Context Engineering for AI Agents in Open-Source Software

Seyedmoein Mohsenimofidi
Heidelberg University
Heidelberg, Germany
s.mohsenimofidi@uni-heidelberg.de

Matthias Galster
University of Bamberg
Bamberg, Germany
mgalster@ieee.org

Christoph Treude
Singapore Management University
Singapore, Singapore
ctreude@smu.edu.sg

Sebastian Baltes
Heidelberg University
Heidelberg, Germany
sebastian.baltes@uni-heidelberg.de

Abstract

GenAI-based coding assistants have disrupted software development. The next generation of these tools are agent-based, operating with more autonomy and potentially without human oversight. One challenge is to provide AI agents with sufficient context about the software projects they operate in. Like human developers, AI agents require contextual information to develop solutions that are in line with the target architecture, interface specifications, coding guidelines, standard workflows, and other project-specific policies. Popular AI agents for software development, such as *Claude Code* and *OpenAI Codex*, advocate for maintaining tool-specific version-controlled Markdown files that cover aspects such as the project structure, building and testing, code style, and other details of the project. The content of these files is automatically added to each prompt. `AGENTS.md` has emerged as a potential standard that consolidates tool-specific formats. However, little is known about whether and how developers adopt this format. Therefore, in this work-in-progress paper, we present the results of a preliminary study investigating the adoption of AI configuration files in 466 open-source software projects, what information developers provide in `AGENTS.md` files, how they present that information, and how the files evolve over time. Our findings indicate that there is no established structure yet for organizing content in `AGENTS.md` files, and that there is a lot of variation in terms of how context is provided (descriptive, prescriptive, prohibitive, explanatory, conditional). We see great potential in studying which modifications in structure or presentation can positively affect the quality of the generated content. Finally, our analysis of commits that have modified `AGENTS.md` files provides first insights into how projects continuously extend and maintain `AGENTS.md` files. We conclude the paper by outlining how the adoption of AI configuration files in open-source software projects provides a unique opportunity to study real-world prompt and context engineering.

Keywords

Software Engineering, Generative AI, AI Agents, Mining Software Repositories, Open-Source Software

1 Introduction

The release of GitHub Copilot in 2021 and ChatGPT in 2022 launched a fundamental shift in how software is being developed. Today, generative AI (GenAI) tools built around large language models (LLMs) support software engineers throughout the software development lifecycle (SDLC)—although there is still a strong focus on code and test generation tasks [8, 17, 20, 32, 34, 39]. The *Devin AI* demo published in March 2024 fueled the first hype around agent-based software development [21], but it took until 2025 for agent-based software development to reach considerable adoption. In February 2025, Anthropic released *Claude Code*, a "*command line tool for agentic coding*" [2], which heralded the next evolutionary step in AI-assisted software development, enabling greater autonomy by allowing developers to assign coding tasks to AI agents via a terminal interface. Human oversight is still built-in, but can be turned off by the developer.

An advantage of agent-based tools is that they allow developers to provide context to LLMs in a fine-grained and targeted manner [16]. The deliberate process of designing, structuring, and providing relevant information to LLMs is referred to as *context engineering* [24, 28]. While *prompt engineering* focuses specifically on instructions and output indicators [7, 27], *context engineering* focuses on collecting and selecting input data for specific tasks, including relevant guidelines, configuration files, documentation, and exemplary code snippets [24, 28].

One way to "engineer" prompts and context is to add machine-readable AI configuration files to source code repositories. AI agents then automatically add the content of these files to their prompts. While traditional `README` files are written for human developers, providing quick start guides and project descriptions, AI configuration files are explicitly designed for AI agents, providing a machine-readable central source of contextual and procedural knowledge. Their content can include everything from terminal commands to build and test the project over links to documentation resources,
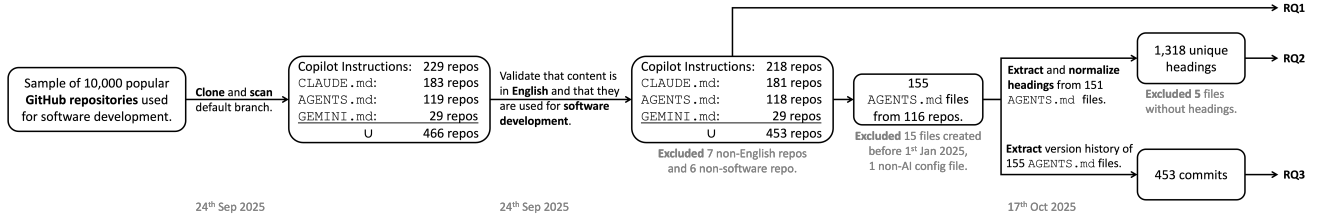
Figure 1: Data collection process.

