose Coupling," and the "Boy Scout Rule" (leave the code cleaner than you found it).
- **Structural Organization:** Instructions to organize code into clearly-defined modules, manage dependencies correctly, and reuse existing utilities rather than creating redundant logic.
- **Code-Level Simplification:** Directives to write simple, clear code by avoiding deep nesting, simplifying complex conditions, and keeping functions short and focused.
- **Naming and Documentation:** A strong focus on clear, descriptive names for variables and methods, as well as the creation and maintenance of comments and docstrings to provide context for future developers.

Finally, developers instructed the agent to produce code that was resilient, efficient, and well-tested. They expected the agent to proactively avoid common pitfalls (e.g., unsafe memory operations).

These rules about code quality communicate a clear developer preference that the efficiency gain from agentic coding should not come at the expense of the long-term health and coherence of the codebase.

4.3 Solve Problems Effectively

Beyond adhering to institutional standards and ensuring code quality, developers actively seek to instill effective problem-solving heuristics in the agent through agent rules. These rules represent a form of procedural knowledge, teaching the agent not just *what* to do, but *how* to approach its tasks like an experienced engineer in the following aspects:

First, the agent should gather and internalize project-specific information before making changes. This involves reading documentation and examining existing code to understand the project's architecture, tech stack, and design patterns.

Second, when approaching a large task, the agent should methodically break it down into small, manageable steps. Directives such as "one build configuration file at a time," "small, single-purpose PRs," and making the "smallest possible code change" were common among agent rules.

Third, developers mandated that the agent take responsibility for the quality of its own output before presenting it to them. The agent was commanded to perform a range of validation actions, including building the project, checking code style, and running tests.

Fourth, to prevent unintended side effects or scope creep, developers imposed strict constraints on the agent's actions. Using imperative phrases like "Only accept," "DO NOT engage," and "Leave... alone," developers created clear boundaries, demanding that the agent remain laser-focused on the specific task at hand.

Fifth, in contrast to the need for strict focus, some developers granted the agent a degree of autonomy to resolve ambiguity, especially in low-stakes situations. Rather than halting and asking for clarification, the agent was sometimes expected to use its reasoning capabilities to infer the developer's intent from the available context.

Finally, related to inferring intent from context, the agent was also expected to reason about the best course of action from examples in the codebase.

4.4 Collaborate with the Developer

Developer defined protocols, through agent rules, for a productive human-agent partnership, focusing on how the agent should communicate, manage uncertainty, formulate plans, and learn over time. Four key collaborative behaviors emerged from our analysis:

First, developers required communication that was precise, transparent, and efficient, prioritizing substantive information over conversational pleasantries. This technical communication style aimed to ensure the precision of information, maintain the developer's situational awareness, and facilitate exception management.

Second, the agent was expected to recognize the limits of its own knowledge and capabilities. Under conditions of ambiguity, high-stakes actions, or environmental blockers, it was instructed to pause and defer to the developer. Critically, developers also wanted these interruptions to be low-friction, often by having the agent provide suggested answers to make it easy for the developer to provide guidance.

Third, before executing complex tasks, developers required the agent to engage in collaborative planning. This involved creating a plan, evaluating the pros and cons of different approaches, and presenting recommendations for approval. In some cases, the agent was even expected to exercise critical thinking, challenge suboptimal initial requests, and propose better alternatives with clear justification.

Last, developers sought to overcome the stateless nature of LLMs by instructing the agent to learn from its mistakes and experiences. While many rules implicitly aimed for this through negative constraints (e.g., "never ever do ..."), some developers attempted to arrange an explicit mechanism for knowledge retention, such as having the agent maintain a "lessons_learned.md" document.

The above collaborative protocols defined in agent rules suggest a sophisticated interplay of developer needs. First, the demands for planning, transparency, and seeking help are primarily risk-mitigation strategies. Developers are concerned about agents performing destructive or incorrect actions without oversight and use these human-in-the-loop checkpoints to maintain control.

Second, efficient communication and the ability to learn from feedback are aimed at making the agent easier to manage. Conversational fluff is perceived as noise that slows down human guidance, while having to repeat corrections is a source of frustration.

