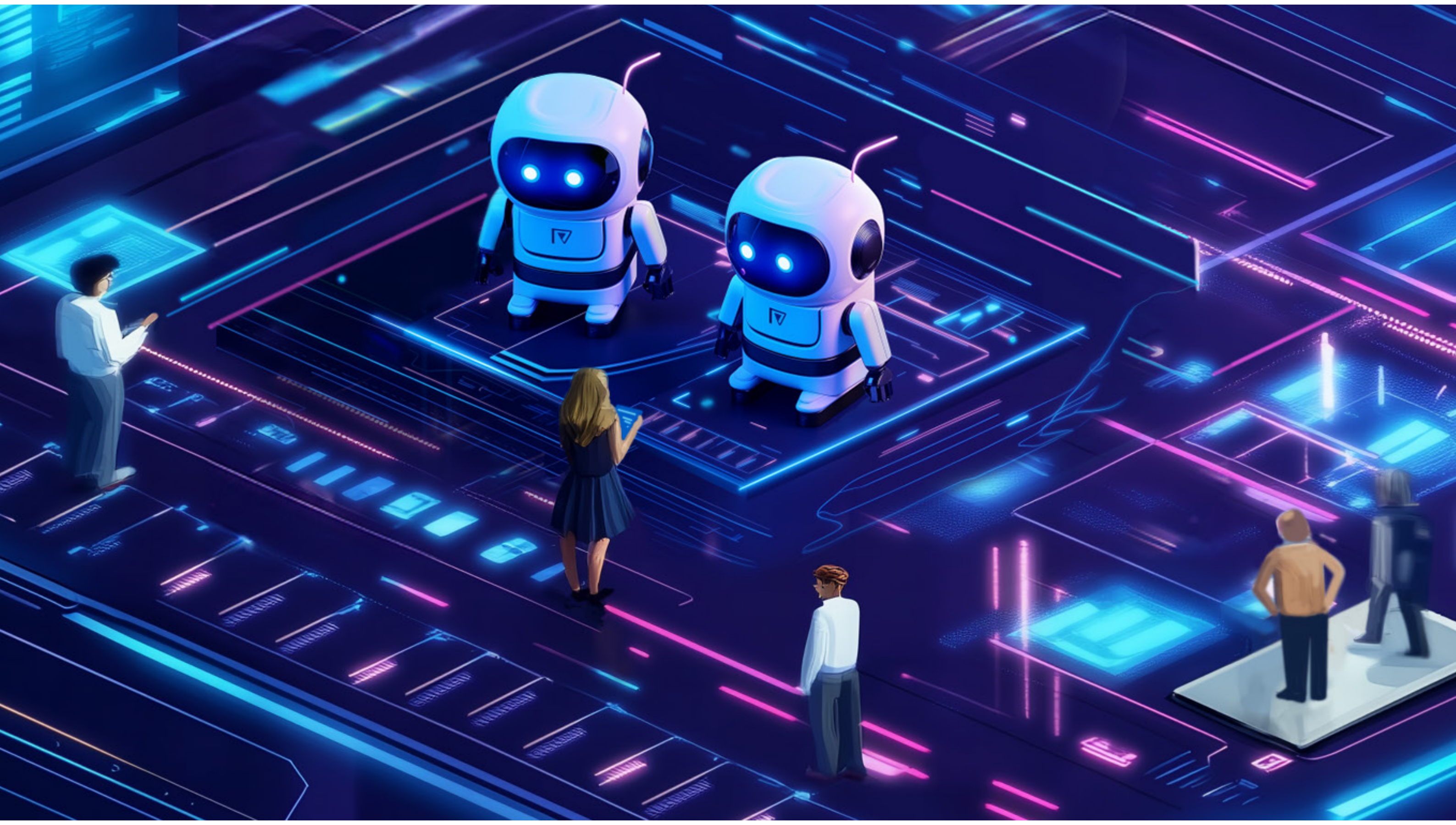
Crafting the AI Software Engineer of the Future

Abhik Roychoudhury[1], National University of Singapore, Singapore

[1]Abhik Roychoudhury is also a Senior Advisor at SonarSource



Provost's Chair Professor Abhik Roychoudhury examines AI's evolving role in software engineering, from LLM-driven automation to unified AI developers and human-AI collaboration.

