

Quo Vadis, Code Review?

Exploring the Future of Code Review

Michael Dorner, *Blekinge Institute of Technology, Karlskrona, Sweden*

Andreas Bauer, *Blekinge Institute of Technology, Karlskrona, Sweden*

Darja Šmite, *Blekinge Institute of Technology, Karlskrona, Sweden*

Lukas Thode, *Blekinge Institute of Technology, Karlskrona, Sweden*

Daniel Mendez, *fortiss & Blekinge Institute of Technology, Karlskrona, Sweden*

Ricardo Britto, *Ericsson & Blekinge Institute of Technology, Karlskrona, Sweden*

Stephan Lukasczyk, *JetBrains Research*

Ehsan Zabardast, *Blekinge Institute of Technology, Karlskrona, Sweden*

Michael Kormann, *SAP*

Abstract—Code review has long been a core practice in collaborative software engineering. In this research, we explore how practitioners reflect on code review today and what changes they anticipate in the near future. We then discuss the potential long-term risks of these anticipated changes for the evolution of code review and its role in collaborative software engineering.

“Nothing is constant except change.”

—Heraclitus

During the early days of software engineering, code was often designed and implemented by individual developers working in isolation. As the software systems became more and more complex, software engineering became a collaborative effort requiring many developers with diverse skill sets and specializations, organized in different teams, and later even distributed around the globe. To maintain and enhance a collective understanding of the code base and the changes to it, developers need to discuss a code change before it gets merged into the code base. Over the years, these discussions have taken various forms ranging from the formal and heavy-weight *code inspections* in the 1980s⁹ to synchronous and steady exchanges between two developers in *pair programming*.¹³ Nowadays, the asynchronous and less formal discussions around code changes are known

as *code reviews*, a practice that has seen widespread adoption in collaborative software engineering.^{3,8}

However, change is constant in software engineering—and code review is no exception. Advances in large language models (LLMs) and new regulatory frameworks, such as the EU Cyber Resilience Act, are beginning to influence how code reviews are conducted.

In this article, we take a glimpse into the future of code review. To that end, we begin by analyzing how practitioners reflect on code review today and what they expect to change, based on a survey of 92 professional software developers from four large software-driven companies, each representing a distinct organizational and software engineering context. Building on this analysis, we discuss potential long-term risks for the future role of code review as a core practice in collaborative software engineering.

Code Review Five Years From Now

What do developers actually do in code review today and how do they think that will change? In this section, we report the results of our survey, capturing how developers currently experience code review today and

Table 1. Organizational and Software Engineering Contexts and Participant Numbers by Company

Attribute	SAP	Ericsson	A Bank	JetBrains
Industry Domain	Enterprise Software	Telecommunications	Banking and Finance	Developer Tools
Business Model	Software products (B2B SaaS, ERP)	Infrastructure provider (network equipment, telecom software)	Financial services (retail and investment banking)	Commercial software vendor (IDEs and tooling)
Primary Software Focus	Customer-facing enterprise platforms	Embedded and network control software	Internal financial transaction and risk management systems	Developer productivity and language tooling
Number of Employees	>100,000	>100,000	>25,000	>2,200
Number of participants	32	24	25	11

how they expect it to change in the coming years.¹ The analysis is structured around three central questions from our survey: the *time* they currently spend on code review and how they expect that to change; the *types of artifacts* they review today versus those they anticipate reviewing in the future; and *who* they expect to be involved in the code review process going forward.

Not Less Time

Currently, respondents report spending a median of approximately three hours per week on code reviews. To contextualize this finding, we compare it against two external data sources: first, a 2013 survey by Bosu et al. involving 416 Microsoft developers; and second, the 2019 Stack Overflow Developer Survey, which includes responses from 49 790 developers². Figure 1 shows the cumulative distributions of weekly review hours across all three data sources. The distribution in our sample is consistently lower compared to both the 2013 Microsoft study by Bosu et al. and the 2019 Stack Overflow Developer Survey, which show strikingly similar distributions with a median of around four hours spent on code reviews. The reasons for this difference are not evident from our data. Possible explanations may include a decline in the perceived importance of code review over the past years, differences in domain-specific practices, or inaccuracies in self-reported estimates.

A vast majority of practitioners do not expect to spend less time on code review in the coming years. According to our survey:

- 47 % expect to spend *more time*,
- 31 % expect to spend *about the same amount* of time, and
- 22 % foresee spending *less time*.

