Support, not automation

towards AI-supported code review for code quality and beyond

Heander, Lo; Söderberg, Emma; Rydenfält, Christofer

Published in:
33rd ACM International Conference on the Foundations of Software Engineering

2025

Document Version:
Peer reviewed version (aka post-print)

Support, Not Automation: Towards AI-supported Code Review for Code Quality and Beyond

Lo Heander
lo.heander@cs.lth.se
Lund University
Lund, Sweden

Emma Söderberg
emma.soderberg@cs.lth.se
Lund University
Lund, Sweden

Christofer Rydenfält
christofer.rydenfalt@design.lth.se
Lund University
Lund, Sweden

Abstract

Code review is a well-established and valuable software development practice associated with code quality, interpersonal, and team benefits. However, it is also time-consuming, with developers spending 10–20% of their working time doing code reviews. With recent advances in AI capabilities, there are more and more initiatives aimed at fully automating code reviews to save time and streamline software developer workflows.

However, while automated tools might succeed in maintaining the code quality, we risk losing interpersonal and team benefits such as knowledge transfer, shared code ownership, and team awareness. Instead of automating code review and losing these important benefits, we envision a code review platform where AI is used to *support* code review to increase benefits for both code quality and the development team.

We propose an AI agent-based architecture that collects and combines information to support the user throughout the code review and adapt the workflow to their needs. We analyze this design in relation to the benefits of code review and outline a research agenda aimed at realizing the proposed design.

CCS Concepts

• **Human-centered computing** → *Human computer interaction (HCI)*; • **Software and its engineering** → **Software verification and validation**; **Software notations and tools**; • **Computing methodologies** → **Multi-agent systems**.

Keywords

code review, agentic systems, human-in-the-loop, code review tools, multi-agent systems, large language models

ACM Reference Format:
Lo Heander, Emma Söderberg, and Christofer Rydenfält. 2025. Support, Not Automation: Towards AI-supported Code Review for Code Quality and Beyond. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728505

1 Introduction

Code review is a valued practice in the software industry. The practice, originally introduced for quality improvement in the 1980s [1], is today valued for a number of properties beyond code quality. Bacchelli and Bird [3] report that developers' motivation for code review is, in order: defect finding, code improvement, alternative solutions, knowledge transfer, team awareness, improve the developer process, share code ownership, avoid build breaks, track rationale, and team assessment. Notably, at least half of these motivations are not directly about code quality but about user needs or interpersonal benefits. Thus, code review is clearly an important source of learning and education within a team.

Although code review is valued, it is also a time-consuming practice. Software developers have been reported to spend between 10-20% of their working time doing code reviews [4, 29]; with an estimated 28 million software developers during 2024 [32] this corresponds to 22-44 million hours every work day. The 2023 DORA State of DevOps report [12], focused on industry best practices, reports that optimizing code reviews is a key factor in overall developer team productivity. There is a need to continue to develop code review and its tools to improve the practice.

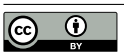
With more and more capable AI models available, there is an increased interest in automated code review. For example, Lu et al. [23] have introduced LLaMA-Reviewer to automate the code review task. Yu et al. [37] present Carllm for improved precision and clarity in automated code review. Tang et al. [33] introduce CodeAgent, an approach in which multiple agents collaborate to find code quality issues. Google's DIDACT project [15] trains ML models on the sequential steps in software development processes, such as code review, to automate them. Although these approaches may be able to ensure code quality in the future, we see an overhanging risk that the interpersonal and team benefits of code review will be lost in such a development.

In this paper, we present a vision for using AI to *support* code review and its users, rather than replacing the activity. We believe that we should strive to boost *all* the positive effects of code review, including interpersonal effects such as knowledge transfer, team awareness, and shared code ownership. We propose to do this by focusing on the participants of the code review process, the authors and reviewers – the users, and their needs. We envision an adaptive code review pipeline, powered by an AI agent-based architecture, that provides support customized to the needs of each code review setting.

The contributions of this paper are an architectural design for an AI agent-based code review platform (Section 3), an analysis of the design with respect to code review benefits (Section 4), and an outline of future research to realize this vision (Section 6).