Artificial intelligence (AI) technologies, particularly large language models, are transforming software engineering by automating coding tasks. However, software engineering extends beyond code generation to maintenance, architecture and collaborative development. The emergence of AI software engineers raises fundamental questions about their role, capabilities and integration into human software teams.

The Evolution of Software Engineering

Software Engineering stands at the intersection of science and engineering. It integrates principles from programming languages, formal methods and other fields to provide rigour in how we design, build, maintain and evolve software systems. Over the past fifty years, it has evolved into a discipline grounded in robust engineering practices — particularly in the management of large-scale software systems through continuous integration and continuous deployment (CI/CD).

Indeed, fifty years is a long time in computing. This milestone coincides with the rise of artificial intelligence (AI), particularly large language models (LLMs), which have demonstrated significant capabilities in automated coding. But software engineering is not just about coding. It is multifaceted — encompassing various aspects like design, maintenance, evolution, architecture and the broader human and social dimensions of software development.

Software engineering is not just about coding. It is multifaceted — encompassing various aspects like design, maintenance, evolution, architecture and the broader human and social dimensions of software development.

The crucial, and perhaps multi-million-dollar, question, then, is this: How will AI shape the role of software engineers beyond coding?

And what does it mean to be an AI software engineer?

AI Agents in Software Engineering

To envision the future of software engineering, we must examine its present trajectory. Significant efforts are underway to develop LLM agents capable of handling software engineering tasks such as bug fixes, feature additions and performance optimisations. But what precisely constitutes an agent, and to what extent does it fulfil the role of an AI software engineer?

At a fundamental level, an LLM agent for issue remediation follows a structured process (see Figure 1). When presented with a problem — be it a bug fix or a feature enhancement — the agent processes it via an LLM-based front-end. This, in turn, interacts with a back-end capable of invoking analysis tools, navigating program structures and generating potential solutions. The agent's goal is not only to propose patches, but also to validate them against tests and other software artifacts. This ensures quality and reliability.

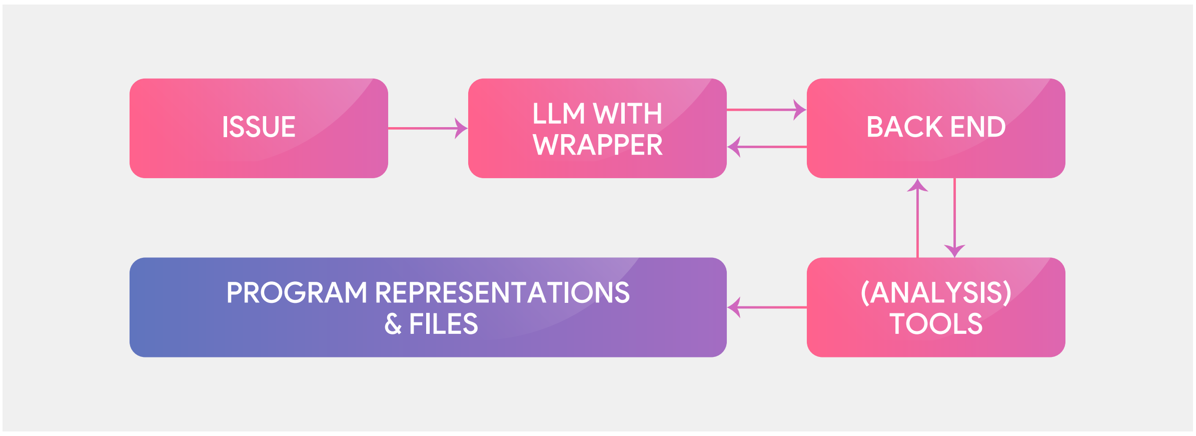


Figure 1: Thinking behind an LLM agent for issue remediation

AutoCodeRover:
A Step Towards AI Software Engineering

An LLM-based issue remediation agent embodies an early prototype of an AI software engineer. Many human software engineers spend their time reviewing issue trackers, resolving assigned problems and committing fixes to a codebase. If an AI system can autonomously remediate issues, it could reasonably be considered a “junior” AI software engineer.

AutoCodeRover [1,2] is a step change in this direction. Its architecture (see Figure 2) includes key capabilities essential for AI-driven software development. Given an issue, AutoCodeRover first reproduces the problem through test cases. It then retrieves relevant code through autonomous search, where it leverages program analysis tools to debug the issue. This process culminates in a code edit or patch, which is subsequently validated against various testing and review mechanisms before integration.