Last, beyond simply preventing errors, some rules aim to leverage the agent's reasoning capabilities. By encouraging the agent to critically examine requests, analyze trade-offs, and identify ambiguity, developers are prompting the agent to exercise its "agency," though often within specified limits.

5 Conclusions and Future Work

To sum up, our work addresses a critical evaluation gap in the evolving landscape of AI agents for software engineering. Our main contribution is a taxonomy of desirable AI agent behaviors for enterprise software engineering, derived from a qualitative analysis of 91 sets of agent rules. The taxonomy defines four key expectations of agent behavior: *Adhere to Standards and Processes*, *Ensure Code Quality and Reliability*, *Solve Problems Effectively*, and *Collaborate with the Developer*, providing a systematic vocabulary for setting goals on effective human-AI partnership.

To keep pace with the rapid advancement of AI agents and evolving developer practices, our future work focuses on three key areas. First, we aim to build a robust, automated pipeline to significantly expand the agent rules dataset and perform periodic re-analysis. Second, a larger dataset will enable more granular analysis to identify variations in expected agent behavior across diverse problem domains, technology stacks, and codebase maturity levels. Finally, we are developing behavioral evaluations for specific taxonomy attributes, beginning with those related to human-AI communication in software development.

6 GenAI Usage Disclosure

GenAI tools were utilized for copy editing and data analysis, as described in the Methods section.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021). https://arxiv.org/abs/2108.07732

[2] John W Ayers, Adam Poliak, Mark Dredze, Eric C Leas, Zechariah Zhu, Jessica B Kelley, Dennis J Faix, Aaron M Goodman, Christopher A Longhurst, Michael Hogarth, et al. 2023. Comparing physician and artificial intelligence chatbot responses to patient questions posted to a public social media forum. *JAMA Internal Medicine* 183, 6 (2023), 589–596. doi:10.1001/jamainternmed.2023.1838

[3] Worawalan Chatlatanagulchai, Hao Li, Yutaro Kashiwa, Brittany Reid, Kundjanasith Thonglek, Pattara Leelaprute, Arnon Rungsawang, Bundit Manaskasemsak, Bram Adams, Ahmed E Hassan, et al. 2025. Agent READMEs: An Empirical Study of Context Files for Agentic Coding. *arXiv preprint arXiv:2511.12884* (2025).

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique P. de O. Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021). https://arxiv.org/abs/2107.03374

[5] Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. 2025. TRAIL: Trace Reasoning and Agentic Issue Localization. *arXiv preprint arXiv:2505.08638* (2025). https://arxiv.org/abs/2505.08638

[6] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*. https://openreview.net/forum?id=chfJJYC3iL original arXiv:2403.07974.

[7] Shaokang Jiang and Daye Nam. 2025. An Empirical Study of Developer-Provided Context for AI Coding Assistants in Open-Source Projects. *arXiv preprint arXiv:2512.18925* (2025).

[8] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world GitHub Issues?. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*. https://openreview.net/forum?id=VTF8yNQM66 original arXiv:2310.06770.

[9] Irina Jurenka, Markus Kunesch, Kevin R. McKee, Daniel Gillick, Shaojian Zhu, Sara Wiltberger, Shubham Milind Phal, Katherine Hermann, Daniel Kasenberg, Avishkar Bhoopchand, Ankit Anand, Miruna Pîslar, Stephanie Chan, Lisa Wang, Jennifer She, Parsa Mahmoudieh, Aliya Rysbek, Wei-Jen Ko, Andrea Huber, Brett Wiltshire, Gal Elidan, Roni Rabin, Jasmin Rubinovitz, Amit Pitaru, Mac McAllister, Julia Wilkowski, David Choi, Roee Engelberg, Lidan Hackmon, Adva Levin, Rachel Griffin, Michael Sears, Filip Bar, Mia Mesar, Mana Jabbour, Arslan Chaudhry, James Cohan, Sridhar Thiagarajan, Nir Levine, Ben Brown, Dilan Gorur, Svetlana Grant, Rachel Hashimshoni, Laura Weidinger, Jieru Hu, Dawn Chen, Kuba Dolecki, Canfer Akbulut, Maxwell Bileschi, Laura Culp, Wen-Xin Dong, Nahema Marchal, Kelsie Van Deman, Hema Bajaj Misra, Michael Duah, Moran Ambar, Avi Caciularu, Sandra Lefdal, Chris Summerfield, James An, Pierre-Alexandre Kamienny, Abhinit Mohdi, Theofilos Strinopoulous, Annie Hale, Wayne Anderson, Luis C. Cobo, Niv Efron, Muktha Ananda, Shakir Mohamed, Maureen Heymans, Zoubin Ghahramani, Yossi Matias, Ben Gomes, and Lila Ibrahim. 2024. Towards Responsible Development of Generative AI for Education: An Evaluation-Driven Approach. arXiv:2407.12687 [cs.CY] https://arxiv.org/abs/2407.12687