2 Today's code review and its user needs

Modern code review practices are centered on a web-based code review tool, such as open source tools like Gerrit, open services like



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728505

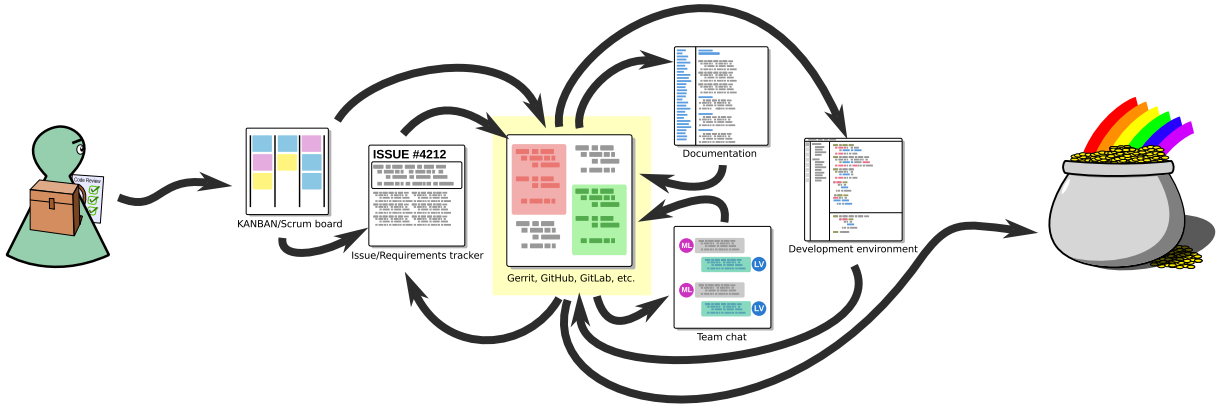


Figure 1: Illustration of navigation between tools in modern code review.

GitHub and GitLab [11], or proprietary tools like Critique [29] and CodeFlow [3]. These tools contain functionality to list code changes awaiting review, compare the changed and original code [17], write review comments, respond to review comments, and vote on the next steps. By integrating with continuous integration (CI) systems [22, 30] they can show results from automated tests, clean code with automatic formatters, and reject code based on compiler or linter errors.

There are several challenges with today's code review practices and tools. The information needed to complete the review is scattered across different systems such as issue trackers, requirements databases, KANBAN boards, team chats, API documentation, and CI reports. Different users involved in code review will have different needs, processes, and goals when using code review systems [20]. For example, the author of the change may want to view the code and the rationale behind it briefly to discover mistakes before submitting it for review. An expert reviewer might want to get an overview of architectural changes and the performance profile before and after the change. A new team member could spend extra time understanding the rationale and need to ask questions about unfamiliar patterns or APIs. Some team members might skim the rationale and code to stay up-to-date with changes in the repository, but not vote or write comments.

The reviewer must use their experience and the team processes to navigate between tools effectively and decide which steps are helpful and when [24]. Sometimes, even check out the code locally to trace variables and execute the code to verify its behavior and performance. This experience takes time to build up, and becoming effective at carrying out code reviews in a new workplace can take up to a year [5]. Even with experience, it demands time, effort, and focus [31]. There are often difficulties in understanding the rationale for the change [8] and reviewing large changes [13]. Multiple review cycles between reviewers and authors, together with long response times, can create delays affecting the overall productivity [12, 13].

Figure 1 illustrates what this can look like for a developer embarking on a code review. Carrying their experience as a backpack full of resources and a checklist with the team code review process, they switch between different tools and systems. The code review system (Gerrit, GitHub, GitLab, etc.) is in the center, and the point to return to and start from. With experience, iteration, and help from their peers, they can reach the "pot of gold" containing improved code quality, better knowledge distribution, team cohesion, and more.