common workflows, coding and naming conventions, instructions for creating pull requests, security considerations, and much more.

`AGENTS.md` was introduced as an open, tool-agnostic convention for such AI configuration files [1]. OpenAI's *Codex* tool relies on this format [26], while *Claude Code* by default searches for a file named `CLAUDE.md`. Anthropic's best-practice guide recommends teams to put that file into version control so that all team members benefit from consistent AI behavior [3]. GitHub introduced a similar AI configuration file for *Copilot* named `copilot-instructions.md` [11].

Since prompts are only rarely preserved after content has been generated [31], AI configuration files offer a unique opportunity to study how developers customize AI agents to their needs, what information they consider relevant to include, how they present it, and how the instructions and contextual descriptions evolve. To our knowledge, we present the first empirical study that analyzes configuration files used to guide AI agents in open-source software projects. Important unanswered questions include:

RQ1 *How widely have open-source software projects adopted AI configuration files?*

RQ2 *What information do open-source developers provide in AI configuration files and how do they present it?*

RQ3 *How do AI configuration files evolve over time?*

An initial search suggests that tens of thousands of GitHub repositories already contain AI configuration files [14]. However, there is no systematic analysis of these files yet that focuses on "engineered" software projects. This work-in-progress paper provides a first step toward filling this gap by mining GitHub repositories to study how AI configuration files are adopted, structured, and maintained, with the overarching goal of understanding how software teams engage in prompt and context engineering in practice. The results of our preliminary study are based on data collected from 10,000 GitHub repositories (**RQ1**). For **RQ2** and **RQ3**, we performed a detailed qualitative analysis of relevant repository data including file content, commits, and issues. As the first study that explores the above aspects, we follow an exploratory bottom-up approach to build an initial understanding of AI configuration files as novel software artifacts. We will extend our study in the future to answer the research questions more holistically.

2 Related Work

Agentic AI introduces autonomous decision-making and proactive problem-solving along the SDLC [18, 30]. Fueled by advances in LLMs, reinforcement learning, and multi-agent frameworks, agent-based AI enables the implementation of software agents that move beyond simple prompt-response interactions [36]. One of the first agent-based software development tools was *Devin AI* [35], which allowed agents to search the web, edit files, and execute commands to complete tasks iteratively and independently. In the academic community, *SWE-agent* [37] enabled LLM agents to communicate with the repository environment by reading, modifying, and executing bash commands [37]. Another example is *AutoCodeRover* [38], which enabled LLM agents to access code search APIs that help them find methods within specific classes for bug location identification [38]. Suri et al. showcase the potential of autonomous agents, particularly *Auto-GPT*, in software engineering tasks and demonstrates the importance of context-specific prompts for precise results in complex frameworks [30]. However, they also reveal that *Auto-GPT* [15], despite performing well on simpler tasks, struggles with ambiguity and complexity, underscoring the need for accurate context. Although context plays a critical role in guiding autonomous agents, prompts are typically treated as temporary artifacts and are rarely preserved or reused [22, 23, 33]. This lack of prompt management limits reproducibility [5] and underscores the importance of making prompt and context information explicit and manageable by using versioned AI configuration files. To the best of our knowledge, no prior work has investigated configuration files for agentic AI as novel software artifacts.

3 Data Collection

To answer our research questions, we collected real-world AI configuration files from open-source projects on GitHub. Since GitHub hosts not only "engineered" software projects, we needed to develop a strategy for selecting repositories. In the past, popularity-based selection based on stars or watchers had high precision in selecting true software projects [25]. However, nowadays GitHub is used to host more diverse content. In October 2025, none of the ten most popular repositories was used to host software projects [13].