[10] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring LLM-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2312–2323. doi:10.1109/ICSE48619.2023.00194

[11] Aayush Kumar, Yasharth Bajpai, Sumit Gulwani, Gustavo Soares, and Emerson Murphy-Hill. 2025. Sharp Tools: How Developers Wield Agentic AI in Real Software Engineering Tasks. *arXiv e-prints arXiv:2506.12347* (2025). https://arxiv.org/abs/2506.12347 Microsoft Research publication / arXiv — no separate peer-reviewed venue found.

[12] Yahan Li, Jifan Yao, John Bosco S Bunyi, Adam C Frank, Angel Hwang, and Ruishan Liu. 2025. CounselBench: A Large-Scale Expert Evaluation and Adversarial Benchmark of Large Language Models in Mental Health Counseling. *arXiv preprint arXiv:2506.08584* (2025). https://arxiv.org/abs/2506.08584

[13] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large language model-based agents for software engineering: A survey. *arXiv preprint arXiv:2409.02977* (2024). https://arxiv.org/abs/2409.02977

[14] Kaushal Kumar Maurya, KV Srivatsa, Kseniia Petukhova, and Ekaterina Kochmar. 2024. Unifying AI tutor evaluation: An evaluation taxonomy for pedagogical ability assessment of LLM-powered AI tutors. *arXiv preprint arXiv:2412.09416* (2024). https://arxiv.org/abs/2412.09416

[15] Xiangxin Meng, Zexiong Ma, Pengfei Gao, and Chao Peng. 2024. An empirical study on llm-based agents for automated bug fixing. *arXiv preprint arXiv:2411.10213* (2024). https://arxiv.org/abs/2411.10213

[16] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. 2025. SWE-Lancer: Can Frontier LLMs Earn $1 Million from Real-World Freelance Software Engineering?. In *Proceedings of the 2025 International Conference on Machine Learning (ICML 2025)*. https://arxiv.org/abs/2502.12115 ICML 2025 (oral); arXiv:2502.12115.

[17] Stoyan Nikolov, Daniele Codecasa, Anna Sjövall, Maxim Tabachnyk, Satish Chandra, Siddharth Taneja, and Celal Ziftci. 2025. How is Google using AI for internal code migrations?. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. https://arxiv.org/abs/2501.06972 conference paper; arXiv version available.

[18] Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. 2025. Why Do Multiagent Systems Fail?. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*. https://openreview.net/forum?id=wM521FqPvI

[19] Pat Rondon, Renyao Wei, José Cambronero, Jürgen Cito, Aaron Sun, Siddhant Sanyam, Michele Tufano, and Satish Chandra. 2025. Evaluating agent-based program repair at Google. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. doi:10.1109/ICSE-SEIP66354.2025.00038 arXiv preprint also available; accepted/presented at ICSE-SEIP 2025..

[20] Yao Shi, Rongkeng Liang, and Yong Xu. 2025. EducationQ: Evaluating LLMs' Teaching Capabilities Through Multi-Agent Dialogue Framework. *arXiv preprint arXiv:2504.14928* (2025). https://arxiv.org/abs/2504.14928

[21] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–7. doi:10.1145/3491101.3519665