3 Design proposal

Our design proposal is a code review platform that is built on an agent-based AI-OS architecture [26]. In the AI-OS architecture, a central LLM takes a role similar to the kernel in an operating system and is responsible for interpreting user input, planning, and coordination. Smaller AI agents connected to the central LLM manage integrations to databases and online APIs; functioning as the input, output, and memory subsystems in the analogy to an operating system. Several specialized Small Language Models are trained to create prompts, construct database queries, build API calls, and combine results [26].

In the case of code review, these agents implement integrations with the version control system, requirements database, issue tracker, continuous integration, API documentation, etc., to collect all the information needed before, during, and after code review. We envision a user interface that embeds existing and familiar tools, such as Gerrit, GitHub, GitLab, CodeFlow, etc., in the center. Information from, guidance by, and interaction with the agent-based system is placed below and in the sidebar.

Figure 2 illustrates our envisioned code review platform architecture with an example flow. Throughout the review, the reviewer interacts with the LLM trained to act as the platform's foundation. Information about the user's preferences and the team's code review process (the backpack filled with experience and the process document from Figure 1) can be configured by the team and the user. Parts of this configuration could potentially be updated with reinforcement learning or similar approaches to make it adaptable over time and for different contexts. The LLM customizes the process for each user using the configuration and coordinates the AI agents to provide the information and support needed at the right time during code review.

For example, for an in-depth review use case, the platform could first assist with picking a change to review, given the reviewer's

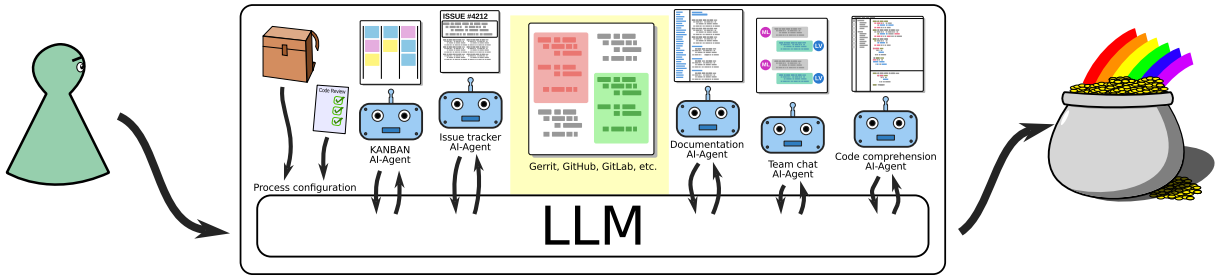


Figure 2: Proposal for a code review platform architecture driven by AI-agents.

time constraints. Then help throughout the process of understanding the rationale of the change, connecting it to related work, reading the code changes, finding potential defects, writing constructive comments, and finally assist in making a decision on accepting the code for integration or sending it back to the author for adjustments.

4 Analysis of design

An AI agent-based architecture has the potential to preserve or amplify all the benefits of code review, as described by Bacchelli and Bird [3], while at the same time reducing users' mental load and time spent. The agents are trained or tuned for each aspect of code review, and the adaptable nature of the platform allows it to fit the needs of different users and teams.

4.1 Defect finding, code improvement, and alternative solutions

Current and future work on automated defect finding can be integrated into this architecture as one of the AI agents. An option would be to run a model similar to CodeAgent [33], but instead of automation support the user by marking parts of the code that could contain a defect and suggesting code review comments. Other agents could be trained to look for performance improvements, refactorings, and alternative solutions.

4.2 Knowledge transfer

Knowledge transfer during code review has, for example, been shown to reduce the impact of developer turnover by exposing developers to code they have not authored [16]. Achieving this requires keeping the human in the loop, i.e. doing the code review supported by AI rather than automated with AI. AI agents can further be used to gather information so that the user does not have to navigate different systems to piece together the rationale, system architecture, and implementation details. They can also expand on user code review comments with references to team guidelines, language conventions, design patterns, and best practices.

4.3 Team awareness

For users reading through code reviews to get awareness of ongoing work, it can be time-consuming and demanding to read through a large code diff just to understand what it does. Language models for code comprehension are developing rapidly, with many options both in open source [19] and in closed source [7]. Receiving a code summary as soon as you open the review could be enough for users looking to be aware of current changes with the option of going deeper into the changed code when needed.

