CoRenameAgent: A Human-In-The-Loop, Multi-Agent Tool for Coordinated Rename Refactoring

Abhiram Bellur
University of Colorado
Abhiram.Bellur@colorado.edu

Mohammed Raihan Ullah
University of Colorado
raihan.ullah@colorado.edu

Fraol Batole
Tulane University
fbatole@tulane.edu

Mohit Kansara
University of Texas at Dallas
mohit.kansara@utdallas.edu

Masaharu Morimoto
NEC Corporation
m-morimoto@nec.com

Kai Ishikawa
NEC Corporation
k-ishikawa@nec.com

Haifeng Chen
NEC Laboratories America
haifeng@nec-labs.com

Helena Klause
JetBrains
helena.klause@jetbrains.com

Yaroslav Zharov
JetBrains Research
yaroslav.zharov@jetbrains.com

Timofey Bryksin
JetBrains Research
timofey.bryksin@jetbrains.com

Tien N. Nguyen
University of Texas at Dallas
tien.n.nguyen@utdallas.edu

Hridesh Rajan
Tulane University
hrajan@tulane.edu

Nikolaos Tsantalis
Concordia University
nikolaos.tsantalis@concordia.ca

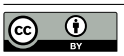
Danny Dig
University of Colorado
danny.dig@colorado.edu

Abstract

Coordinated renaming, where a single rename refactoring triggers refactorings in multiple, related identifiers, is a frequent yet challenging task. Developers must manually propagate rename refactorings across numerous files and contexts, a process that is both subjective and tedious. Deciding which names should change depends on developers' opinions about the code-base. We present CoRenameAgent, a multi-agent, interactive tool working in tandem with developers, to assist with *coordinated renaming*. CoRenameAgent operates alongside the developer, combining automated analysis with human oversight to ensure contextually appropriate renaming.

We developed CoRenameAgent as a plugin for IntelliJ-IDEA, a leading IDE for the Java programming language, and designed it with usability and practical adoption in mind. To understand the real-world applicability of CoRenameAgent, we conducted a case study on a popular open-source project, where CoRenameAgent successfully replicated 90% rename operations previously performed by developers. To quantify the developer effort required to use the agent, we conducted a pilot study with 5 users. On average, reviewing CoRenameAgent's 12 suggestions required 1.5 minutes of developer time per task, demonstrating that the agent can significantly reduce manual workload while keeping developers in control. A screencast of CoRenameAgent can be found on YouTube[1].

[1]https://www.youtube.com/watch?v=C4GGvCHEf5M



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3806418

1 Introduction

Recent advances in Large Language Models and autonomous systems have sparked significant interest in using software agents to support and transform software engineering workflows. These agents act as proactive, goal-directed collaborators and have demonstrated promising results across a range of tasks, including code synthesis [13], test generation [3, 14], and automated program repair [9, 23, 32]. In these settings, agentic systems can substantially reduce developer effort by autonomously producing artifacts that can be validated using objective criteria such as compilation success or test-case satisfaction.

However, most existing agentic workflows adopt a request-and-response model that positions the developer as a passive validator of an agent's final output. While effective for tasks with objective correctness criteria, this model is ill-suited for subjective, preference-sensitive activities—such as naming or readability-oriented refactoring—that lack a single correct solution and cannot be validated by tests alone. These tasks encode developer intent, domain semantics, and project-specific conventions, and therefore require mixed-initiative workflows in which agents and developers collaboratively shape outcomes. For such settings, agents should support iterative exploration and refinement, keeping the developer meaningfully in the loop rather than sidelined by full autonomy.