¹See sidebar for more details on study design.

²<https://survey.stackoverflow.co/2019>

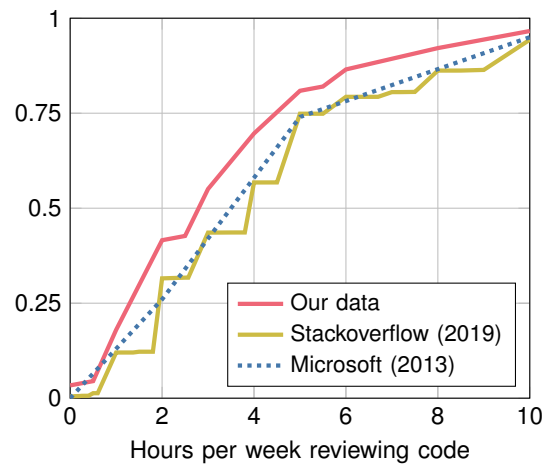


Figure 1. Cumulative distribution of hours spent on code review today (≤ 10 hours). In the absence of raw data, we approximated the cumulative distribution from the reported ordinal-scale results at Microsoft.⁴

Although those results suggest that code review might play even a more central role in collaborative software engineering in the near future, we are cautious about extrapolating this insight to the entire software industry. The distributions in our sample are consistently lower than those in our reference datasets—though these datasets are older—which raises the possibility that the companies in our study are simply catching up to broader industry standards rather than indicating a widespread shift in code review practices.

Broader Coverage

Practitioners expect to review a broader range of artifacts in the near future in comparison to today. Over the next five years, we see an increasing interest in reviewing more artifact types, including production code, test code, configuration files, documentation, and GUI test code. Notably, the share of practitioners who

Sidebar: Study Design

Sampling To gain a diverse perspective on the future of code review grounded in real-world practice, we employed *quota sampling*, a non-random, two-stage strategy. First, we purposively selected four relevant companies representing varied organizational and software engineering contexts. Then, we aimed to recruit approximately 25 professional developers from each company, closing recruitment after eight weeks or once the quota was reached.

Participants Our sample includes 92 developers from (1) one of the world's largest enterprise software providers (SAP), (2) a globally leading provider of telecommunications infrastructure (Ericsson), (3) one of Europe's largest banks, which has requested anonymity, and (4) a mid-sized vendor of professional developer tools, including widely used IDEs (JetBrains). The number of participants per company varied slightly. See Table 1 for an overview of the participating organizations, their contexts, and participant numbers. Though not statistically representative and with varying participation per company, the sample captures diverse industry perspectives on code review.

Questions To capture developers' current code review practices and their expectations for how the practice may evolve over the next five years, we asked six questions:

1. How many hours on average do you currently review code per week? (numerical response)
2. Do you expect to spend more, less, or the same amount of time on code review in five years (compared to today)? (Multiple choice)
3. What software artifacts do you review today? (Multiple selection)
4. What software artifacts do you anticipate reviewing in five years? (Multiple selection)
5. What major changes do you anticipate in code review in five years? (Open-ended)
6. Based on those changes, what implications for code review do you see? (Open-ended)

The survey covered six types of software artifacts covered in code review: Production code, Test code, Parameter and configuration files, Documentation, and GUI-based test code (end-to-end testing). Respondents could also add further artifact types or choose to select none.

Data collection The survey was conducted end of 2024 and beginning of 2025 via an online questionnaire, shared through internal communication channels at the respective companies. Participation was both voluntary and anonymous, allowing respondents to share their perspectives freely. To further protect confidentiality, we deliberately excluded demographic questions such as age, gender, or company affiliation, and report the results on sample level rather than by individual company. Participants were informed about the study's purpose and assured that their participation was voluntary, and their responses would remain anonymous.

Data Analysis Quantitative and multiple-choice responses were analyzed using descriptive statistics. Open-ended responses were examined using *thematic analysis*⁵ to uncover common patterns and emerging trends.

Data availability All anonymized data and analysis scripts are publicly available at github.com/michaeldorfner/quo-vadis-code-review

currently do not review any artifacts (around 6%) is expected to decrease.