4.4 Improve developer process

A unified platform for the whole code review flow, instead of manually switching between code review tool, documentation, KANBAN boards, etc. will streamline the developer process as a whole. Introducing, configuring, and refining the settings and use of the platform can also encourage teams to examine and improve their process. The adaptability of a multi-agent approach can also help situations where different teams follow different workflows but still need to work together.

4.5 Share code ownership

Software development teams are at risk of developing a blame culture [18], where developers are held personally responsible for introduced bugs. This culture can lead to a reluctance to contribute new features and undermines trust between team members. Code review can contribute to preventing or mitigating this. It is no longer solely the fault of the author if a bug is introduced, but also of all the reviewers who did not discover it.

This is another benefit that would be at risk if code reviews were automated. Supporting the author and reviewers in assessing the code and giving them good grounds for their decision using AI, but leaving the decision of approving, rejecting, or revising the code up to the users keeps the developers accountable and encourages a culture of shared ownership.

4.6 Avoid build breaks

Connecting CI with code reviews is an effective way to encourage reviewer participation and ensure passing builds [27]. This kind of system already automatically rejects changes that break the build. An AI agent could help by suggesting a fix or pointing out the most likely causes.

4.7 Track rationale

Finding the rationale can involve piecing together information from commit message, issue tracker, product plan, recent team meetings, team chat conversations, and more. Bringing this information together and summarizing it in one place makes it easier to find the rationale behind a code change and connect it to larger goals and efforts.

4.8 Team assessment

Code reviews generate metrics such as the number of comments written, acceptance rate for posted changes, time from comment to resolution, etc. Measuring individual and team performance in software development is very difficult, and looking at reviews may provide additional insight. Our proposed design does not affect the collection of metrics, but has the potential to make code reviews more efficient, increase quality, and improve assessment metrics for the whole team.

5 Related work

There are a few recent studies, also focused on supporting rather than automating code review, that complement the vision in this paper.

Unterkalmsteiner et al. [34] explore providing reviewers with a better context for the code under review with a proposal called the Code Review Contextualizer. Using existing literature on what developers need help with during code review, they present several parts that could be improved by data collection and assistance. Although they do not go into detail regarding what kind of technologies and architectures could be used to implement such a system, their research on use cases that should be supported is a valuable foundation for building future AI-supported code review systems.

Almedia et al. [2] present AICodeReview, a plugin for the IntelliJ integrated development environment that takes advantage of GPT 3.5. The plugin analyzes code snippets while they are being written and identifies potential issues. Comments, resolutions, and improvement suggestions are provided in the editor. This approach can likely reduce human code review time since changes submitted for code review are hopefully of higher quality than they would have been without the AICodeReview plugin.

Wang et al. [35] presents an AI agent-based approach to recommend which reviewers that should be assigned to each code review. Their work complements the suggestion in this paper very well in that it seeks to build AI-based support systems for human reviewers instead of automating the activity. The article shows better preliminary performance using AI agents compared to previous state of the art for reviewer recommendation.

6 Research agenda

Here, we list research activities that we believe are important for realizing our proposed design vision.

6.1 Increased understanding of user needs

Recent work improves our understanding of the causes of confusion [8], anxiety [21], and misalignments [31] in code review. This research helps to provide a deeper understanding of user needs and user experience in code review, but there is much more to study here. For example, the needs of each user in code review vary [31], and this variation goes beyond roles such as author and reviewer, and may extend into tasks such as gatekeeping [29].

6.2 Measuring effectiveness of code review

Despite the wide use of code review in industry and its time-consuming nature, there is no unified way to measure the effectiveness of code review. The primary benefit explored with regard to measuring is 'defect finding' [25]. Other benefits, such as knowledge sharing, team awareness, and shared ownership, have not been studied as extensively. With a deeper understanding of the effectiveness of code review, we can consider cases where code review is the most effective with respect to different benefits. This understanding would open up for addressing the reported industry need for optimization [12], but without an unintended loss of code review benefits.