Our starting point for selecting true software projects was the SEART GitHub search tool [6, 29]. Based on our experience from a previous research project, we selected non-fork repositories that have at least two contributors, have a license, and were created before 1st January 2024 with commit since 1st June 2024. We then excluded archived, disabled, or locked repositories, resulting in a first sample of 225,582 repositories. In the next step, we selected repositories with OSI-compliant open source license [19] and then manually filtered out licenses not intended for software (e.g. font licenses). We also excluded licenses with usage below the median value of 261 repositories. Further filters considered the programming languages (focus on 10 most popular languages: Python, TypeScript,

JavaScript, Go, Java, C++, Rust, PHP, C#, and C) and the commit count as we filtered out repositories with fewer than 271 commits (median value) or fewer than 7 watchers (median value). This resulted in a final sample of 49,311 repositories. Our long-term goal is to collect AI configuration files from all these repositories. For this work-in-progress paper, we selected 10,000 repositories based on a custom ranking approach that balances the popularity and maturity of the projects. Figure 1 outlines our data collection process. We cloned all repositories and scanned their default branch to find all types of configuration files that the GitHub Copilot coding agent supports [12]: Copilot instructions, CLAUDE.md, AGENTS.md, and GEMINI.md. We then manually checked all repositories to exclude non-English and non-software projects. We used the resulting data to answer **RQ1** (see Section 4.1). Since AGENTS.md is the only format to encode the project-specific context of AI coding agents that is not directly related to a specific tool, we decided to focus on these files for answering **RQ2** and **RQ3** (see Sections 4.2 and 4.3). Our data collection and analysis scripts and the collected and analyzed data are part of our supplementary material [4].

4 Results

4.1 Adoption (RQ1)

Only 466 (5%) of the repositories that we scanned had already adopted at least one of the formats that we considered. However, we focused only on four commercial tools, although other commercial or open-source alternatives such as *Cline* and *OpenCode* exist. We consider extending the analysis to cover more tools (and more repositories) an important direction for our future work. It will also be interesting to study trends over time, e.g., whether projects actually converge toward one file format or whether the tool-specific formats will persist.

The distribution of languages was roughly aligned with the languages' general representation in our sample, although Go was slightly overrepresented. We found AI configuration files in 135 repositories with TypeScript as main language, 58 Go, 58 Python, 56 C#, 36 Java, 34 JavaScript, 32 C++, 29 Rust, 19 PHP, and 9 C. The Copilot instruction files were on average the longest ($M = 310$ lines, $SD = 127$ lines), followed by CLAUDE.md files ($M = 287$, $SD = 112$). AGENTS.md files had the highest variation ($M = 142$, $SD = 231$). GEMINI.md files were the shortest ($M = 106$, $SD = 65$). We noticed that certain files types were more prevalent in certain programming languages. C#, for example, had a strong focus on *Copilot*, while *Claude Code* was very popular for TypeScript. We also looked at which files most commonly co-occured, finding that AGENTS.md and CLAUDE.md was the most common pair (in 25 repositories).

4.2 Information and Structure (RQ2)

To answer **RQ2**, we extracted all section headings from the 155 AGENTS.md file in our sample, converted them to lower case, removed special characters, and lemmatized the words to be able to group semantically equivalent variations (e.g., "tests" and "testing"). We excluded 15 AGENTS.md files that were created before January 1, 2025, i.e., before the AGENTS.md convention was introduced. For each lemmatized heading, we determined (1) how many different repositories used it, (2) in how many distinct files it appeared (some

repositories contained multiple files), and (3) how many total occurrences it had (some headings appeared multiple times per file). Five repositories contained AGENTS.md files without any heading structure; these were excluded from the analysis. We recorded the heading levels (from #, i.e., level 1, to #####, i.e., level 5) to understand the structural depth of the documents. Following related work on README files [10], which found that the first- and second-level headings are the most informative and consistent, we restricted our initial analysis to the headings on levels 1 and 2. We manually developed an initial coding guide based on the 44 lemmatized section headings that appeared in at least three different repositories and at least three times at heading levels 1 or 2; we also examined examples of section content for each heading. The resulting coding guide, shown in Table 1, groups semantically similar headings into broader conceptual categories. We then applied this guide to a larger set of 91 lemmatized section headings that were used in at least two repositories and at least twice at heading levels 1 or 2. This analysis provides an overview of the types of information most commonly provided in AGENTS.md files. Topics such as code conventions and best practices, contribution guidelines, and architecture or project structure appear frequently, while sections on troubleshooting or security occur less often.