Production code will remain the primary artifact under review in the future. There is also a broad consensus that parameter and configuration files, documentation, and test code will continue to be reviewed regularly. A particularly notable trend is the comparatively sharp increase in the future prediction of review focus on GUI-based test code, suggesting a growing recognition and integration of it in future software development practices.

The connected dot graph in Figure 2 highlights the expected changes in the coverage of reviewed artifacts in the future.

The New Code Review Participant

Given the current hype around generative AI, it is hardly surprising that nearly all practitioners, when asked about anticipated major changes in future code reviews, expect large language models to become active participants in the process. This trend is also highlighted by Davila et al.,⁶ whose gray literature review identified strong practitioner interest in both proposing novel generative AI-based solutions and adopting established tools such as ChatGPT for code review.

In contrast to the deterministic, rule-based, or

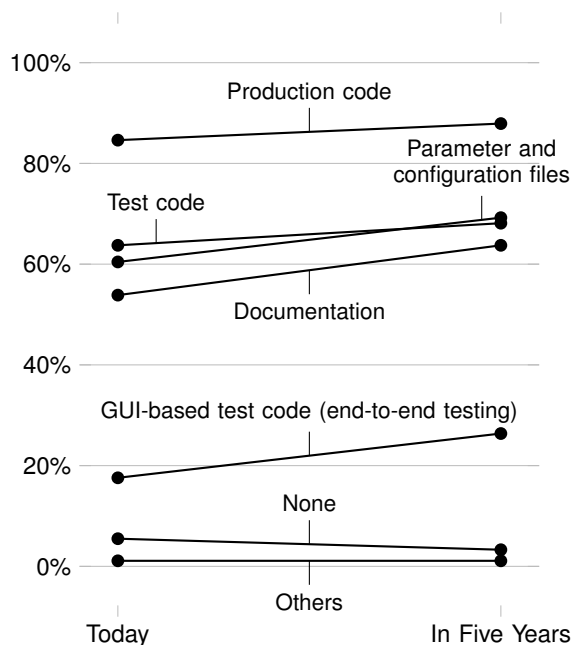


Figure 2. Practitioners anticipate reviewing a broader range of artifacts in the future, with notable growth in attention to GUI-based test code and a decline in those reviewing no artifacts at all.

pattern-matching bots that long assisted in code reviews,² survey respondents envision LLMs taking a way more active role: An LLM might act as the code reviewer, assessing the code written by humans and providing feedback through an *automated review*. The anticipated autonomy of LLMs varies and can be viewed along a spectrum, ranging from supporting human reviewers as an additional assistant (“*AI might take over some parts of the code review process.*”) to fully conducting the review independently (“*complete code review with the help of AI*”).

Alternatively, LLMs could serve as the authors, *automatically generating code* that is then reviewed by humans, keeping developers in the loop: “*Code will be generated by generative-AI based tools, but reviewed by humans. A review by humans will be mandatory to avoid any law suits on the code development organization.*” Again, the depth of involvement in code authoring lies on a spectrum, from assisting developers to autonomously proposing and integrating code changes. The amount and pace of generated code by AI will force us to review the code in a different way. In this scenario, concerns arise about the long-term impact on developer expertise. As one participant warned: “*I am a bit concerned about the future gener-*

ation of developers as they will have less experience actually coding.”

Two respondents also considered that automated code review intersects with automated code generation and “*AI [is] everywhere*”. We refer to the intersection where both code generation and review are fully handled by LLMs and eliminating the need for human involvement as *unsupervised software engineering*. Yet this shift comes with caveats, as one participant explicitly warned: “*We will have to decide if we want to write code with AI or review code with AI [...] I don't think both at the same time will work, because there can be instances, where the programmer generates bad code with the AI, and a mistake is kept in, and because of time shortage the reviewer also uses AI to review the same code, and the AI doesn't catch the mistake because it generated it.*” Expressing concern about the future towards unsupervised software engineering, one developer stated: “*I expect that there will be more generated code of a worsen quality. I expect that there will be a lot of AI code review systems, which will be pretty useless.*”

We also encountered fundamental opposition to the use of AI in code review—whether as author or reviewer—as one participant put it: “*As long as AI is involved, it is likely to be a more tedious process [code review] than before.*” We refer to this stance as human-led software engineering, a position in direct contrast to a future of fully unsupervised software engineering without human oversight.

