Research Statement

Christoph Treude
School of Computing and Information Systems, Singapore Management University
Tel: (65) 6808 5438; Email: ctreude@smu.edu.sg
19 (Day) 12 (Month) 2025 (Year)

Background

Software systems underpin much of modern society, supporting critical activities in areas such as communication, finance, healthcare, transportation, and scientific research. As software becomes more pervasive and complex, the consequences of failures, security vulnerabilities, and poor usability become increasingly severe. Ensuring that software systems are reliable, secure, and maintainable therefore remains a central challenge for both industry and society.

Software development is fundamentally a human-driven activity. It requires coordination among developers with different roles and expertise, interpretation of evolving requirements, and continuous decision-making under uncertainty. Decades of software engineering research have shown that technical quality cannot be separated from human and social factors such as communication, knowledge sharing, and tooling support. Improving software quality therefore requires not only better algorithms and tools, but also a deeper understanding of how developers work and how they interact with those tools.

In recent years, artificial intelligence has moved from a supporting role to a central component of software development practice. Large language models and other AI-based techniques are now routinely used for tasks such as code completion, documentation, vulnerability detection, requirements tracing, and testing. These tools promise substantial gains in productivity and scalability, but they also introduce new challenges. AI-generated outputs can be incorrect, insecure, biased, or misleading, and their integration into developer workflows raises questions about trust, responsibility, and control. As AI systems increasingly participate in software development activities, software engineering must grapple with how to design, evaluate, and govern these systems in practice.

My research addresses this shift by examining software development as a socio-technical system in which humans and AI increasingly collaborate. Rather than treating AI as a black-box automation layer, my work focuses on understanding interaction patterns between developers and AI tools, measuring how AI-generated artifacts are actually used, and identifying the conditions under which AI support improves or degrades software outcomes. This perspective builds on empirical software engineering traditions, while extending them to account for AI-driven workflows and AI-mediated decision-making.

A second pillar of my work focuses on the use of AI to improve core software engineering tasks, particularly those related to software quality and security. AI-based techniques have demonstrated strong performance in areas such as vulnerability detection, code analysis, documentation, and requirements tracing, yet they also introduce new risks, including brittleness, adversarial susceptibility, privacy leakage,

and misalignment with developer intent. My research investigates both how AI techniques can be made more effective for software engineering tasks and how their limitations can be systematically identified and addressed. This includes treating AI-based software engineering tools as artifacts that themselves require rigorous evaluation, testing, and robustness analysis.

Finally, my work contributes to the emerging area of AI for science, with a particular focus on empirical software engineering and research methodology. Many empirical studies rely on costly and time-consuming human evaluations, which limits scale and reproducibility. Recent advances in AI raise the possibility of augmenting or partially automating aspects of this evaluation process. My research explores when and how AI systems can responsibly complement human judgment in research workflows, with the goal of improving scalability while preserving validity, transparency, and interpretability.

Across these strands, my overarching goal is to develop principled, evidence-based approaches for integrating AI into software engineering practice and research. By grounding AI-driven techniques in empirical evidence, interaction-aware design, and robustness considerations, my work aims to support the development of software systems—and software engineering methods—that remain trustworthy, effective, and aligned with human needs as AI becomes an integral part of the software ecosystem.

Research Area 1: Human–AI Interaction in Software Engineering

This research area focuses on how software developers interact with AI-based tools, and how these interactions shape adoption, trust, and software outcomes. As AI systems increasingly generate code, documentation, and analysis results, developers are no longer merely tool users but collaborators who must interpret, evaluate, and decide whether to rely on AI-generated artifacts. Understanding these interactions is essential for designing AI systems that genuinely support developers rather than introducing new sources of error, over-reliance, or cognitive overhead.

My recent work in this area examines human–AI interaction in software engineering from complementary perspectives that combine empirical measurement, conceptual framing, and methodological reflection.

A first line of work consists of controlled empirical studies that investigate when and why developers adopt AI-generated code. Rather than relying on self-reported intentions or attitudes, this work uses objective, behavioral measures of adoption, such as code similarity between AI-generated suggestions and developers' final solutions. In a controlled experiment with programmers of varying expertise, we showed that seemingly small properties of AI-generated artifacts—such as the presence of comments—can have a significant impact on whether developers adopt AI-generated code, regardless of their experience level [1]. These results highlight that adoption is influenced not only by perceived correctness or productivity, but also by factors related to comprehensibility, communication, and perceived intent.

