

Developer Skills for Agentic AI in Software Architecture

Muhammad Auwal Abubakar¹

¹University of Bamberg, Germany

Abstract

Agentic AI tools such as Claude Code, Cursor, and GitHub Copilot can pursue developer-specified goals across multiple steps by inspecting files, planning changes, editing artifacts, and revising outputs without requiring the developer to direct every step. The developer's role can therefore shift from directly producing artifacts to framing goals, providing context, monitoring the agent's actions, and evaluating its output.

Software architecture provides a relevant context for examining this shift because architectural work involves system-level structures and design decisions, quality attributes, constraints, and trade-offs that require developer judgment. When architecture-related work is delegated to an agent, the resulting output may compile and pass tests while still violating an architectural constraint, such as a layering rule, or conflicting with an existing design decision. Detecting such problems requires developers to assess whether the output is consistent with the system's architectural context, decisions, and constraints.

Agentic AI workflows for software architecture can involve developer intervention at multiple points, yet it remains unclear which skills developers need when intervening and how these skills correspond to existing software architecture competencies. This research investigates that gap through a systematic mapping study, semi-structured interviews, cross-study synthesis, and conceptual framework development. The Software Architecture knowledge area of SWEBOK v4 provides the analytical basis for comparing the characterized skills with existing software architecture competencies.

Keywords

Agentic AI, Software architecture, Developer skills, Developer interventions, Human-AI collaboration

1. Introduction

Software developers work alongside agentic AI tools that can execute multi-step software development tasks with some degree of autonomy, without requiring the developer to direct every intermediate step. For example, tools such as Claude Code, Cursor, and GitHub Copilot can inspect a codebase, plan a sequence of changes, edit multiple files, run tests, and revise their outputs without waiting for the developer to direct each step [1, 2]. This is different from asking a chatbot to write a function in a prompt. The developer is no longer the sole author of every software artifact produced or modified during the task; instead, developers frame tasks, provide context, monitor the agent's actions, and decide what to do with an agent's output [1].

Research on agentic AI for software engineering has focused on coding tasks, including code generation, testing, debugging, and program repair [3]. Many coding-related tasks produce concrete implementation-level artifacts that can be evaluated using tests, regression checks, code review, and related evidence. In contrast, software architecture concerns system-level structures and design decisions. It requires consideration of multiple components, their interactions, design constraints, and system-wide quality attributes and trade-offs between quality attributes [4]. For example, architectural decisions may also involve selecting tactics and patterns to achieve specific quality attributes [4]. Moreover, software architecture and source code represent design decisions at different levels of abstraction: architecture captures higher-level structures and decisions, whereas source code represents their lower-level implementation [5]. Architectural knowledge may therefore be expressed through different views, decisions, patterns, and other architectural representations, rather than being directly observable from source code or files alone [5, 4].

IDOESE'26: International Doctoral Symposium on Empirical Software Engineering, October 07, 2026, München, Germany

✉ muhammad.abubakar@uni-bamberg.de (M. A. Abubakar)

id 0009-0006-1028-0650 (M. A. Abubakar)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Software architecture activities include choosing among design alternatives, defining component boundaries, specifying dependency constraints, and documenting decisions and judgments that entail trade-offs among quality attributes such as maintainability, performance, and security [6]. Architectural decisions inherently involve trade-offs because quality attributes, architectural concerns, and design constraints may conflict and cannot always be satisfied simultaneously [4]. Consequently, the quality of an architectural decision cannot always be reduced to a simple pass-or-fail assessment. For example, an agent asked to implement a feature may produce code that compiles and passes all tests while still introducing a layering violation, ignoring a documented design decision, or applying an architectural pattern that conflicts with the rest of the system. The developer must therefore recognize such violations and assess whether the resulting architectural trade-offs are acceptable. Architecture knowledge is also context-dependent and spans multiple aspects of a system, including its structure, component interactions, quality attributes, architectural tactics, and architecture decisions [4]. The suitability of architectural solutions, therefore, depends on the system and design context in which they are applied [4]. Consistent with the complexity of such problems, architecture-related questions have been reported as among the more difficult on Stack Overflow [7, 5].