6.3 Effective code review interaction

The interaction with today's code review tooling has stayed largely the same since the introduction of the ICICLE tool in the 90s [6], with variations of interfaces centered around textual diff views of changed files (Gerrit, GitHub, Critique, and so on). Although this user interface design helps to provide answers to questions best answered with a textual diff, they are less successful at answering questions connected to, for example, requirements or execution behavior [31]. There is room for more exploration and innovation here to better align the interaction with user needs.

An interesting research direction would be to explore the use of collaborative and user-centered design processes [10] to take advantage of the depth of experience in the software development community. Another interesting direction would be to explore new ideas from conversational interaction design [14]. There are examples on the use of conversational interaction design for software development tools [9, 28] and recent advances in language models provide interesting new possibilities.

6.4 Effective AI integration

There are technical challenges in how best to build an AI agent-based code review pipeline, as outlined in our proposed design. One challenge is to identify suitable tasks for agents. For example, should an agent focus on one aspect of program comprehension, like code summation, or rather be trained for larger functional areas? There is also the challenge of choosing suitable models for different agents, along with data collection, training, and tuning. The models need to be integrated, and the main LLM trained to manage orchestration based on configured user needs.

An interesting direction is to identify a minimal viable use case and iterate on a smaller instance of the design with fewer AI agents. Follow best practices for AI in software development [36] and maintain close interaction with industry practitioners for rapid prototyping, early user feedback, and testing.

Acknowledgments

The authors would like to thank the Swedish Strategic Research Environment ELLIIT, the Swedish Foundation for Strategic Research (grant nbr. FFL18-0231), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, for partly funding this work.

References

[1] A Frank Ackerman, Priscilla J Fowler, and Robert G Ebenau. 1984. Software inspections and the industrial production of software. In *Proc. of a symposium on Software validation: inspection-testing-verification-alternatives*. 13–40.

[2] Yonatha Almeida, Danyllo Albuquerque, Emanuel Dantas Filho, Felipe Muniz, Katyusco De Farias Santos, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. 2024. AICodeReview: Advancing Code Quality with AI-enhanced Reviews. *SoftwareX* 26 (May 2024), 101677.

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

[4] Amiangshu Bosu and Jeffrey C Carver. 2013. Impact of peer code review on peer impression formation: A survey. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 133–142.

[5] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 146–156.

[6] L. Brothers, V. Sembugamoorthy, and M. Muller. 1990. ICICLE: Groupware for Code Inspection. In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work (CSCW '90)*. ACM, 169–181.

[7] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. *GPTutor: A ChatGPT-Powered Programming Tool for Code Explanation*. Springer Nature Switzerland, 321–327.

[8] Moataz Chouchen, Ali Ouni, Raula Gaikovina Kula, Dong Wang, Patanamon Thongtanunam, Mohamed Wiem Mkaouer, and Kenichi Matsumoto. 2021. Anti-patterns in Modern Code Review: Symptoms and Prevalence. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 531–535.

[9] Luke Church, Emma Söderberg, and Alan McCabe. 2021. Breaking down and making up-a lens for conversing with compilers. In *Psychology of Programming Interest Group Annual Workshop 2021*.

[10] Sasha Costanza-Chock. 2020. *Design Justice: Community-Led Practices to Build the Worlds We Need*. The MIT Press. arXiv:https://direct.mit.edu/books-pdf/2248508/book_9780262356862.pdf

[11] Nicole Davila and Ingrid Nunes. 2021. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software* 177 (2021), 110951.

[12] DORA. 2023. *Accelerate State of DevOps 2023*. Technical Report. DORA. https://dora.dev/research/2023/dora-report/

[13] Emre Doğan and Eray Tüzün. 2022. Towards a taxonomy of code review smells. *Information and Software Technology* 142 (2022), 106737.

[14] Hugh Dubberly and Paul Pangaro. 2019. *Cybernetics and Design: Conversations for Action*. Springer International Publishing, 85–99.