We also noticed differences in writing style when analyzing the files. To examine these differences more closely, we analyzed all 50 sections labeled CONVENTIONS/BEST PRACTICES, the most common category in our dataset. We found that the writing style can be characterized along five stylistic dimensions: *descriptive*, *prescriptive*, *prohibitive*, *explanatory*, and *conditional*. Some sections were *descriptive*, documenting existing conventions without giving explicit instructions, e.g., "*This project uses the Linux Kernel Style Guideline.*" Such statements summarize current practices or configurations that the AI agent should be aware of, rather than prescribing behavior. Others were *prescriptive*, written as direct imperatives that instruct how to act, e.g., "*Use factories for all test data*" or "*Follow the existing code style and conventions.*" This style provides clear behavioral guidance, often formatted as concise bullet points to be interpreted as explicit rules. *Prohibitive* statements were also common, explicitly indicating what not to do, e.g., "*Never commit directly to the main branch.*" These prohibitions set boundaries and clarify the constraints that AI agents should respect. Some projects added short explanations after the rules, resulting in an *explanatory* style, e.g., "*Avoid hard-coded waits to prevent timing issues in CI environments.*" Here, the justification ("*to prevent timing issues*") provides context for why a convention exists, aiming to support understanding rather than mere compliance. Finally, we observed *conditional* formulations that specify what to do in certain situations, e.g., "*If you need to use reflection, use* ReflectionUtils *APIs.*" This style encodes situational logic, specifying conditional actions that depend on the context of the agent's task.

In summary, AGENTS.md files vary widely in both the information they contain and how it is presented, yet some recurring patterns are emerging. Projects often document architecture, contribution processes, and coding conventions—but without a consistent structure. Likewise, stylistic choices range from *descriptive* to *directive*, reflecting experimentation with how best to communicate expectations to AI agents. These observations suggest that conventions for documenting context are still evolving and point to promising

opportunities for future work on how information structure and style influence agent behavior.

4.3 Evolution (RQ3)

To answer **RQ3**, we first screened the commit histories of all 155 AGENTS.md files and found that 77 (50%) of them had not been changed after the initial commit, 36 (23%) only once, and 32 (21%) between two and seven times. For this study, we decided to focus on 10 files (6%) with at least 10 commits, which yielded a sample of 169 commits to annotate (37% of the 453 collected commits). We were primarily interested in understanding the types of changes that developers make to AI configuration files. The considered time spans vary between repositories, with some histories spanning a short period with many changes (e.g., neomjs/neo with 49 changes over 19 days) and others spanning longer periods with fewer changes (e.g., gofiber/fiber: 11 changes, 148 days). Although we did not analyze files with fewer than 10 commits in detail, we noticed that the history of these files varies significantly as well, ranging from 0 to 127 days, with 2 to 8 commits. A more detailed analysis of evolution patterns is an important direction for future work.

To understand what developers change in AI configuration files and to inductively identify general categories of changes, we manually reviewed the commits, including the source code diff, the commit messages, and any related issues or pull requests. Two authors developed an initial coding guide and then iteratively refined the emerging categories and descriptions For the analysis, we treated each commit as an isolated change. However, we observed that one change in the commit history can exemplify multiple types of changes rather than only one, but most commits (111, 66%) represented only one change category. Table 2 shows that some change categories such as '**Add** section(s)' refer to the overall structure of AGENTS.md files and indicate that the change to instructions was rather significant, while others such as '**Add** instruction(s)' refer to the content of sections and the extent and scope of change was more focused and local. Note that, at this stage, we do not differentiate between single and multiple changes of a category within a file. For example, '**Add** section(s)' can mean that one section was added or multiple sections were added. Furthermore, we do not discuss the intent of a change. For example, for the category '**Remove** section(s)' we do not analyze why this may have happened. An analysis of intent is subject of future work.