Agentic AI tools can also be applied to software architecture activities, for instance, to support architectural design, decision-making, documentation, or evaluation. Given the judgment and trade-offs involved in architectural decisions, the use of agents in these activities may require developer oversight. Dhanorkar et al. found that oversight of agentic AI tools is active and demanding rather than passive and that developers can miss problems even when actively monitoring the agent's actions [8]. They also observed a shift in the competencies required when working with software agents, with greater emphasis on architectural design, critique, and review [8]. Developers can interact with agentic AI at different points in a workflow, not just by providing an initial instruction. They may define the task, provide context or constraints, monitor or intervene during execution, and review, accept, reject, modify, or request revision of outputs. The extent of human involvement may also differ across software development activities. Planötscher et al. [9], for example, examine human-AI collaboration across several activities, including software architecture design, and distinguish collaboration modes based on how work is divided between humans and AI. Evidence from industrial use of generative AI (GenAI) also suggests differences across software development activities. Pereira et al. [10] found that, within the organization examined in their case study, GenAI use was concentrated on code-intensive tasks. At the same time, collaborative activities saw minimal or no use, and high-level architectural design was among the creative tasks with limited reported influence from GenAI.

Galster et al. show that developers configure agentic AI coding tools through versioned repository-level artifacts, including Markdown-based context files such as `CLAUDE.md` and `AGENTS.md`, which provide build commands, coding conventions, and project-specific rules [2]. Mohsenimofidi et al. found architecture and project structure to be among the frequently documented categories in `AGENTS.md` files [11], while Santos et al. found software architecture to be a prominent concern in `CLAUDE.md` files [12]. A study of Agent Plan likewise identified architectural and design information, as well as design constraints and decisions, whereas requirements clarification and design exploration were not observed in the corpus studied [13]. Therefore, these studies provide evidence that architectural information is already present in repository-level artifacts that guide agentic coding tools. However, they focus on particular types of repository artifacts rather than systematically examining the broader use of agentic AI across software architecture activities.

What remains unclear is how developers interact with agentic AI tools when performing software architecture activities, what skills they need in these interactions, and how those skills correspond to existing software architecture competencies. The proposed research investigates this gap using the Software Architecture knowledge area of SWEBOK v4 as an analytical framework [6].

1.1. Research Objectives and Research Questions

This research pursues three broad objectives, which are addressed through the research questions described below.

- Establish what is currently known about the use of agentic AI for software architecture.
- Investigate how developers interact with agentic AI when performing software architecture activities and characterize the skills they apply in these interactions.
- Organize the identified skills in a conceptual framework grounded in empirical findings and mapped against existing software architecture competency frameworks.

1.1.1. RQ1: What is the state of research on the use of agentic AI in software architecture?

This question establishes what is currently known about the use of agentic AI in software architecture.

Sub-questions.

- *Which software architecture activities have been supported by agentic AI tools?*
- *In which application domains and system contexts is agentic AI for software architecture studied, and which tools, models, and configurations are used?*
- *Which modes of GenAI use (agentic, conversational, or hybrid) are identified in studies of agentic AI for software architecture activities?*
- *What inputs do developers provide to agentic AI tools for software architecture activities, and what outputs do these tools produce?*
- *How do developers interact before, during, and after AI execution in agentic AI workflows for software architecture activities?*
- *Which skills or competencies are described for developers interacting with agentic AI tools during software architecture activities?*
- *How do existing studies evaluate agentic AI tools for software architecture activities?*

Method: A systematic mapping study following Petersen et al. [14]. Electronic database searching will be the primary search method, complemented by manual searches. The database search will cover ACM Digital Library, IEEE Xplore, Scopus, Web of Science, ScienceDirect, and SpringerLink. Studies published from 2022 up to the date of the final search will be considered, with arXiv searched as a supplementary source for relevant scholarly preprints. A manual search will cover the main research or technical tracks of ICSE and ESEC/FSE, followed by ICSE, FSE, ASE, and ESEM, as well as TSE, TOSEM, EMSE, and JSS. After study selection, backward and forward snowballing will be conducted on all included primary studies. The identified studies will be classified and mapped according to the sub-questions above.