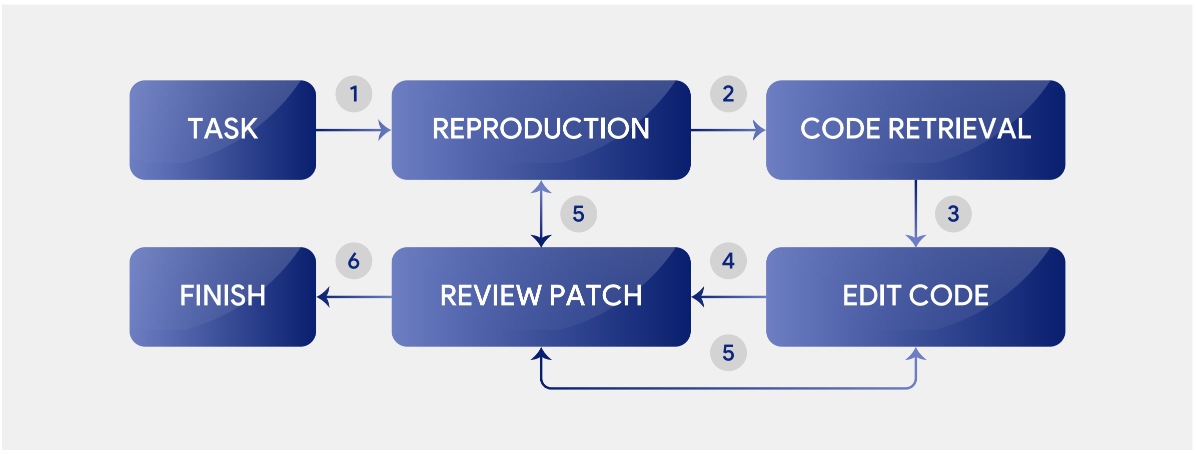


Figure 2: High-level design of AutoCodeRover

Towards a Unified AI Software Engineer

Where do we go from the current burst of interest in LLM agents? Will AI agents remain specialised tools, or will they morph into more comprehensive software engineers? How can we close the chasm between current AI capabilities and the many workflows managed by human counterparts?

Current AI-driven software development is fragmented, with different agents tackling specific tasks such as bug fixes, code generation and testing. The next logical step is to unify these capabilities into a comprehensive AI software engineer capable of handling complex software development scenarios.

A seasoned, human software engineer does far more than isolated tasks; they manage interdependent workflows. Consider the following scenarios:

Implementing a new feature while pre-emptively addressing potential bugs

Completing a fix that was initially incomplete, while ensuring robustness across the codebase

Taking over the code of a developer who has left the organisation, running tests to understand its functionality and then extending it with confidence

Current AI-driven software development is fragmented, with different agents tackling specific tasks such as bug fixes, code generation and testing. The next logical step is to unify these capabilities into a comprehensive AI software engineer capable of handling complex software development scenarios.

Achieving this level of competence requires moving beyond primitive task execution. AI software engineers will need to operate within a structured action space, wherein all tasks — debugging, patching, testing, refactoring — are accessible through a unified interface (see Figure 3). Such an approach enables AI agents to autonomously select and sequence actions while maintaining awareness of system state.

In the near future, we can reasonably anticipate unified AI agents functioning as junior members in software teams, working alongside human developers. Nevertheless, for widespread, and confident, adoption, it will require high-quality training datasets spanning real-world software engineering tasks, extending beyond current benchmarks such as SWE-bench [3].