Table 2 shows that the most frequent change categories are '**Add** instruction(s)' and '**Modify** instruction(s)'. For all examined AGENTS.md files, these categories occurred as the first or second change in the history of changes. Looking at commit messages related to changes, we found a few interesting cases. For example, for AGENTS.md in rsyslog, one of the commit messages states "*AI support: Agent shall no longer call stylecheck.sh*". The related change category is '**Remove** instruction(s)', which indicates that an agent should not perform a certain operation. A commit in eclipse-rdf4j/rdf4j fixed a flaky test and at the same time updated the AGENTS.md file to handle potentially flaky tests during test execution. Another bug fix for the same file fixed a null pointer exception, again adjusting to test execution instructions to prevent related issues. This co-changes of source code, tests, and AI configuration files are a promising direction for future work. In

Table 1: Categories of information provided in AGENTS.md files (last column: number of level 1 or 2 headings).

Category	Description	#
Conventions/best practices	Outlines coding standards, naming/formatting conventions, and best practices for writing consistent and maintainable code.	50
Contribution guidelines	Provides instructions for contributing to the repository, such as branching, code reviews, or CI requirements.	48
Architecture and project structure	Describes how the project or repository is organized, including key directories, modules, components, and relationships between them.	47
Build commands	Lists commands for building, running, or deploying.	40
Project description	Summarizes what the project or agent does, its goals or purposes, and high-level functionality or capabilities.	32
Testing instructions	Explains how to execute test suites or individual tests, including tools, commands, and environments.	32
Document metadata	Contains file metadata or configuration (e.g., tags).	29
Testing strategy and guidelines	Describes the overall approach to testing (unit, integration, end-to-end), test organization, or principles guiding test coverage and design.	24
Tech stack and dependencies	Lists programming languages, libraries, frameworks, or other dependencies used in the project.	15
Getting started and setup	Covers installation prerequisites, environment setup, and initial steps required to run/use the project locally.	11
References and cheat-sheets	Provides a concise list of frequently used commands, API references, or quick tips for developers or users.	9
Troubleshooting	Offers guidance for diagnosing and resolving common errors, failures, or configuration problems encountered during development or deployment.	8
Patterns and examples	Shows reusable patterns, sample agent configs, or example use cases to guide understanding or extensions.	8
Security	Highlights security-related advice, configurations, or precautions (e.g., managing secrets or access controls).	6

Table 2: Categories of changes for AGENTS.md files with $\geq$ 10 commits (last column: category frequency across all files).

Category	Description	#
Add instruction(s)	Add one or more lines of instructions to existing sections.	78
Modify instruction(s)	Modify one or more lines of instructions within a section (not related to fixing typos or adding external references).	59
Add section(s)	Add new sections to the AGENTS.md file.	26
Remove instruction(s)	Remove one or more lines of instructions from existing sections.	23
Modify headings	Modify existing section heading title or level.	23
Modify text (minor)	Minor changes to content of AGENTS.md file, such as fixing typos.	19
Reformat instruction style	Changing visual appearance of content in AGENTS.md file (not related to structure).	10
Remove section(s)	Remove sections from AGENTS.md. file	2
Update reference(s)	Update references, e.g., URLs.	2

summary, the evolution of the AGENTS.md files in our sample varies, and we did not identify clear patterns in terms of when and how often changes occur. However, we did identify frequent change categories. Based on these categories, it appears that changes early on in the lifetime of AI configuration files are mostly about fine-tuning and adjusting instructions rather than major changes.

5 Conclusion

In addition to README files written for human contributors, projects increasingly include AI configuration files that provide instructions

and context for AI coding agents. In other words, software developers are now writing and maintaining documentation for machines. Our results show that conventions for this new form of documentation are still in flux. Projects differ widely in what they encode (e.g., architecture, conventions, workflows) and how they express it (e.g., prescriptive, prohibitive, explanatory). These stylistic variations mirror broader prompt writing practices and raise questions about whether and how tone and phrasing influence agent adherence or output quality. In this sense, open-source repositories serve as natural laboratories for studying how developers experiment with "talking" to agent-based AI tools.