Based on practitioner input, we summarize the anticipated roles of AI in code review as a spectrum of involvement, gradually ranging from fully human-led to fully LLM-led, in both the roles of code author and reviewer. As illustrated in Figure 3, four possible futures of software engineering emerge from this gradual interplay between humans and LLMs, each reflecting a distinct balance in the roles of author and reviewer.

Doesn't LGTM

Our survey highlights two key expectations from practitioners about the future of code review. First, code review will remain a central practice in collaborative software engineering. Developers anticipate that it will continue to demand the same or even more time in the coming years, while expanding to include a broader range of artifacts. Second, with the growing integration of LLMs, the traditionally human-driven nature of code review with developers leading discussions, making judgments, and approving changes^{2,8} is expected to shift.

In this section, we invite the reader to envision the

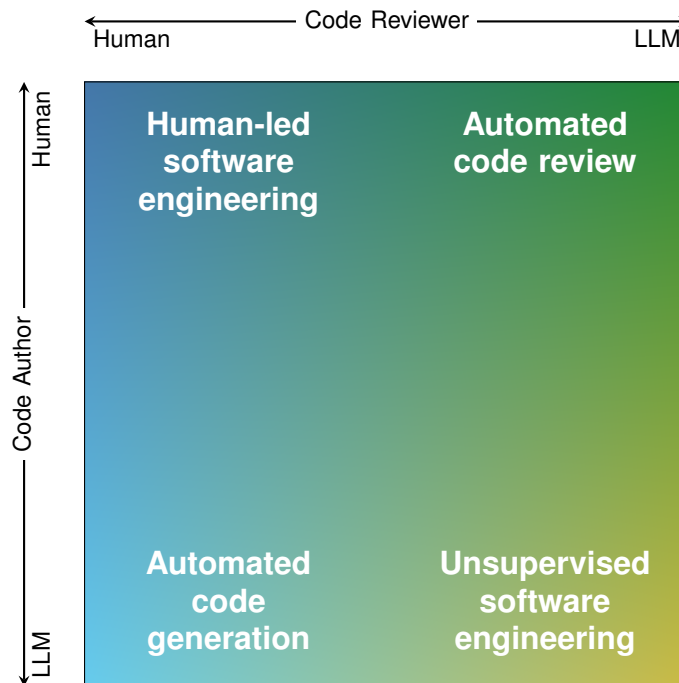


Figure 3. Practitioners expect the roles of code author and code reviewer to lie on continuous spectra, each ranging from fully human-led to fully LLM-led. Based on this human–AI interplay in code review, four possible futures of collaborative software engineering begin to take shape.

future that practitioners anticipate, one where AI takes a central role in code review. Based on this premise, we discuss three forms of erosion that are likely to affect collaborative software engineering broadly, but surface first in code review.

Erosion of Understanding

At its core, code review, and all the many review guidelines that support it,³ ultimately boils down to a single key question: *Would I be able to maintain this code change in the future if the original author were no longer available?* If the answer is no, it signals that the code may be too complex, poorly documented, badly structured, or unclear, and likely needs revision. By introducing a second pair of eyes, code review not only enhances maintainability and consistency but also acts as a mechanism for capturing and preserving human understanding of the software system and the broader implications of ongoing changes.

Now, imagine a future where an LLM handles code reviews and optimizes code to its own capacity to “understand” it. Ironically, this could lead to code becoming less maintainable over time—from a human perspective. What is readable and logical to an LLM may be opaque and not intuitive to a human developer.

As a result, human understanding of the changes in the evolving code base could gradually erode, leaving only other LLMs capable of maintaining it. And this might happen subtly: Even for AI-assisted code reviews where we keep the human in the process, reviewers focus more on the annotated code section and potentially overlooking issues elsewhere which could limit a comprehensive understanding.¹²

Should we, then, make it a priority for humans to review code that is authored (partially or entirely) by LLMs to ensure long-term human maintainability of the code base, not just short-term AI comprehension? This approach also has its cons. When developers routinely defer to LLM-generated solutions—especially without critically engaging with or questioning them—we risk fostering a *passive development culture*. Combined with an increased volume of LLM-generated code and thus review workload, developers may be forced to treat code as a black box—accepting solutions that seem to work, but without fully grasping their structure, intent, or long-term consequences.

Code is more than just functional software. It is a shared language among developers. If only machines can understand it, we risk losing not only our connection to the people we build systems for, but also our

ability to meaningfully control the systems we create.