1.1.2. RQ2: How do developers intervene in agentic AI workflows for software architecture activities?

This question investigates developer interventions in agentic AI workflows for software architecture activities, focusing on the types of intervention, where they occur in the workflows, and the factors that prompt them.

Sub-questions.

- *What forms of intervention do developers describe in agentic AI workflows for software architecture activities?*
- *At what points in the agentic AI workflows do developers intervene?*
- *What factors prompt developers to intervene?*

Method: A qualitative study using semi-structured interviews [15] as the primary data collection method. Think-aloud sessions may complement the interviews for participants who can perform a software architecture activity with an agentic AI tool during the study. The interviews will first elicit

participants' descriptions of agentic AI workflows they have used for software architecture activities before examining the interventions within those workflows. Participants will be recruited through professional and institutional networks and through open calls in developer communities whose members use agentic AI coding tools. Recruitment will target developers and software architects who have used agentic AI tools for software architecture activities and will seek variation in architectural experience, application domains, and development settings. Interviews will last 45–60 minutes. We aim to conduct 15–20 interviews, with recruitment continuing up to 30 interviews if saturation is not reached [16]. Interview data will be analyzed using qualitative content analysis following Mayring [17]. To strengthen the credibility of the analysis, the interview guide will be piloted before data collection, coding decisions will be documented, and a second researcher will review a subset of the coded material.

1.1.3. RQ3: What skills do developers need when intervening in agentic AI workflows for software architecture activities?

This question characterizes the skills developers need when intervening in agentic AI workflows for software architecture activities. A skill is understood as an ability or proficiency acquired through training and practice [18].

Sub-questions.

- *Which skills do developers need at the points where they intervene in agentic AI workflows?*
- *Which of these skills correspond to existing software architecture competencies, and do any appear specific to the use of agentic AI tools?*
- *How do developers with different levels of software architecture expertise differ in how they apply these skills when intervening in agentic AI workflows for software architecture activities?*

Method: A cross-study synthesis of the findings from RQ1 and RQ2, following the synthesis approach described by Cruzes and Dybå [19]. The synthesis will proceed in three steps. First, the RQ2 interview data will be analyzed to identify the knowledge developers use and the judgments they make at the points where they intervene in agentic AI workflows. These findings will be used to characterize the skills associated with those interventions. Second, the resulting skill characterization will be mapped against the Software Architecture knowledge area of SWEBOK v4 [6] to examine which skills correspond to existing software architecture competencies and whether any appear specific to the use of agentic AI tools. Third, the RQ1 mapping findings will be used to compare this characterization with the skills and competencies described in prior studies of agentic AI for software architecture activities. The synthesis will also examine how developers with different levels of software architecture expertise differ in how they apply these skills.

1.1.4. RQ4: How can developer skills in agentic AI workflows for software architecture activities be organized into a conceptual framework?

This question examines how the skills characterized in RQ3 can be organized into a conceptual framework based on the points of intervention identified in RQ2.

Method: The conceptual framework will be developed from the findings of RQ2 and RQ3 in three steps. First, the skill characterization from RQ3 will define the core elements of the framework. Second, these skills will be organized according to the points of intervention identified in RQ2, while the mapping to existing software architecture competencies established in RQ3 will also be represented in the framework. This allows the structure of the framework to emerge from the empirical findings rather than from a predefined set of intervention points. Third, the resulting framework will be evaluated through expert review [20]. Experts will assess its clarity, coherence, coverage, and practical relevance, and their feedback will be used to refine the framework.