[15] Google Research. 2023. *Large sequence models for software development activities*. https://research.google/blog/large-sequence-models-for-software-development-activities/

[16] Fahimeh Hajari, Samaneh Malmir, Ehsan Mirsaeedi, and Peter C. Rigby. 2023. *Factoring Expertise, Workload, and Turnover into Code Review Recommendation*. doi:10.48550/arXiv.2312.17236 arXiv:2312.17236 [cs]

[17] Lo Heander, Emma Söderberg, and Christofer Rydenfält. 2024. Design of Flexible Code Block Comparisons to Improve Code Review of Refactored Code. In *Companion Proceedings of the 8th International Conference on the Art, Science, and Engineering of Programming (Programming '24)*. ACM, 57–67.

[18] F. Hein. 1998. The Blame Game. *IEEE Software* 15, 6 (1998), 89–91.

[19] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. *Qwen2.5-Coder Technical Report*. doi:10.48550/ARXIV.2409.12186

[20] Raula Gaikovina Kula, Ana E. Carmago Cruz, Norihiro Yoshida, Kazuki Hamasaki, Kenji Fujiwara, Xin Yang, and Hajimu Iida. 2012. Using Profiling Metrics to Categorise Peer Review Types in the Android Project. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*. IEEE, 146–151.

[21] Carol S Lee and Catherine M Hicks. 2024. Understanding and effectively mitigating code review anxiety. *Empirical Software Engineering* 29, 6 (2024), 161.

[22] Anton Ljungberg, David Åkerman, Emma Söderberg, Gustaf Lundh, Jon Sten, and Luke Church. 2021. Case study on data-driven deployment of program analysis on an open tools stack. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 208–217.

[23] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. 647–658.

[24] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2018. Code Reviewing in the Trenches: Challenges and Best Practices. *IEEE Software* 35, 4 (2018), 34–42.

[25] Mika V. Mäntylä and Casper Lassenius. 2009. What Types of Defects Are Really Discovered in Code Reviews? *IEEE Transactions on Software Engineering* 35, 3 (2009), 430–448.

[26] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560 [cs] http://arxiv.org/abs/2310.08560

[27] Mohammad Masudur Rahman and Chanchal K. Roy. 2017. Impact of Continuous Integration on Code Reviews. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 499–502.

[28] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. ACM, 491–514.

[29] 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 (ICSE-SEIP '18)*. ACM, 181–190.

[30] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. 2015. Tricorder: Building a program analysis ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 598–608.

[31] Emma Söderberg, Luke Church, Jürgen Börstler, Diederick Niehorster, and Christofer Rydenfält. 2022. Understanding the Experience of Code Review: Misalignments, Attention, and Units of Analysis. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering (EASE '22)*. ACM, 170–179.

[32] Statistia. 2025. *Number of software developers worldwide in 2018 to 2024*. https://www.statista.com/statistics/627312/worldwide-developer-population/

[33] Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawendé F. Bissyandé. 2024. CodeAgent: Autonomous Communicative Agents for Code Review. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11279–11313.

[34] Michael Unterkalmsteiner, Deepika Badampudi, Ricardo Britto, and Nauman Bin Ali. 2024. Help Me to Understand this Commit!-A Vision for Contextualized Code Reviews. In *Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments*. 18–23.

[35] Luqiao Wang, Yangtao Zhou, Huiying Zhuang, Qingshan Li, Di Cui, Yutong Zhao, and Lu Wang. 2024. Unity Is Strength: Collaborative LLM-Based Agents for Code Reviewer Recommendation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento CA USA, 2024-10-27). ACM, 2235–2239. doi:10.1145/3691620.3695291

[36] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2024. Agents in Software Engineering: Survey, Landscape, and Vision. arXiv:2409.09030 [cs.SE] https://arxiv.org/abs/2409.09030

[37] Yongda Yu, Guoping Rong, Haifeng Shen, He Zhang, Dong Shao, Min Wang, Zhao Wei, Yong Xu, and Juhong Wang. 2024. Fine-Tuning Large Language Models to Improve Accuracy and Comprehensibility of Automated Code Review. *ACM Trans. Softw. Eng. Methodol.* 34, 1, Article 14 (Dec 2024), 26 pages.