Erosion of Accountability

Code review is not just about finding bugs—it is about reinforcing accountability, ownership, and responsibility in software development.¹ By requiring developers to justify their changes and reviewers to critically evaluate them, code review ensures that every line of code has a clear owner—someone responsible for its quality, security, and long-term maintainability.¹⁴ In contrast, AI systems can neither be held responsible nor accountable (let alone liable) in the same way, creating a gap in ownership of the code base.

This gap places software companies in a legal limbo in the context of regulatory compliance. Emerging regulations such as the EU's *Cyber Resilience Act*, which mandates effective and regular reviews of software systems with a focus on security, or the *Transfer Pricing Guidelines* by the OECD, which call for a more precise definition of code ownership, both place accountability at the forefront of regulatory priorities and expectations.⁷

As AI systems take on a greater role in collaborative software engineering, ensuring clear human accountability will be critical,¹⁰ not only for maintaining code quality, but also for meeting the growing legal and regulatory demands placed on software organizations.

Erosion of Trust

Trust is fundamental for code review—just as it is in any form of human collaboration. With the increasing involvement and reliance on LLMs, the foundation for trust is at risk. It is becoming increasingly unclear who truly authored a piece of code or a code review. Was it written by a human? By an LLM? Or, perhaps more worryingly, a human blindly parroting LLM-generated content without critical thought or understanding? This growing uncertainty signals a deeper issue—a gradual *erosion of trust* in the authenticity and accountability of the review process.

The erosion of trust in code review is not a distant threat—it is a problem we already face. In his blog post titled “The I in LLM stands for intelligence,” Daniel Stenberg, the maintainer of the curl project, describes how some individuals use LLMs to analyze curl's code and generate security vulnerability reports.¹¹ These AI-generated reports often appear legitimate but are shallow and inaccurate, forcing maintainers to spend valuable time assessing and eventually dismissing them. This problem highlights the growing challenges to distinguish meaningful contributions from synthetic.

We fear a similar risk in code review. When reviewers cannot distinguish between genuine human insight and automated suggestions, feedback becomes superficial and potentially harmful. Code review loses its purpose and deviates from its critical function, becoming an empty shell—a routine for the sake of routine, rather than being a meaningful safeguard for code quality and collaboration.

Conclusion

Code review has long been at the heart of collaborative software engineering where experience, reasoning, and thoughtful feedback drive the evolution of software systems. And our results indicate that code review will not disappear in the foreseeable future. In fact, developers expect to spend the same or even more time reviewing code, and reviewing more artifacts in the process.

While code review will remain essential, it will evolve. The results of our survey clearly show that LLMs are expected to become ubiquitous and to take on more tasks. The question remains to what extent code review shall be delegated to AI or kept in human hands? Without unnecessarily demonizing LLMs, we raise concerns for the consequences of routine over-reliance on LLMs in code review—risks of eroding human understanding, accountability, and trust, ultimately reducing code review to an empty shell.

As software engineering is in constant evolution, raising questions about how to improve efficiency through automation, we advocate for a nuanced approach to code review. Human understanding (even if not very efficient), accountability (even if not on today's agenda), and trust (even if not easily measured) ensure that we keep control of the long-term evolution of our software systems and remain resilient in the face of future changes, whether technical or transformational.

We advocate for further research to explore the implications of integrating LLMs and other AI technologies into code review, as we believe code review is where the effects of LLM adoption in collaborative software engineering are likely to surface first. An over-reliance on AI gradually shifts the focus of understanding from human teams to the LLM—an entity that, despite its capabilities, cannot be held accountable, reason contextually, or justify its decisions beyond probabilistic associations. To preserve the integrity of software engineering as a collaborative, transparent, trustworthy, and responsible engineering practice, we must ensure that code review remains grounded in human understanding and oversight.

Key Insights


- Practitioners expect code review to remain essential, spending as much or more time on reviewing an increasingly broad range of artifacts in the near future.
- Large language models (LLMs) are expected to transform both code authorship and review, shifting code review away from a primarily human-centered practice.
- Routine over-reliance on LLMs in code review risks eroding understanding, accountability, and trust, ultimately reducing code review to an empty shell.
- Further research is needed to explore the implications of integrating LLMs into code review, where the effects of AI adoption in collaborative software engineering are likely to surface first.

Acknowledgement

We thank all participants of the survey. This work was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology.

