Context Engineering for AI Agents in Open-Source Software

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

Matthias Galster
University of Bamberg
Germany
mgalster@ieee.org

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

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

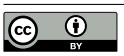
Abstract

GenAI-based coding assistants have disrupted software development. The next generation of these tools is agent-based, operating with more autonomy and potentially without human oversight. Like human developers, AI agents require contextual information to develop solutions that are in line with the standards, policies, and workflows of the software projects they operate in. Vendors of popular agentic tools (e.g., Claude Code) recommend maintaining version-controlled Markdown files that describe aspects such as the project structure, code style, or building and testing. The content of these files is then automatically added to each prompt. Recently, `AGENTS.md` has emerged as a potential standard that consolidates existing tool-specific formats. However, little is known about whether and how developers adopt this format. Therefore, in this paper, we present the results of a preliminary study investigating the adoption of AI context files in 466 open-source software projects. We analyze the information that 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 content structure yet and that there is a lot of variation in terms of how context is provided (descriptive, prescriptive, prohibitive, explanatory, conditional). Our commit-level analysis provides first insights into the evolution of the provided context. AI context files provide a unique opportunity to study real-world context engineering. In particular, we see great potential in studying which structural or presentational modifications can positively affect the quality of the generated content.

Keywords

Software Engineering, Generative AI, AI Agents, Open Source

ACM Reference Format:
Seyedmoein Mohsenimofidi, Matthias Galster, Christoph Treude, and Sebastian Baltes. 2026. Context Engineering for AI Agents in Open-Source Software. In *23rd International Conference on Mining Software Repositories (MSR '26), April 13–14, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3793302.3793350



MSR '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2474-9/2026/04
https://doi.org/10.1145/3793302.3793350

1 Introduction

The launches of GitHub Copilot in 2021 and ChatGPT in 2022 have started to transform how software is developed. Today, generative AI (GenAI) tools built around large language models (LLMs) support software engineers throughout the software development lifecycle (SDLC)—although most published work focuses on code and test generation [7, 15, 17, 34, 36, 41]. The *Devin AI* demo published in March 2024 fueled the first hype around agent-based software development [18], 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*" [1], which represents a further step toward more autonomous AI-assisted software development, enabling 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. This brings GenAI assistants closer to the inherent meaning of an *agent* that operates autonomously, adapts to change, and creates and pursues goals (from the Latin 'agere', 'to do' in English) [27].

Context engineering is the deliberate process of designing, structuring, and providing task-relevant information to LLMs [22, 29]. While *prompt engineering* focuses on how a task is described to the model (e.g., instructions and output indicators) [6, 28], context engineering focuses on what task-relevant information the model has access to, including relevant guidelines, configuration files, documentation, and exemplary code snippets [22, 29]. An advantage of agent-based tools compared to conversational tools is that they allow persistent, structured, and task-specific context to be provided in a more fine-grained and targeted manner [14]. One way to "engineer" context is to add machine-readable AI context files to source code repositories. The AI agents then automatically add the content of these files to their prompts. While traditional `README` files are written for humans, AI context files are explicitly designed for AI agents, providing a central machine-readable source of contextual information. Their content can include everything from the required terminal commands to build and test the project over documentation links, common workflows, coding conventions, to instructions for creating pull requests.

`AGENTS.md` was introduced as an open tool-agnostic convention for such AI context files [19], and it was recently announced as a project in the *Agentic AI Foundation*. 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 [2]. GitHub introduced a similar AI context file for *Copilot* named `copilot-instructions.md` [13].

Since prompts are only rarely preserved after content has been generated [33], AI context 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 instructions and contextual descriptions evolve. To our knowledge, we present the first holistic empirical study that analyzes context files used to guide different AI agents in open-source software (OSS) projects. We addressed the following research questions:

- **RQ1** *How widely have OSS projects adopted AI context files?*
- **RQ2** *What information do open-source developers provide in AI context files and how do they present it?*
- **RQ3** *How do AI context files evolve over time?*

An initial search in October 2025 suggested that tens of thousands of GitHub repositories already contained AI context files [12]. However, there was no systematic analysis focusing on "engineered" software projects. This paper provides a first step toward filling this gap by mining GitHub repositories to study how AI context files are adopted, structured, and maintained, with the overarching goal of understanding how software teams engage in 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 context files as novel software artifacts. We will extend our study in the future to answer the research questions more holistically.