In this paper, we address how humans and AI-agents can collaborate, in the context of identifier renaming. Renaming an identifier is a common refactoring that updates the name of a program element and all its references and/or call sites within the program's scope. Renaming is among the top-five most frequently performed refactorings [19, 20, 29], both manually and with automated tools. However, a significant portion of renames involve ***coordinated renaming***, a tedious task that requires renaming multiple distinct identifiers to

preserve semantic consistency. For example, if developers decide to rename the field `CustomerAccount` to `ClientAccount`, they must also rename the method `processCustomerProfile` to `processClientProfile` to maintain conceptual consistency. Previous research on the prevalence of *coordinated renaming* [21] analyzed 1M renaming instances from open-source commits and found that 57% of the renamings were done in a coordinated manner. On average, 5 renames were performed together. Further, we found commits in famous open-source projects where *coordinated renaming* triggered 919 distinct rename operations [16]. *Coordinated renaming* is the kind of repetitive task in which LLM-based agents can significantly reduce the software developer's effort.

We present CoRenameAgent, a novel multi-agent framework that automates *coordinated renaming* in a new refactoring workflow, while keeping the developer in the driver's seat. A developer initiates a rename refactoring in the IDE that serves as a signal of refactoring intent. An agent observes this signal, curates a scope of renaming, and propagates rename-changes across the entire codebase after obtaining human approval. When the developer rejects the agent's unfavourable suggestions, the agent refines its refactoring scope with the help of the developer. The agent coordinates with IDE-based refactoring tools to safely and precisely apply changes across the codebase. Rather than acting in isolation, these agents work alongside **developers** and **automated program analysis tools** (such as refactoring engines) to create an intelligent, collaborative environment.

In this new paradigm, the role of the developer shifts from searching, remembering, and executing refactorings to reviewing refactorings proposed by an agent and guiding the agent to align with their underlying intent. This new paradigm removes a key barrier to adopting AI agents – the fear of losing control and ownership [17, 25] – by redefining the developer's role from direct executor to supervisor of the agent's work.

We iterated on CoRenameAgent's design using pilots and discovered key design principles that would enable the adoption of CoRenameAgent: transparency, non-disruptive integration, and accountability. Next, we conducted a case study on the Apache Flink project, using CoRenameAgent to replicate real refactoring scenarios and successfully automate 90% of the renaming operations performed by open-source developers. Finally, we conducted a pilot study with 5 users to understand the developer effort required to use the tool. We found that CoRenameAgent provides an average of 12 suggestions for review. Reviewing all suggestions requires of 1.5 minutes of active developer effort. CoRenameAgent requires $0.5 to execute per task, and runs for a total of 5 minutes.

CoRenameAgent is available for download from our companion website [24]. A video screencast, demonstrating how to use CoRenameAgent is on YouTube [2]. Our full paper [8] presents the approach, a formative study, a benchmark and a thorough evaluation of CoRenameAgent complementary to this submission.

2 CoRenameAgent

2.1 Workflow and UI Design

In this section, we describe the workflow and UI design of CoRenameAgent through an example of renaming a field `duration` to `durationMillis` to clarify its unit of measurement. As illustrated in Figure 1, the workflow begins when CoRenameAgent observes an initial rename operation performed by the developer in the IDE (①). The developer changes the name of field `duration` to `durationMillis`. Upon detection, CoRenameAgent displays a pop-up notification requesting to start a *coordinated renaming* session.

Once triggered, CoRenameAgent analyzes the developer's refactoring to infer a broad refactoring scope, represented as an explicit specification called the *Declared Scope*. The *Declared Scope* consists of two parts: the renaming pattern and a *guard* (the code-context for application). Under the hood, the Scope Inference Agent, infers a pattern `duration` → `durationMillis` with an initially empty guard. To reduce UI complexity, this scope is hidden by default but remains accessible for developer inspection.

CoRenameAgent then searches for matching identifiers across the project, by utilizing two agents in the background – Planned Execution Agent and Replication Agent. The Planned Execution Agent looks for renaming opportunities within a given file, and the Replication Agent searches through the project to discover files which should be changed. While processing, the tool's UI displays an animated spinner to indicate background activity. The agent's tool window also displays a status bar indicating progress – the number of files and suggestions inspected. When the agent finishes its analysis, the agent's spinner stops, and a second spinner (corresponding to the developer) begins moving, to indicate that the developer's review is required.