References

- [1] Alberto Bacchelli and Christian Bird. “Expectations, outcomes, and challenges of modern code review”. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 712–721. DOI: [10.1109/ICSE.2013.6606617](https://doi.org/10.1109/ICSE.2013.6606617).
- [2] Deepika Badampudi, Michael Unterkalmsteiner, and Ricardo Britto. “Modern Code Reviews - A Survey of Literature and Practice”. In: *ACM Transactions on Software Engineering and Methodology* (Feb. 2023). DOI: [10.1145/3585004](https://doi.org/10.1145/3585004).
- [3] Andreas Bauer, Riccardo Coppola, Emil Alégroth, and Tony Gorschek. “Code review guidelines for GUI-based testing artifacts”. In: *Information and Software Technology* 163 (Nov. 2023), p. 107299. DOI: [10.1016/j.infsof.2023.107299](https://doi.org/10.1016/j.infsof.2023.107299).
- [4] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. “Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft”. In: *IEEE Transactions on Software Engineering* 43 (1 2017), pp. 56–75. DOI: [10.1109/TSE.2016.2576451](https://doi.org/10.1109/TSE.2016.2576451).
- [5] Victoria Clarke and Virginia Braun. “Thematic analysis”. In: *The Journal of Positive Psychology* 12 (3 May 2017), pp. 297–298. DOI: [10.1080/17439760.2016.1262613](https://doi.org/10.1080/17439760.2016.1262613).
- [6] Nicole Davila, Jorge Melegati, and Igor Wiese. “Tales From the Trenches: Expectations and Challenges From Practice for Code Review in the Generative AI Era”. In: *IEEE Software* 41 (6 Nov. 2024), pp. 38–45. DOI: [10.1109/MS.2024.3428439](https://doi.org/10.1109/MS.2024.3428439).
- [7] Michael Dorner, Maximilian Capraro, Oliver Treidler, Tom-Eric Kunz, Darja Šmite, Ehsan Zabardast, Daniel Mendez, and Krzysztof Wnuk. “Taxing Collaborative Software Engineering”. In: *IEEE Software* (2024), pp. 1–8. DOI: [10.1109/MS.2023.3346646](https://doi.org/10.1109/MS.2023.3346646).
- [8] Michael Dorner, Daniel Mendez, Krzysztof Wnuk, Ehsan Zabardast, and Jacek Czerwinka. “The Upper Bound of Information Diffusion in Code Review”. In: *Empirical Software Engineering* (June 2023).
- [9] M. E. Fagan. “Design and code inspections to reduce errors in program development”. In: *IBM Systems Journal* 15 (3 1976), pp. 182–211. DOI: [10.1147/sj.153.0182](https://doi.org/10.1147/sj.153.0182).
- [10] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekk, and David Doermann. “Future of software development with generative AI”. In: *Automated Software Engineering* 31.1 (2024). DOI: [10.1007/s10515-024-00426-z](https://doi.org/10.1007/s10515-024-00426-z).
- [11] Daniel Stenberg. *The I in LLM Stands For Intelligence*. 2024. URL: <https://daniel.haxx.se/blog/2024/01/02/the-i-in-llm-stands-for-intelligence/> (visited on 05/29/2025).
- [12] Rosalia Tufano, Alberto Martin-Lopez, Ahmad Tayeb, Ozren Dabic, Sonia Haiduc, and Gabriele Bavota. “Deep Learning-based Code Reviews: A Paradigm Shift or a Double-Edged Sword?”. In: *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 597–597. DOI: [10.1109/ICSE55347.2025.00060](https://doi.org/10.1109/ICSE55347.2025.00060).
- [13] Laurie Williams. “Integrating pair programming into a software development process”. In: *Proceedings 14th Conference on Software Engineering Education and Training. 'In search of a software engineering profession' (Cat. No.PR01059)*. IEEE Comput. Soc, pp. 27–36. DOI: [10.1109/CSEE.2001.913816](https://doi.org/10.1109/CSEE.2001.913816).
- [14] Ehsan Zabardast, Javier Gonzalez-Huerta, and Binish Tanveer. “Ownership vs Contribution: Investigating the Alignment Between Ownership



and Contribution". In: *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Mar. 2022, pp. 30–34. DOI: [10.1109/ICSA-C54293.2022.00013](https://doi.org/10.1109/ICSA-C54293.2022.00013).