2 Related Work

Agentic GenAI tools promise to introduce autonomous decision-making and proactive problem-solving along the SDLC [16, 32]. Advances in LLMs, reinforcement learning, and multi-agent frameworks enabled the implementation of software agents that go beyond simple prompt-response interactions [38]. One of the first agent-based software development tools was *Devin AI* [37], which allowed agents to search the web, edit files, and execute commands to complete tasks iteratively and independently. In the academic community, *SWE-agent* [39] allowed LLM-based agents to communicate with the repository environment by reading, modifying, and executing bash commands [39]. Another example is *AutoCodeRover* [40], which enabled agents to access code search APIs to help them find methods within specific classes for bug location identification [40]. Suri et al. showcased the potential of autonomous agents, particularly *Auto-GPT*, in software engineering tasks and demonstrates the importance of context-specific prompts in complex frameworks [32]. However, they also revealed that *Auto-GPT* [31], 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 [20, 21, 35]. This lack of prompt management limits reproducibility [3] and underscores the importance of making prompt and context information explicit and manageable by using versioned AI context files. Recently, researchers have started investigating AI context files as novel software artifacts [4]. However, a holistic analysis across tools and formats has, to the best of our knowledge, not been published yet.

3 Data Collection

To answer our research questions, we collected AI context files from OSS projects on GitHub. Since GitHub hosts not only "engineered' software projects, we needed to develop a strategy for selecting repositories [24]. Many popular repositories on GitHub are not software projects [11], which complicates sampling.

Our starting point for selecting true software projects was the SEART GitHub search tool [5, 30]. We selected non-fork repositories that have at least two contributors, have a license, and were created before 1st January 2024 with commits since 1st June 2024. We then excluded archived, disabled, or locked repositories, resulting in a first sample of 228,890 repositories. In the next step, we selected repositories with an OSI-compliant open-source license [25] and then manually filtered out licenses not intended for software and licenses with low adoption, i.e., used in fewer than 261 repositories (median). We focused on the ten most popular languages (Python, TypeScript, JavaScript, Go, Java, C++, Rust, PHP, C#, and C) and excluded repositories with fewer than 271 commits (median) or fewer than 7 watchers (median). This resulted in a final sample of 48,795 repositories. The purpose of this filtering process was to select repositories that represent actively maintained software projects with a sufficiently long development history, written in one of the major programming languages. For this paper, our goal was to start with mature popular repositories. Thus, we selected 10,000 repositories based on a ranking approach that balances popularity (#stars, #watchers, #contributors) and maturity (#commits to default branch, project age, LOC).

Figure 1 outlines our data collection process for these 10,000 repositories. We cloned them and scanned their default branch to find all types of context files that GitHub Copilot supports: Copilot instructions, `CLAUDE.md`, `AGENTS.md`, and `GEMINI.md` [10]. 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 that serves as an open tool-agnostic convention, we decided to focus on it to answer **RQ2** and **RQ3** (see Sections 4.2 and 4.3). Our data collection and analysis scripts and the analyzed data are available online [23].

4 Results

4.1 Adoption (RQ1)

Only 466 (5%) of the repositories that we scanned had already adopted at least one of the formats we considered, reflecting that we are still in an early stage of adoption. One limitation is our focus on four selected tools. 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 converge toward one file format or whether tool-specific formats persist.

The distribution of languages was roughly aligned with the languages' general representation in our sample, although Go was slightly overrepresented. We found AI context files in 135 repositories with TypeScript as main language, 58 Go, 58 Python, 56 C#, 36

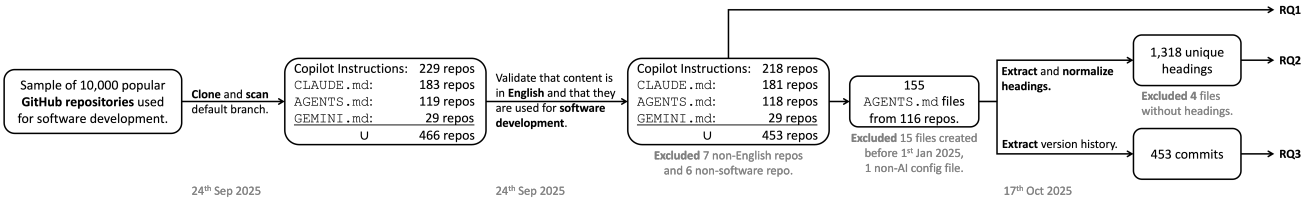


Figure 1: Data collection process.

Java, 34 JavaScript, 32 C++, 29 Rust, 19 PHP, and 9 C. Certain file 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 investigated which files most commonly co-occurred, finding that (`AGENTS.md`, `CLAUDE.md`) was the most common pair (25 repositories).