As shown in Figure 1, CoRenameAgent presents rename suggestions for review (2A and 2B). The UI highlights the suggested change and provides a *rationale* explaining why the name should be updated. In our example, CoRenameAgent identifies the `durationDiff` method and suggests renaming it to `durationMillisDiff`. The developer can either **Accept** or **Reject** the suggestion. If accepted, CoRenameAgent triggers an IDE rename operation to propagate the change safely (③).

If a suggestion is invalid—such as the `duration` (2B) variable which holds seconds instead of milliseconds—the developer chooses to **Reject** (④). This causes CoRenameAgent to refine the *Declared Scope* (⑤). Under the hood, Scope Inference Agent analyses the rejected suggestion in its surrounding context and proposes a refined guard: "perform renames when `duration` holds a time-value in milliseconds." The refined scope is presented in editable text-boxes, allowing the developer to tune the logic before clicking **Approve Refined Scope** (⑥). Steps ② - ⑥ repeat until CoRenameAgent finds no further suggestions matching the refined *Declared Scope*.

2.2 Implementation

We implemented CoRenameAgent as an IntelliJ IDEA plugin to ensure ease of adoption and seamless integration into developers' existing workflows. In addition to the plugin, we distribute a Docker container that encapsulates a Python-based implementation of our multi-agent system. This separation enables rapid development of agentic behaviours in Python, while delegating language- and IDE-specific tooling to IntelliJ.

The IntelliJ plugin is responsible for: (1) interaction with the developer, and (2) providing mature and deterministic tools to the agentic framework. To interact with the developer, we use a tool

[2] https://www.youtube.com/watch?v=C4GGvCHEf5M

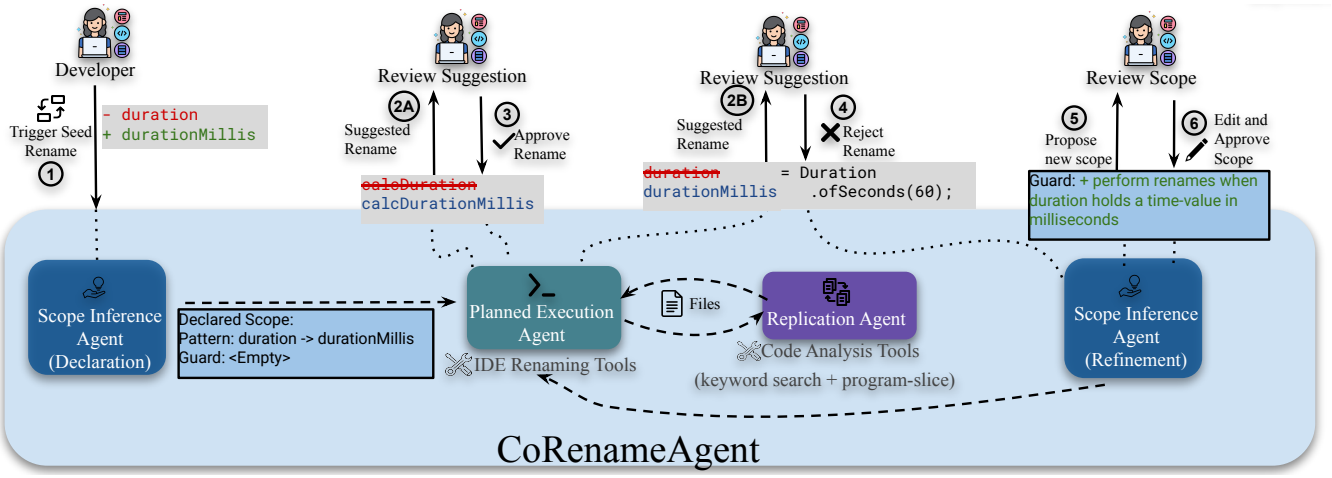


Figure 1: Workflow of CoRenameAgent

window to display renaming suggestions and update the status of the agent. We implement listeners to IntelliJ's existing rename-refactoring workflow, which launch pop-up notifications when a rename is executed by a developer. If a developer chooses to start a *coordinated renaming* session, the plugin triggers the agent by spawning a Docker container.