A second line of work contributes conceptual foundations for studying developer–AI interaction. As AI tools have diversified rapidly, interaction patterns have become increasingly heterogeneous, ranging from passive auto-completion to conversational

assistance and workflow-triggered interventions. To support systematic study and comparison of these tools, we proposed a taxonomy of developer–AI interaction types that characterizes how interactions are initiated, how AI responds, and how developers react to AI output [2]. This taxonomy provides a shared vocabulary for analyzing interaction design choices, comparing tools, and situating empirical findings within a broader interaction landscape. Related perspective work further explores how such interaction patterns may evolve as AI systems become more autonomous and integrated into development environments.

A third line of work examines how AI systems themselves can participate in software engineering research workflows, particularly in evaluation tasks that traditionally require human judgment. Human-subject studies are essential but costly, and they often limit scale and reproducibility. In this line of work, we explored whether large language models can partially replace or complement human annotators for subjective software engineering tasks, such as judging code-related artifacts. Our results show that, for some tasks, model–human agreement approaches human–human agreement, while for others it does not, motivating mixed human–AI evaluation strategies rather than full automation [3]. This work provides concrete guidance on when AI can responsibly support human judgment and when human expertise remains indispensable.

Taken together, these studies frame human–AI interaction as a measurable, designable, and evaluable aspect of software engineering, rather than an incidental byproduct of tool adoption.

Looking ahead, my research in this area will focus on adaptive interaction strategies that adjust AI behavior based on developer context, task characteristics, and prior interaction history; on mechanisms for human control and oversight in AI-assisted development, including the communication of uncertainty and alternatives; and on extending empirical studies beyond short, isolated tasks to longitudinal and workflow-level settings. Understanding these longer-term effects is critical for assessing how AI tools influence learning, collaboration, and division of labor within software teams.

Research Area 2: AI for Software Engineering (AI4SE)

This research area focuses on the design, evaluation, and stress-testing of AI-based techniques for core software engineering tasks, with particular emphasis on software quality, security, and reliability. While recent advances in machine learning and large language models have led to strong performance on tasks such as vulnerability detection, code completion, and documentation support, their deployment in practice raises important questions about robustness, trustworthiness, and unintended side effects. My work in this area treats AI-based software engineering tools as software systems in their own right—systems that must be carefully evaluated, analyzed for failure modes, and improved using principled engineering approaches.

A central theme of my recent work has been the use of AI for software vulnerability detection. Existing learning-based approaches have demonstrated promising accuracy but often provide coarse-grained results that are difficult for developers to act upon. In collaborative work, we proposed techniques that move beyond file- or function-level classification toward fine-grained vulnerability localization, enabling AI

systems to identify specific code statements that contribute to a vulnerability. By reformulating vulnerability detection as a sequential decision-making problem and applying reinforcement learning, this work captures interactions among multiple statements rather than treating them independently, leading to more actionable results for developers [4].

Beyond improving effectiveness, my research also investigates the robustness of AI-based software engineering tools. As these tools are increasingly deployed in real-world settings, understanding their susceptibility to adversarial manipulation becomes critical. In this line of work, we studied black-box adversarial attacks on vulnerability detectors, focusing on realistic scenarios where model internals and confidence scores are not accessible. We demonstrated that state-of-the-art detectors can be evaded using semantics-preserving code transformations, exposing fundamental weaknesses in current approaches and highlighting the need for robustness-aware evaluation protocols [5].

Closely related to robustness is the issue of privacy and memorization in code language models. AI-based code completion systems are trained on large corpora that may include sensitive or proprietary code, raising concerns about unintended data leakage. My work in this area develops membership inference attacks tailored to code models, using adversarial prompting strategies to infer whether specific code snippets were present in a model's training data. These techniques provide a systematic way to assess memorization behavior and privacy risks, and they contribute to broader efforts around responsible and compliant use of code language models [6]. Related work on AI-generated code further explores how static analysis and feedback loops can be used to improve AI outputs beyond surface-level correctness.