4.2 Information and Structure (RQ2)

Before we dive into our answer to **RQ2**, we briefly characterize the content of AI context files. Copilot instruction files were on average the longest ($M = 310$ lines, $SD = 127$ lines), followed by `CLAUDE.md` files ($M = 287$, $SD = 112$); `GEMINI.md` files were the shortest ($M = 106$, $SD = 65$). Interestingly, `AGENTS.md` files had the highest variation in file length ($M = 142$, $SD = 231$). This variation may reflect the amount of information that developers provide. Exploring whether the context length is proportional to the project size or other factors is part of our future work.

To answer **RQ2**, we extracted all section headings from the 155 `AGENTS.md` files 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 1st January 2025, i.e., before the `AGENTS.md` convention was introduced. For each lemmatized heading, we determined (1) in how many distinct repositories it occurred, (2) in how many distinct files it occurred (#files > #repositories), and (3) how many total occurrences it had (multiple equivalent headings per file). We excluded five repositories that contained `AGENTS.md` files without any heading structure. For the remaining files, we recorded the heading levels (from `#`, i.e., level 1, to `#####`, i.e., level 5) to understand their structural depth. Following related work on `README` files [9], which found that the first- and second-level headings are the most informative and consistent, we restricted our initial analysis to these levels.

We manually developed an initial coding guide based on the 44 lemmatized section headings that appeared in ≥ 3 different repositories and ≥ 3 times at heading levels 1 or 2. We then examined examples of section content for each heading. The resulting coding guide, shown in Table 1, groups semantically similar headings into categories. We then applied this guide to a larger set of 91 lemmatized section headings that were used in ≥ 2 repositories and ≥ 2 times at heading levels 1 or 2. This analysis provides an overview of the information most commonly provided in `AGENTS.md` files. Topics such as code conventions and best practices, contribution guidelines, and architecture or project structure appeared frequently, in contrast to sections on troubleshooting or security.

We also noticed differences in writing style when analyzing the files. To examine these differences more closely, we analyzed all 50 sections labeled Conventions, 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., "*Follow the existing code style and conventions.*" This style provides explicit behavioral rules and was often formatted as concise bullet points. *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. 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 both in the information they contain and how they are presented, yet some recurring patterns are emerging. Projects often document architecture, contribution processes, and coding conventions, but without a consistent structure. 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 analyzed the commit histories of all 155 `AGENTS.md` files and found that 77 (50%) of them had not been changed, 36 (23%) only once, and 32 (21%) between two and seven times. For this study, we were primarily interested in understanding the types of changes developers make in AI context files. We decided to focus on the 10 files (6%) with at least 10 commits, which yielded a sample of 169 commits to annotate (37% of all collected commits). The resulting modification patterns varied per file, with some histories spanning a short period with many changes (e.g.,

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

Category	Description	#
Conventions	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/ 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
Goals/purposes	Summarizes what the project or agent does, its goals or purposes, and high-level functionality or capabilities.	32
Test execution	Explains how to execute test suites or individual tests, including tools, commands, and environments.	32
Metadata	Contains file metadata or configuration (e.g., tags).	29
Test strategy	Describes the overall approach to testing (unit, integration, end-to-end), test organization, or principles guiding test coverage and design.	24
Tech stack	Lists programming languages, libraries, frameworks, or other dependencies used in the project.	15
Setup	Covers installation prerequisites, environment setup, and initial steps required to run/use the project locally.	11
References	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/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

`neomjs/neo`: 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 varied 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 context files and to inductively identify general change categories, 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. We considered each commit as an isolated change. Although most commits (111 commits, i.e., 66%) represented only one change category, we observed commits that represented multiple. Table 2 shows that some categories refer to the overall structure of `AGENTS.md` files while others refer to the content of specific sections, indicating a narrower scope. Note that we currently do not quantify changes per category, meaning that, e.g., '**Add** section(s)' can represent a change that added one or multiple sections. Furthermore, we did not label the intent of a change, e.g., why a certain section was added or removed. An analysis of the intent of changes is the subject of our 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,

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

Category	Description	#
Add instruction(s)	Add instruction line(s) to existing sections.	78
Modify instruction(s)	Modify instruction line(s) within a section (ignoring typo fixes and references additions).	59
Add section(s)	Add new section(s) to the `AGENTS.md` file.	26
Remove instruction(s)	Remove line(s) with instructions from existing sections.	23
Modify heading(s)	Modify existing section heading title or level.	23
Modify text	Minor changes to content of `AGENTS.md` file, such as fixing typos.	19
Reformat 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

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)', because the change deleted an instruction. A commit in `eclipse-rdf4j/rdf4j` fixed a flaky test and updated the `AGENTS.md` file to handle flaky tests during test execution. Analyzing co-changes of AI context files and other artifacts is a promising direction for future work.