2. Related Work

2.1. Software Development Expertise and Skills

A skill is an ability or proficiency acquired through training and practice [18]. Baltes and Diehl characterize software development expertise as task- and context-dependent and distinguish expertise from years of experience [18]. Their theory emphasizes the quality of experience, deliberate practice, monitoring, feedback, and self-reflection in the development of expertise. Although their theory focuses primarily on programming, it provides a conceptual basis for examining the skills developers need in other software engineering activities. This research examines such skills in the context of software architecture activities performed with agentic AI tools. Developers may need to define tasks for an agent, provide architectural context and constraints, intervene during execution, and evaluate the resulting output. Which skills are involved in these interventions, and how they correspond to existing software architecture competencies, remain empirical questions. The Software Architecture knowledge area of SWEBOK v4 describes established knowledge concerning architectural concepts, representation, processes, and evaluation [6], but does not specifically address the skills developers need when working with agentic AI tools.

2.2. Skills for AI-Assisted Software Development

The skills developers need when working with GenAI tools have begun to receive empirical attention. Jeuring et al. investigate whether computational-thinking skills predict the ability to develop software using LLM-based tools and find that performance on a computational-thinking assessment predicts participants' performance when developing software using ChatGPT [21]. Vaithilingam et al. found that participants experienced difficulties understanding, editing, and debugging code generated by GitHub Copilot [22]. Barke et al. identified two interaction modes: acceleration, in which programmers know how they intend to proceed and use Copilot to work faster, and exploration, in which programmers are unsure how to proceed and use Copilot to explore possible solutions. They also found differences in validation strategies between the two modes [23]. Together, these studies show that working with code-generating tools involves skills related to understanding, evaluating, and modifying generated code. However, they focus on programming and code-generation contexts rather than software architecture activities. They also do not examine developer interventions in agentic AI workflows or how the skills involved in such interventions correspond to existing software architecture competencies.

2.3. Software Architecture Competencies

Prior work describes what software architects need to know and the activities they are expected to perform. Kruchten characterizes the software architect's role as requiring a balance between internal architectural work, such as decision making, evaluating, and documenting design choices, and external activities, such as understanding stakeholder needs, supporting development teams, and following technological developments [24]. SWEBOK v4 provides a complementary knowledge-oriented perspective through its Software Architecture knowledge area, which covers architectural concepts, styles and patterns, architectural design, representation, evaluation, quality attributes and trade-offs, and architectural decisions [6]. Kruchten's characterization and the Software Architecture knowledge area of SWEBOK v4 provide a basis for examining the competencies developers use in software architecture work. However, it remains unclear which of these competencies developers apply when intervening in agentic AI workflows, how they apply them in these interactions, and whether additional skills emerge that are specific to working with agentic AI tools.

3. Preliminary Work and Results to Date

Two prior studies have shaped both the direction and the empirical grounding of the proposed work. Galster et al. examined how agentic AI coding tools are configured through repository-level artifacts,

including Markdown-based context files such as `CLAUDE.md` and `AGENTS.md` [2]. Related studies of these context files show that they contain project architecture and other architectural guidance [11, 12]. A second study examined Agent Plans, task-oriented Markdown files that provide guidance for agent execution [13]. The study found architecture and design information, as well as design constraints and decisions, in the analyzed plan files, while requirements clarification and design exploration were not observed in the corpus. Together, these studies show that architectural information appears in both project-level context files and task-oriented planning artifacts. However, they do not examine the skills developers need when intervening in agentic AI workflows for software architecture activities, which motivates the proposed research.

4. Expected Contributions

The research is expected to contribute an empirically grounded understanding of the developer skills involved in agentic AI workflows for software architecture. By combining evidence on how developers intervene in these workflows with the skills associated with those interventions, the research will provide an account of the skills developers need when working with agentic AI tools on software architecture activities. A further contribution is the comparison of these skills with existing software architecture competencies. This will clarify which established competencies are applicable in agentic AI workflows and whether additional skill requirements emerge in this context. The resulting conceptual framework will organize these findings and provide a basis for subsequent empirical research, competency assessment, and education and training.