Across these contributions, a recurring insight is that performance metrics alone are insufficient for evaluating AI-based software engineering tools. Accuracy improvements must be considered alongside robustness, security, privacy, and downstream developer use, particularly when these tools are applied in security- or safety-critical contexts.

My future research in this area will further integrate robustness and security considerations into the design of AI-based software engineering techniques; explore verification-oriented and hybrid approaches that combine learning-based methods with symbolic reasoning or explicit program representations; and investigate governance and accountability mechanisms for AI-assisted software development, including auditing, traceability, and compliance.

Research Area 3: AI for Science (AI4Science)

This research area focuses on the use of AI as a methodological instrument to support scientific inquiry, with an emphasis on empirical software engineering. Empirical studies in software engineering frequently rely on human judgment to evaluate tools, techniques, and artifacts. While such evaluations are essential for external validity, they are often expensive, time-consuming, and difficult to scale. These constraints limit the size of studies, the diversity of participants, and the reproducibility of results. Recent advances in large language models raise the question of whether AI can

responsibly support parts of the scientific process itself, particularly in evaluation and data annotation tasks.

My work in this area explores how AI can be integrated into research workflows without undermining scientific rigor. Rather than aiming to replace human judgment wholesale, this research investigates mixed human–AI approaches that balance scalability with validity, and that make the limitations of AI explicit rather than implicit.

A central contribution of my recent work is an empirical investigation into whether large language models can substitute for human annotators in software engineering research tasks. Many evaluations in our field involve subjective judgments, such as assessing the quality of code summaries, deciding whether a warning is actionable, or judging the intent of a code change. In this work, we systematically compared annotations produced by humans and by state-of-the-art language models across a range of software engineering datasets and tasks [3]. Our findings show that the suitability of AI as a proxy for human judgment depends strongly on the task, motivating selective and transparent use of AI rather than blanket automation.

This research contributes methodological guidance that is directly applicable to empirical software engineering, where evaluation cost is a persistent bottleneck, and it connects to broader work on reproducibility and transparency in scientific software. By treating AI as a research instrument whose behavior must be understood, validated, and reported, this work supports more repeatable and inspectable study designs.

Looking forward, my research in AI for science will focus on principled frameworks for mixed human–AI evaluation pipelines, on enabling large-scale and longitudinal empirical studies that are currently infeasible due to cost constraints, and on examining the epistemic implications of AI participation in scientific workflows. This includes clarifying assumptions, limitations, and reporting practices for AI-assisted research.

Overall, this research area positions AI as a tool for improving the practice of science itself. By carefully integrating AI into empirical workflows, my work aims to reduce practical barriers to high-quality empirical research while preserving the methodological standards that underpin credible scientific knowledge.

Selected Publications and Outputs

[1] Changwen Li, Christoph Treude, and Ofir Turel. Do comments and expertise still matter? An experiment on programmers' adoption of AI-generated JavaScript code. Journal of Systems and Software, 2026.

[2] Christoph Treude and Marco A. Gerosa. How Developers Interact with AI: A Taxonomy of Human-AI Collaboration in Software Engineering. Forge '25: Proceedings of the 2nd International Conference on AI Foundation Models and Software Engineering, 2025.

[3] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. Can LLMs Replace Manual Annotation of Software Engineering Artifacts?. MSR ‘25: International Conference on Mining Software Repositories, 2025.

[4] Yuan Jiang, Zhichen Qu, Christoph Treude, Xiaohong Su, and Tiantian Wang. Enhancing Fine-Grained Vulnerability Detection with Reinforcement Learning. IEEE Transactions on Software Engineering, 2025.

[5] Yuan Jiang, Shan Huang, Christoph Treude, Xiaohong Su, and Tiantian Wang. Shield Broken: Black-Box Adversarial Attacks on LLM-Based Vulnerability Detectors. IEEE Transactions on Software Engineering, 2026.

[6] Yuan Jiang, Zehao Li, Shan Huang, Christoph Treude, Xiaohong Su, and Tiantian Wang. Effective Code Membership Inference for Code Completion Models via Adversarial Prompts. ASE ‘25: International Conference on Automated Software Engineering, 2025.