In order to provide tools to the agent, the plugin spawns a local HTTP server and exposes a set of IDE-level capabilities including identifier renaming, static program context, and program slicing. CoRenameAgent executes rename refactorings via IntelliJ's built-in refactoring APIs, ensuring correct propagation of identifier changes across files, as well as support for safe rollbacks. The agent extracts static program context such as method signatures, type information, references, and usage locations using IntelliJ Platform's Program Structure Interface (PSI). The agent uses program slices provided by interfacing with IntelliJ's Data Flow Analysis framework – enabling precise tracking of data dependencies for identifiers, including fields, parameters, and local variables.

The Agentic Framework is implemented in Python, which leverages the LangChain [10] and LangGraph [27] libraries. These libraries provide reusable features such as prompt templates, easy integration with various LLMs, and the ability to define the agent as a graph. The agents communicate with each other via an episodic memory [26], which we implemented as a SQL database. To execute rename-refactorings or fetch static context, the agent communicates with the plugin via HTTP requests. We monitor and debug the agentic execution using LangSmith [28], which provides fine-grained telemetry including token usage, LLM invocation traces, latency, and estimated cost. All LLM-driven components use OpenAI's `o4-mini` model for its reasoning capabilities with default parameters. We encapsulate the entire agentic framework in a Docker image. This ensures portability across different environments.

For full details on the internal design of CoRenameAgent, see our research paper [8].

3 Usability of CoRenameAgent

To ensure CoRenameAgent fits into development workflows, we iterated over CoRenameAgent's design, starting with a bare-bones version of CoRenameAgent with the workflow presented in Figure 1. We demonstrated the tool to pilot users, and refined the tool based on their feedback, understanding important design principles while building agents. Next, we performed a case-study of using CoRenameAgent on 10 *coordinated renaming* scenarios from the Apache Flink project to understand the application of CoRenameAgent in real-world scenarios. Finally, we ran a pilot study with 5 users who applied CoRenameAgent on diverse renaming scenarios and collected telemetry data to understand its operational cost from a developer's perspective.

3.1 Design Evolution through Feedback

Early user feedback revealed several areas where CoRenameAgent's initial design was lacking. Below, we discuss the major changes we made to CoRenameAgent based on the feedback.

Principle #1: Need for Insight and Transparency. Users expressed uncertainty about the internal state of CoRenameAgent and were unsure when their intervention was expected. Early prototypes of CoRenameAgent relied on generic status messages (e.g., "Analyzing file…"), which several users described as unhelpful. Participants reported difficulty distinguishing between a long-running analysis and a stalled or hung process.

Solution: We replaced generic status messages with real-time, fine-grained status updates – including the filename under inspection, the specific identifiers that were being analysed, and which keywords were currently being searched for. We also introduced two animated spinners: one indicating when the agent was actively running, and another signaling when user intervention was required. We used progress bars to provide continuous visual feedback about what percentage of files and identifiers were analysed. We found that transparency is critical for agentic tools, especially for long-running or exploratory tasks. Transparency helps maintain

user trust, reduces anxiety during delays, and supports smoother turn-taking between human and agent.

Principle #2: Non-disruptive Integration. In an initial version of CoRenameAgent, the agent automatically opened, navigated, and highlighted sections in files associated with a proposed renaming. While intended to be helpful, users reported that this behaviour was disruptive, as it did not allow them to switch contexts while waiting for the agent to finish.

