BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026
http://excel.fit.vutbr.cz

Filling the Gaps of AI Code Review in Enterprise

Rastislav Budinský*

Abstract
Commercial AI code reviewers find general bugs well, but they cannot enforce the in-house coding standards my team has built up over years of shipping. To fill that gap I designed and shipped a multi-agent review system in production at our company: specialized agents that each see only the rules in their own domain, a rule middleware that also serves those rules to colleagues using Claude Code, a code knowledge base that lets agents reason beyond the lines actually changed in a PR, and a deduplication pipeline that posts surviving comments straight into our Azure DevOps pull requests. Across 436 production pull requests over five months, the system contributed 1,104 actionable threads at a 41.8 % code-change rate; together with a commercial reviewer added later, total review coverage on our PRs roughly tripled compared with the human-only baseline (2.22 → 6.67 threads per PR). What my colleagues and I value the most—though I cannot rigorously measure it—is that human reviewers now spend more attention on logic and business questions rather than style.

*xbudin05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

[Motivation] The team I work on enforces a steadily-growing body of in-house standards: best practices learned from shipping production software for years, architecture and layering restrictions that keep the codebase maintainable as it grows, SQL conventions, and a lot of subtler rules that long-tenured engineers carry around in their heads. A surprising amount of this knowledge had never been written down before this thesis—you only learned about a rule when a senior reviewer caught you breaking it. Catching all those specific style and structural violations had become a tedious chore for already-overloaded human reviewers. Commercial AI reviewers such as CodeRabbit [1] do let you add custom rules, but I wanted to test whether a more tailored architecture—per-domain agents, the same rule documents available to the developer's Claude Code client, a codebase knowledge base—could enforce in-house standards more thoroughly than a single generic reviewer.

[Problem definition] I wanted an AI reviewer that *enforces the rules our team has actually agreed on*, precise enough that developers act on its comments and quiet enough that the tool stays trusted in our daily PR flow. The measure of success is concrete: how often a posted comment is followed by a code change on the flagged lines in our production pull requests.

[Existing solutions] Commercial tools [1, 2] handle broad bug detection and generic style well and accept custom rules; their internal agent architecture is not publicly documented. My own early attempt was a single agent handed all our rules, and as the ruleset grew I watched it enforce some rules carefully and quietly skip others—which is what pushed me toward splitting review work across per-domain agents.

[Our solution] I built *RIXO Code Reviewer*, in which review work is split across several specialized agents—for example a per-file *File Integrity* agent that checks that file structure and naming follow our conventions, and a per-PR *Architecture* agent that uses a stronger model and the knowledge base to check that a change respects our layering and dependency rules across files. The working hypothesis: giving each agent only the rules and context it needs keeps hallucination low and lifts per-rule detection above what a single generic reviewer can manage. A rule middleware (*rixo-dev-mcp*) serves the same rules to the agents and, via MCP, to colleagues using Claude Code; a knowledge base (*rixo-code-graph-rag*) lets agents reason about parts of the codebase the PR diff does not show.

[Contributions] This paper contributes (i) a specialized multi-agent architecture validated on 436 production PRs at a real company; (ii) a comparative evaluation against CodeRabbit and my own teammates' re-

views over the same five-month window; and (iii) a set of deployment lessons—especially how expensive it is to ship a noisy first version to the people you work with every day.

2. System

The architecture (Figure 1 on the poster) is three co-operating services.

rixo-code-reviewer runs the agents. Per-file agents work in parallel, each receiving only its domain's slice of the 30 rules grouped into 11 rule groups; the per-PR Architecture agent reads the whole change with help from the knowledge base. Specialization is not just engineering convenience: narrowing each agent's context structurally prevents it from hallucinating about rules it never sees.

rixo-dev-mcp is the rule middleware. Rules live as Markdown documents and are served both as a REST API to the agents and as an MCP server to my colleagues' Claude Code clients, so the same authoritative rules drive both sides of the workflow.

rixo-code-graph-rag is a graph-plus-vector knowledge base (Memgraph, Tree-sitter ASTs, dense embeddings) so agents can reason *outside the scope of the PR diff*—"how do other services handle this pattern?", "which classes implement this interface?"—which a raw diff cannot support.

Comment pipeline. New agent findings are not posted directly. A *Thread Manager* first walks every existing bot thread from earlier PR iterations to decide its lifecycle (replied, code changed, silently dismissed) and pre-loads silent dismissals as probable false positives to suppress similar future findings. Separately, a *two-tier deduplication engine* filters newly generated comments against the same file's existing threads: a vector-similarity tier rejects obvious duplicates and an LLM tier handles paraphrased or merged findings. Only what survives both is posted to Azure DevOps; Figure 2 on the poster shows the full comment pipeline.

3. Evaluation

The system ran continuously on my team's production codebase from November 2025 to March 2026, covering 436 PRs. From January 2026 I also logged CodeRabbit and my human teammates' threads for direct comparison. Effectiveness is the *code-change rate*: the fraction of actionable threads whose flagged region was modified in a later iteration of the same PR.

Coverage roughly doubled with the bot, tripled once CodeRabbit joined. Compared with the preceding year on the same codebase (501 PRs, human-only), human threads per PR stayed essentially flat (2.22 → 2.43). With my system added, the human-plus-RIXO total per PR was just over double the baseline (≈ 4.96), and only the addition of CodeRabbit pushed the combined three-source total to 6.67 threads per PR—the threefold figure quoted on the poster.

Table 1. Code-change rate on 436 production PRs. The RIXO system posts only actionable threads; 60.3 % of CodeRabbit's output is informative and excluded from its rate.

Reviewer	Threads	Actionable	Change rate
RIXO (ours)	1,104	1,104	41.8 %
CodeRabbit	744	295	51.2 %
Human reviewers	1,061	1,044	80.1 %

The qualitative win. In retrospectives the team and I noticed human reviewers spending less time on style nits and more on logic and domain correctness—informal but consistent feedback, and in our opinion the most valuable effect of the project.

Agent breakdown and trend. Code-change rate varies from 36 % (File Integrity) to 65 % (Documentation); see Figure 3 on the poster. The narrower the agent's domain, the more targeted its findings. RIXO's monthly change rate drifted from 32 % in November 2025 up to 50 % in March 2026 as I tightened rules and shipped deduplication.

4. Lessons Learned

Deploy quality from day one. My most expensive mistake was shipping early with both incomplete deduplication and rules that were still poorly defined. Colleagues learned to dismiss bot comments on sight—a habit that persisted long after I fixed the underlying quality, measurably dampening the change-rate trend.

Specialize, don't generalize. This is what turned the project serious. My first prototype was one agent handed every rule; as the rule set grew, the model enforced some rules carefully and quietly forgot others. Recognizing that growing-rule-set vs. hallucination trade-off is when I escalated from weekend experiment to thesis topic and committed to per-domain agents.

Complement, don't replace. Framing the system as complementing (not replacing) human reviewers—Table 1 on the poster shows who does what best—is what let all three coexist productively.

5. Conclusions

Five months in production, still used daily; human review attention has visibly shifted from style to logic.

Acknowledgements

Huge thanks to my supervisor doc. Ing. Ivan Homoliak, Ph.D., and to my colleagues at work who put up with a loud review bot in their PRs long enough for me to make it useful.

References

[1] CodeRabbit. CodeRabbit: AI code reviews for developers. https://coderabbit.ai/. Accessed: 2026-04-10.

[2] Anysphere. Cursor: The AI code editor. https://cursor.com/. Accessed: 2026-04-10.