In 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 common change categories. Based on these categories, it appears that changes are mostly made to fine-tune and adjust instructions.

5 Conclusion

In addition to `README` files for humans, OSS projects increasingly include AI context files 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 software artifact are still in flux. Projects differ widely in what they encode (e.g., conventions, architecture) and how they express it (e.g., prescriptive vs. prohibitive). These stylistic variations mirror different prompt writing practices. Thus, OSS repositories serve as natural laboratories for studying how developers experiment with "talking" to agent-based AI tools.

AI context files are maintained software artifacts. They are 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 should also investigate the co-evolution of source code and related AI context files, similar to the co-evolution of source code and comments [8]. Open questions include whether standard schemas could improve interoperability, whether repositories should maintain one or multiple AI context files, 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 context files has great potential to provide actionable recommendations to practitioners.

References

[1] Anthropic. 2025. Claude 3.7 Sonnet and Claude Code. https://www.anthropic.com/news/claude-3-7-sonnet.

[2] Anthropic. 2025. Claude Code Best Practices. https://www.anthropic.com/engineering/claude-code-best-practices.

[3] Sebastian Baltes, Florian Angermeir, Chetan Arora, Marvin Muñoz Barón, Chunyang 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

[4] Worawalan Chatlatanagulchai, Kundjanasith Thonglek, Brittany Reid, Yutaro Kashiwa, Pattara Leelaprute, Arnon Rungsawang, Bundit Manaskasemsak, and Hajimu Iida. 2025. On the Use of Agentic Coding Manifests: An Empirical Study of Claude Code. In *Proceedings of the 26th International Conference on Product-Focused Software Process Improvement (PROFES 2025)*.

[5] 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

[6] DAIR.AI Prompt Engineering Guide. 2025. Elements of a Prompt | Prompt Engineering Guide. https://www.promptingguide.ai/introduction/elements.

[7] 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

[8] 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.

[9] 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

[10] GitHub. 2025. Copilot coding agent now supports AGENTS.md custom instructions. https://github.blog/changelog/2025-08-28-copilot-coding-agent-now-supports-agents-md-custom-instructions/.

[11] GitHub. 2025. Search for most popular repositories. https://github.com/search?q=stars%3A%3E10000&type=Repositories&s=stars&o=desc.

[12] GitHub. 2025. Search for `AGENTS.md` files. https://github.com/search?q=path%3A**%2FAGENTS.md&type=code.

[13] GitHub Changelog. 2025. Copilot Code Review: Customization for All. https://github.blog/changelog/2025-06-13-copilot-code-review-customization-for-all/.

[14] Dexter Horthy. 2025. Getting AI to Work in Complex Codebases. https://github.com/humanlayer/advanced-context-engineering-for-coding-agents/blob/main/ace-fca.md.

[15] 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

[16] 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

[17] 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.

[18] Will Knight. 2024. Forget Chatbots. AI Agents Are the Future. https://www.wired.com/story/fast-forward-forget-chatbots-ai-agents-are-the-future/.

[19] LF Projects. 2025. Why AGENTS.md? https://agents.md/.

[20] 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

[21] 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

[22] 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

[23] Seyedmoein Mohsenimofidi, Matthias Galster, Christoph Treude, and Sebastian Baltes. 2025. *Context Engineering for AI Agents in Open-Source Software (Supplementary Material)*. doi:10.5281/zenodo.17428770

[24] 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

[25] Open Source Initiative. 2025. Approved Licenses. https://opensource.org/licenses.

[26] OpenAI. 2025. Introducing Codex. https://openai.com/index/introducing-codex/.

[27] Stuart Russell and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach*.

[28] 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

[29] Philipp Schmid. 2025. The New Skill in AI is Not Prompting, It's Context Engineering. https://www.philschmid.de/context-engineering.

[30] SEART. 2025. GitHub Search. https://seart-ghs.si.usi.ch/.

[31] Significant Gravitas. 2023. AutoGPT. https://agpt.co/.

[32] 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

[33] 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

[34] 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

[35] 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

[36] 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

[37] Scott Wu. 2024. Introducing Devin, the first AI software engineer. https://cognition.ai/blog/introducing-devin.

[38] 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

[39] 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

[40] 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

[41] 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