Solution: We redesigned the interaction to be explicitly user-driven. Renaming suggestions are now highlighted only when the user clicks on them. Additionally, we augmented each suggestion with a short natural-language **rationale** explaining why the rename was proposed, enabling users to quickly assess its validity without navigating away from their current context. We found that effective developer tools should preserve, rather than interrupt a programmer's flow.

Principle #3: Accountability and Review. After a *coordinated renaming* sessions, users reported difficulty recalling which changes had been made by the agent and expressed a desire for a consolidated overview.

Solution: We presented a **Refactoring Report** that summarizes all renaming actions performed during a session, including affected identifiers and files. This report can be reviewed afterward. Providing an audit trail improves the accountability of agentic systems.

3.2 Case Study – Apache Flink

We tested CoRenameAgent's real-world applicability by replicating 10 *coordinated renamings* performed by developers from the Apache Flink [5] project. Across the 10 *coordinated renamings*, developers modified a total of 185 identifiers. We replicated each *coordinated renaming* with CoRenameAgent and compared its performance against three baselines: (1) a *Vanilla LLM* setup using `gpt-4o-mini`, where a response to a single prompt is directly applied to the codebase; (2) RENAS [11], a static-analysis tool specifically designed for *coordinated renaming*; and (3) `ClaudeCode` [4], a state-of-the-art coding agent for general-purpose programming tasks. We calculated recall as the number of developer-renamed identifiers reproduced by the tool, and precision as the number of extra identifiers renamed.

Table 1: Comparison of CoRenameAgent with baselines on 10 *coordinated renamings* from Apache Flink.

Tool	Recall	Precision	Avg. Files Changed	Avg. Cost / Task
Vanilla LLM	9 / 185 (5%)	50%	1	$0.005
RENAS [11]	42 / 185 (23%)	42%	3	-
`ClaudeCode`	55 / 185 (30%)	33%	4	$0.6
CoRenameAgent	168 / 185 (90%)	71%	10	$0.5

As shown in Table 1, CoRenameAgent outperformed all baselines. We attribute `ClaudeCode`'s lower recall to its limited ability to search the project for all relevant renaming candidates. Interestingly, the vanilla LLM achieved higher precision than RENAS and `ClaudeCode`, suggesting that limiting edits to a single file produces more focused results. At the same time, this contrast highlights the need for project-wide exploration to perform *coordinated renaming* effectively. For detailed, comparative analysis of CoRenameAgent with these approaches, look at [1].

3.3 Cost to Developer

Finally, we conducted a pilot study with 5 participants to estimate the developer effort and monetary cost associated with using CoRenameAgent. To do so, we collected anonymous telemetry data from users' refactoring sessions. The participants invoked the tool on 28 different *coordinated renamings*.

On average, CoRenameAgent presented 12 candidate renaming suggestions for developers to review, of which 11 suggestions were accepted, and 1 was rejected. Reviewing each suggestion took approximately 8 seconds, resulting in an average of 1.5 minutes of active developer effort per *coordinated renaming* session. Independently, CoRenameAgent required 3.5 minutes to complete its analysis, yielding a total end-to-end time of 5 minutes per refactoring task (including both developer effort and tool execution).

When using OpenAI's `o4-mini` model as the backend LLM, the average cost per task was 12 cents. In return for this cost, CoRenameAgent automated a substantial portion of the refactoring process: the agent analyzed 11 files, and proposed coordinated rename refactorings consistent with the underlying conceptual change.

4 Related Work

Identifier renaming is one of the most frequent refactorings in modern codebases [11, 15, 21]. Prior techniques focus on recommending improved names for individual identifiers [2, 12, 15]. In contrast, our work targets *coordinated renaming*, which aims to identify groups of semantically related identifiers that should be renamed together. Our approach complements existing techniques by focusing on where renaming should propagate, not just what name to choose.