AI configuration files should be treated as maintained software artifacts: versioned, reviewed, quality-assured, and tested. Future work needs to evaluate how their content, structure, and style affect agent behavior and task performance, and how automated feedback loops could update or refine these files based on observed results. Research could also investigate the co-evolution of source code and related AI configuration files, similar to the co-evolution of source code and comments [9].

Design questions include whether standard schemas could improve interoperability, whether repositories should maintain one centralized file or multiple module-level ones, and how to coordinate instructions for multiple agents. Beyond technical considerations, this new form of documentation has the potential to reshape communication, review, and collaboration patterns in software teams as instructions move from being written for humans to being negotiated between humans and AI. The systematic study of AI configuration files has great potential to provide actionable recommendations to practitioners.

References

[1] agents.md. 2025. Why AGENTS.md? https://agents.md/.
[2] Anthropic. 2025. Claude 3.7 Sonnet and Claude Code. https://www.anthropic.com/news/claude-3-7-sonnet.
[3] Anthropic. 2025. Claude Code Best Practices. https://www.anthropic.com/engineering/claude-code-best-practices.
[4] Anonymous Authors. 2025. *Context Engineering for AI Agents in Open-Source Software (Supplementary Material)*. doi:10.5281/zenodo.17428770
[5] Sebastian Baltes, Florian Angermeir, Chetan Arora, Marvin Muñoz Barón, Chun-yang Chen, Lukas Böhme, Fabio Calefato, Neil Ernst, Davide Falessi, Brian Fitzgerald, Davide Fucci, Marcos Kalinowski, Stefano Lambiase, Daniel Russo, Mircea Lungu, Lutz Prechelt, Paul Ralph, Rijnard van Tonder, Christoph Treude, and Stefan Wagner. 2025. Guidelines for Empirical Studies in Software Engineering involving Large Language Models. arXiv:2508.15503 [cs.SE] https://arxiv.org/abs/2508.15503
[6] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021*. IEEE, 560–564. doi:10.1109/MSR52588.2021.00074
[7] DAIR.AI Prompt Engineering Guide. 2025. Elements of a Prompt | Prompt Engineering Guide. https://www.promptingguide.ai/introduction/elements.
[8] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 31–53. doi:10.1109/ICSE-FOSE59343.2023.00008
[9] B. Fluri, M. Würsch, E. Giger, and H. Gall. 2009. Analyzing the co-evolution of comments and source code. *Software Quality Journal* 17 (2009), 367–394.
[10] Haoyu Gao, Christoph Treude, and Mansooreh Zahedi. 2025. Adapting Installation Instructions in Rapidly Evolving Software Ecosystems. *IEEE Trans. Software Eng.* 51, 4 (2025), 1334–1357. doi:10.1109/TSE.2025.3552614
[11] GitHub. 2025. Copilot code review: Customization for all - GitHub Changelog. https://github.blog/changelog/2025-06-13-copilot-code-review-customization-for-all/.
[12] GitHub. 2025. Copilot coding agent now supports AGENTS.md custom instructions. https://github.blog/changelog/2025-08-28-copilot-coding-agent-now-