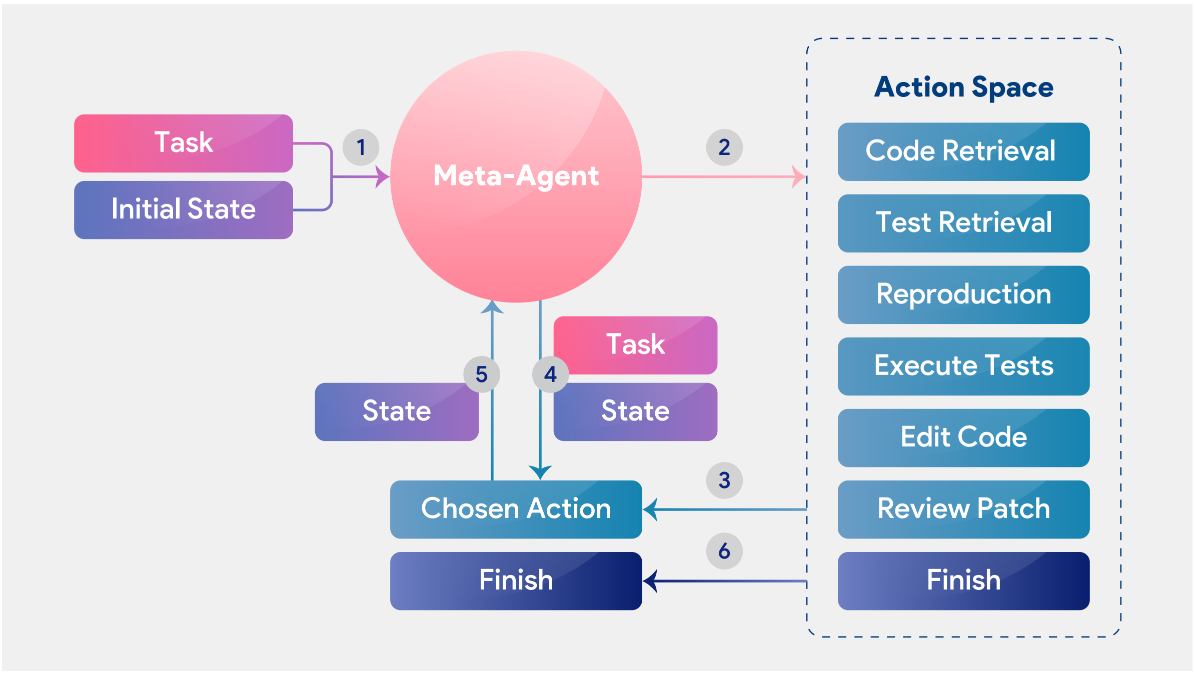


Figure 3: Thinking behind a Unified Software Engineering Agent

AI in Formal Methods and Verification

While AI-driven software engineering is gaining traction in activities such as maintenance and debugging, its potential extends to more formal facets of software engineering. For instance, mathematical verification of program properties. In this regard, can AI agents contribute meaningfully to formal verification?

Let's take a look at recent advancements such as Alphaproof [4], an AI system capable of solving International Mathematics Olympiad problems at a silver-medal standard. This suggests promising avenues. What implications does this have for those of us in software engineering? Is it merely an interesting lunch-time conversation to lazily partake in with our colleagues? Or does it have some lessons for us with respect to program verification?

If AI can construct mathematical proofs, might it also aid in program verification? One emerging approach involves integrating LLM-based agents with automated theorem provers in a feedback loop, where the AI proposes proofs, and formal tools validate them.

But what could be novel is the role of lemmas in proofs. For general mathematical proofs, a critical challenge lies in identifying the right lemmas — intermediate steps crucial to constructing a proof. If AI agents like AutoCodeRover can extend their code-search capabilities to intelligently and intuitively identify proof obligations, this could revolutionise program verification. Figure 4 illustrates a potential architecture for such AI-driven verification agents.