Discovering *coordinated renaming*. *Coordinated renaming* focuses on identifying which identifiers should be renamed together. Empirical studies show that more than half of renames affect multiple elements [21], motivating automated support. Prior works based on static-analysis, RENAS [11] and RenameExpander [18], do not involve the human in the loop, and provide all renaming suggestions at once. CoRenameAgent, by contrast, incorporates and adapts based on feedback from the developer.

Refactoring with LLMs and Agents. LLMs have been used for suggesting extract/move method refactorings (e.g., EM-Assist [22], MM-Assist [7]). Multi-agent systems like Mantra [31], ISmell [30], and LocalizeAgent [6] perform refactorings such as smell removal or suggest design improvements. However, these systems propose new edits, while CoRenameAgent propagates renames already initiated by the developer.

5 Conclusions

By positioning the developer as a supervisor and reviewer, our framework resolves a major barrier to adopting AI assistants – the fear of losing control. CoRenameAgent contributes a methodological shift. It shows that agentic refactoring systems can achieve real-world impact when they operate through trusted IDE refactoring APIs and respect the boundaries of human authority.

References

[1] 2025. Raw Data from Flink Case Study. https://docs.google.com/spreadsheets/d/1B-aC5Nf26gthTIRBe9uCgv4EoKga_buBPRnyrt1iFi4/edit?usp=sharing. Accessed: 2026-01-21.

[2] Surafel Lemma Abebe and Paolo Tonella. 2013. Automated identifier completion and replacement. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 263–272.

[3] Juan Altmayer Pizzorno and Emery D. Berger. 2025. CoverUp: Effective High Coverage Test Generation for Python. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE128 (June 2025), 23 pages. doi:10.1145/3729398

[4] Anthropic. 2025. Claude Code. https://www.anthropic.com/claude-code. Accessed: 2025-10-08.

[5] Apache Software Foundation. 2024. Apache Flink – Stateful Computations over Data Streams. https://github.com/apache/flink/. Accessed: 2025-01-21.

[6] Fraol Batole, David OBrien, Tien Nguyen N., Robert Dyer, and Hridesh Rajan. 2025. An LLM-Based Agent-Oriented Approach for Automated Code Design Issue Localization. In *2025 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, ICSE.

[7] Abhiram Bellur, Fraol Batole, Malinda Dilhara, Mohammed Raihan Ullah, Yaroslav Zharov, Timofey Bryksin, Kai Ishikawa, Haifeng Chen, Masaharu Morimoto, Shota Motoura, Takeo Hosomi, Tien N. Nguyen, Hridesh Rajan, Nikolaos Tsantalis, and Danny Dig. 2025. Together We Are Better: LLM, IDE and Semantic Embedding to Assist Move Method Refactoring. In *International Conference on Software Maintainance and Evolution (ICSME)*.

[8] Abhiram Bellur, Mohammed Raihan Ullah, Fraol Batole, Mohit Kansara, Masaharu Morimoto, Kai Ishikawa, Haifeng Chen, Yaroslav Zharov, Timofey Bryksin, Tien N. Nguyen, Hridesh Rajan, and Danny Dig. 2026. Multi-Agent Coordinated Rename Refactoring. arXiv:2601.00482 [cs.SE] https://arxiv.org/abs/2601.00482

[9] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2024. RepairAgent: An Autonomous, LLM-Based Agent for Program Repair. arXiv:2403.17134 [cs.SE]

[10] Harrison Chase. 2022. LangChain. https://github.com/langchain-ai/langchain.

[11] Naoki Doi, Yuki Osumi, and Shinpei Hayashi. 2024. RENAS: Prioritizing Co-Renaming Opportunities of Identifiers. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 562–573.