[13] GitHub. 2025. Search for most popular repositories. https://github.com/search?q=stars%3A%3E10000&type=Repositories&s=stars&o=desc.
[14] GitHub. 2025. Search for AGENTS.md files. https://github.com/search?q=path%3A**%2FAGENTS.md&type=code.
[15] Significant Gravitas. 2023. AutoGPT. https://agpt.co/.
[16] Dexter Horthy. Getting AI to Work in Complex Codebases. https://github.com/humanlayer/advanced-context-engineering-for-coding-agents/blob/main/ace-fca.md.
[17] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8 (2024), 220:1–220:79. doi:10.1145/3695988
[18] Laurie Hughes, Yogesh K. Dwivedi, F. Tegwen Malik, Mazen Shawosh, Mousa Ahmed Albashrawi, Il Jeon, Vincent Dutot, Mandanna Appanderanda, Tom Crick, Rahul De', Mark Fenwick, Madugoda Gunaratnege Senali, Paulius Jurcys, Arpan Kumar Kar, Nir Kshetri, Keyao Li, Sashah Mutasa, Spyridon Samothrakis, Michael R. Wade, and Paul Walton. 2025. AI Agents and Agentic Systems: A Multi-Expert Analysis. *J. Comput. Inf. Syst.* 65, 4 (2025), 489–517. doi:10.1080/08874417.2025.2483832
[19] Open Source Initiative. 2025. OSI Approved Licenses. https://opensource.org/licenses.
[20] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2025. A Survey on Large Language Models for Code Generation. *ACM Trans. Softw. Eng. Methodol.* (July 2025). doi:10.1145/3747588 Just Accepted.
[21] Will Knight. 2024. Forget Chatbots. AI Agents Are the Future. https://www.wired.com/story/fast-forward-forget-chatbots-ai-agents-are-the-future/.
[22] Hao Li, Hicham Masri, Filipe Roseiro Côgo, Abdul Ali Bangash, Bram Adams, and Ahmed E. Hassan. 2025. Understanding Prompt Management in GitHub Repositories: A Call for Best Practices. *CoRR* abs/2509.12421 (2025). arXiv:2509.12421 doi:10.48550/ARXIV.2509.12421
[23] Ziyou Li, Agnia Sergeyuk, and Maliheh Izadi. 2025. Prompt-with-Me: in-IDE Structured Prompt Management for LLM-Driven Software Engineering. *CoRR* abs/2509.17096 (2025). arXiv:2509.17096 doi:10.48550/ARXIV.2509.17096
[24] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A Survey of Context Engineering for Large Language Models. *CoRR* abs/2507.13334 (2025). arXiv:2507.13334 doi:10.48550/ARXIV.2507.13334
[25] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empir. Softw. Eng.* 22, 6 (2017), 3219–3253. doi:10.1007/S10664-017-9512-6
[26] OpenAI. 2025. Introducing Codex. https://openai.com/index/introducing-codex/.
[27] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *CoRR* abs/2402.07927 (2024). arXiv:2402.07927 doi:10.48550/ARXIV.2402.07927
[28] Philipp Schmid. 2025. The New Skill in AI is Not Prompting, It's Context Engineering. https://www.philschmid.de/context-engineering.
[29] SEART. 2025. GitHub Search. https://seart-ghs.si.usi.ch/.
[30] Samdyuti Suri, Sankar Narayan Das, Kapil Singi, Kuntal Dey, Vibhu Saujanya Sharma, and Vikrant Kaulgud. 2023. Software Engineering Using Autonomous Agents: Are We There Yet?. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 1855–1857. doi:10.1109/ASE56229.2023.00174
[31] Mahan Tafreshipour, Aaron Imani, Eric Huang, Eduardo Santana de Almeida, Thomas Zimmermann, and Iftekhar Ahmed. 2025. Prompting in the Wild: An Empirical Study of Prompt Evolution in Software Repositories. In *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 686–698. doi:10.1109/MSR66628.2025.00106
[32] Rosalia Tufano, Ozren Dabic, Antonio Mastropaolo, Matteo Ciniselli, and Gabriele Bavota. 2024. Code Review Automation: Strengths and Weaknesses of the State of the Art. *IEEE Trans. Software Eng.* 50, 2 (2024), 338–353. doi:10.1109/TSE.2023.3348172
[33] Hugo Villamizar, Jannik Fischbach, Alexander Korn, Andreas Vogelsang, and Daniel Méndez. 2025. Prompts as Software Engineering Artifacts: A Research Agenda and Preliminary Findings. *CoRR* abs/2509.17548 (2025). arXiv:2509.17548 doi:10.48550/ARXIV.2509.17548
[34] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Trans. Software Eng.* 50, 4 (2024), 911–936. doi:10.1109/TSE.2024.3368208
[35] Scott Wu. 2024. Introducing Devin, the first AI software engineer. https://cognition.ai/blog/introducing-devin.
[36] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin,

Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui. 2025. The rise and potential of large language model based agents: a survey. *Sci. China Inf. Sci.* 68, 2 (2025). doi:10.1007/S11432-024-4222-0

[37] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/

hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html

[38] 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, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, Maria Christakis and Michael Pradel (Eds.). ACM, 1592–1604. doi:10.1145/3650212.3680384

[39] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. 2025. Towards an understanding of large language models in software engineering tasks. *Empir. Softw. Eng.* 30, 2 (2025), 50. doi:10.1007/S10664-024-10602-0