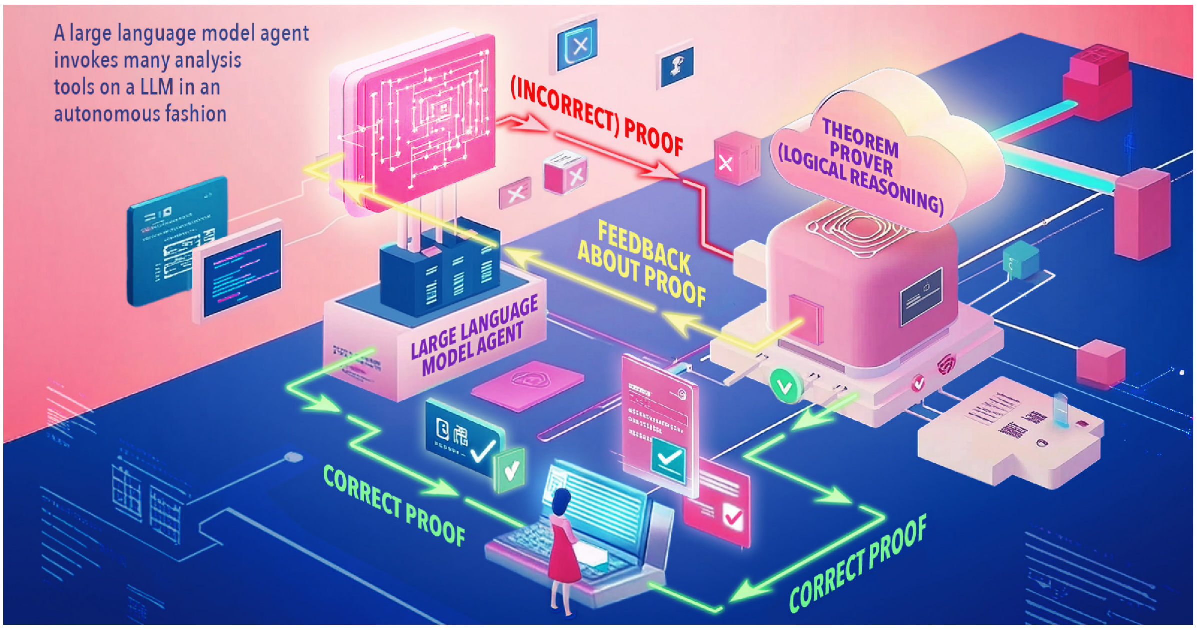

A large language model agent invokes many analysis tools on a LLM in an autonomous fashion
(INCORRECT) PROOF
THEOREM PROVER (LOGICAL REASONING)
FEEDBACK ABOUT PROOF
LARGE LANGUAGE MODEL AGENT
CORRECT PROOF
CORRECT PROOF

Figure 4: AI generated schematic for Program Verification Agents

The Future of AI in Software Teams

So far, I have explored the growing capabilities of individual AI-driven software engineers. However, as a human software engineer, my output is not entirely my output. It is also triggered by a conversation that I had with a colleague over a coffee break. It is also influenced by an architectural diagram another colleague showed me in a group meeting. In other words, real-world software development is inherently collaborative. A software engineer's work is not limited to siloed problem-solving — it is shaped and influenced by team discussions, informal exchanges, shared architectural insights and much more.

A software engineer's work is not limited to siloed problem-solving — it is shaped and influenced by team discussions, informal exchanges, shared architectural insights and much more.

The integration of AI into software teams raises questions about team dynamics. How will human developers interact with AI agents? How should AI systems collaborate with one another? Traditional cognitive theories of cooperative work, which focus on human goal influence, may not fully encapsulate the nuances of software-engineering collaboration [5]. Unlike environments where game-theoretic incentives prod behaviour, software development is driven by long-term design considerations, iterative improvements and architectural vision [6].

Addressing these challenges will be the next frontier in AI-driven software engineering. Before we can fully welcome AI software engineers into teams, we must first refine their individual competencies. The future of software engineering is ever-evolving — but the journey towards truly intelligent, collaborative AI engineers is only just beginning! ◆

Acknowledgements

I would like to acknowledge discussions with attendees at three recent meetings:
(a) Dagstuhl Seminar on Automated Programming and Program Repair (Oct 2024);
(b) Shonan Meeting on Trusted Automatic Programming (Jan 2025); and
(c) IFIP Working Group 2.4 meeting (Feb 2025).

References

[1] AutoCodeRover: Autonomous Program Improvement, Zhang, Ruan, Fan, Roychoudhury, International Symposium on Software Testing and Analysis, 2024.
[2] SpecRover: Code Intent Extraction via LLMs, Ruan, Zhang, Roychoudhury, International Conference on Software Engineering, 2025.
[3] SWE-bench: Can Language Models Resolve Real world GitHub issues? Jimenez et al, International Conference on Language Representations, 2024.
[4] AI achieves silver-medal standard solving International Mathematical Olympiad problems, Alphaproof and AlphaGeometry teams, Google Deepmind, 2024.
[5] Bayesian Theory of Mind: Modeling Joint Belief Desire Attribution, Baker, Jara-Ettinger, Saxe, Tennebaum, Annual Meeting of Cognitive Science Society, 2011.
[6] AI Software Engineer: Programming with Trust, Roychoudhury, Pasareanu, Pradel, Ray, [2502.13767] AI Software Engineer: Programming with Trust, 2025.