[12] Chunhao Dong, Yanjie Jiang, Nan Niu, Yuxia Zhang, and Hui Liu. 2024. Context-Aware Name Recommendation for Field Renaming. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[13] Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madan Musuvathi, and Shuvendu Lahiri. 2024. Exploring the Effectiveness of LLM based Test-driven Interactive Code Generation: User Study and Empirical Evaluation. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 390–391. doi:10.1145/3639478.3643525

[14] Christopher Foster, Abhishek Gulati, Mark Harman, Inna Harper, Ke Mao, Jillian Ritchey, Hervé Robert, and Shubho Sengupta. 2025. Mutation-Guided LLM-based Test Generation at Meta. arXiv:2501.12862 [cs.SE] https://arxiv.org/abs/2501.12862

[15] Patricia Jablonski and Daqing Hou. 2007. CReN: A tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*. 16–20.

[16] Alexey Kudravtsev. 2025. cleanup: rename file to psiFile to distinguish from VirtualFile. https://github.com/JetBrains/intellij-community/commit/6d1f55f16043ad48f67762c50ab1035e0a589ca7. Accessed: 2025-07-16.

[17] Aayush Kumar, Yasharth Bajpai, Sumit Gulwani, Gustavo Soares, and Emerson Murphy-Hill. 2025. How Developers Use AI Agents: When They Work, When They Don't, and Why. arXiv:2506.12347 [cs.SE] https://arxiv.org/abs/2506.12347

[18] Hui Liu, Qiurong Liu, Yang Liu, and Zhouding Wang. 2015. Identifying renaming opportunities by expanding conducted rename refactorings. *IEEE Transactions on Software Engineering* 41, 9 (2015), 887–900.

[19] Emerson Murphy-Hill, Chris Parnin, and Andrew P. Black. 2012. How We Refactor, and How We Know It. *TSE* (2012). doi:10.1109/TSE.2011.41

[20] Stas Negara, Nicholas Chen, Mohsen Vakilian, Ralph E. Johnson, and Danny Dig. 2013. A Comparative Study of Manual and Automated Refactorings. In *ECOOP*.

[21] Yuki Osumi, Naotaka Umekawa, Hitomi Komata, and Shinpei Hayashi. 2022. Empirical study of co-renamed identifiers. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 71–80.

[22] Dorin Pomian, Abhiram Bellur, Malinda Dilhara, Zarina Kurbatova, Egor Bogomolov, Andrey Sokolov, Timofey Bryksin, and Danny Dig. 2024. EM-Assist: Safe Automated ExtractMethod Refactoring with LLMs. In *FSE*.

[23] Shanto Rahman, Sachit Kuhar, Berk Cirisci, Pranav Garg, Shiqi Wang, Xiaofei Ma, Anoop Deoras, and Baishakhi Ray. 2025. UTFix: Change Aware Unit Test Repairing using LLM. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 85 (April 2025), 26 pages. doi:10.1145/3720419

[24] ReplicationPackage [n. d.]. CoRenameAgent replication package. https://renameagent.netlify.app/.

[25] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. 2025. Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology* 178 (2025), 107610. doi:10.1016/j.infsof.2024.107610

[26] Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2023. Cognitive Architectures for Language Agents. arXiv:2309.02427 [cs.AI]

[27] LangChain Community / LangChain Team. 2025. LangGraph. https://github.com/langchain-ai/langgraph. Open-source graph-based orchestration library for LLM workflows.

[28] LangChain Community / LangChain Team. 2026. LangSmith. https://smith.langchain.com/. The platform for agent engineering.

[29] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. 2022. RefactoringMiner 2.0. *TSE* (2022). doi:10.1109/TSE.2020.3007722

[30] Di Wu, Fangwen Mu, Lin Shi, Zhaoqiang Guo, Kui Liu, Weiguang Zhuang, Yuqi Zhong, and Li Zhang. 2024. iSMELL: Assembling LLMs with Expert Toolsets for Code Smell Detection and Refactoring. In *ASE*.

[31] Yisen Xu, Feng Lin, Jinqiu Yang, Nikolaos Tsantalis, et al. 2025. Mantra: Enhancing automated method-level refactoring with contextual rag and multi-agent llm collaboration. *arXiv preprint arXiv:2503.14340* (2025).

[32] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1592–1604.