5. Timeline

Table 1
Planned timeline and research outputs

Period	Planned work and research outputs
Winter 2026/27	(RQ1) Develop and pilot the systematic mapping study protocol, conduct the study, and prepare the first paper on agentic AI for software architecture.
Summer 2027	(RQ2) Develop and pilot the interview study protocol, recruit participants, conduct and analyze the interviews, and prepare the second paper on developer interventions in agentic AI workflows.
Winter 2027/28	(RQ3) Synthesize the findings from RQ1 and RQ2, characterize developer skills, and prepare the third paper.
Summer 2028	(RQ4) Develop and evaluate the conceptual framework and prepare the fourth paper.
Winter 2028/29	Integrate the findings from the four studies and develop the thesis-level discussion.
Summer 2029	Write the remaining dissertation chapters and integrate the individual studies.
Winter 2029/30	Revise and complete the dissertation.
Summer 2030	Finalize and submit the doctoral thesis.

Declaration on Generative AI

During the preparation of this proposal, the author used ChatGPT to paraphrase, reword, and improve writing style. The author used Grammarly to check grammar and spelling. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

- [1] A. E. Hassan, H. Li, D. Lin, B. Adams, T.-H. Chen, Y. Kashiwa, D. Qiu, Agentic software engineering: Foundational pillars and a research roadmap, 2025. doi:10.48550/arXiv.2509.06216. arXiv:2509.06216.
- [2] M. Galster, S. Mohsenimofidi, J. L. Lulla, M. A. Abubakar, C. Treude, S. Baltes, Configuring agentic AI coding tools: An exploratory study, in: Proceedings of the 3rd ACM International Conference on AI-Powered Software, Association for Computing Machinery, 2026, pp. 11–20. doi:10.1145/3805760.3814887.
- [3] N. A. Otoum, N. El-Khalili, Methods and techniques of agentic software engineering: A systematic literature review, IEEE Access 14 (2026) 7443–7465. doi:10.1109/ACCESS.2026.3652325.
- [4] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 4 ed., Addison-Wesley Professional, 2021.
- [5] F. Tian, P. Liang, M. A. Babar, Relationships between software architecture and source code in practice: An exploratory survey and interview, Information and Software Technology 141 (2022) 106705. doi:10.1016/j.infsof.2021.106705.
- [6] H. Washizaki (Ed.), Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0, IEEE Computer Society, 2024. URL: <https://www.swebok.org>.
- [7] M. N. Raida, Z. N. Sristy, N. Ulfat, S. M. A. Monisha, M. J. I. Mostafa, M. N. Haque, A study on classifying stack overflow questions based on difficulty by utilizing contextual features, Journal of Systems and Software 208 (2024) 111884. doi:10.1016/j.jss.2023.111884.
- [8] S. Dhanorkar, S. Passi, M. Vorvoreanu, Human oversight of agentic systems in practice: Examining the oversight work, challenges, and heuristics of developers using software agents, in: Proceedings of the 2026 ACM Conference on Fairness, Accountability, and Transparency, ACM, 2026, pp. 6438–6465. doi:10.1145/3805689.3812402.
- [9] D. Planötscher, X. Wang, M. Adil, P. Gregory, S. Larimian, Human-AI collaboration in software development activities: Perspectives of agile practitioners, in: Agile Processes in Software Engineering and Extreme Programming, volume 578 of *Lecture Notes in Business Information Processing*, Springer, Cham, 2026, pp. 274–289. doi:10.1007/978-3-032-22375-3_17.
- [10] G. V. Pereira, V. Jackson, R. Prikladnicki, A. van der Hoek, L. Fortes, C. Araújo, A. Coelho, L. Chelli, D. Ramos, Exploring genai in software development: Insights from a case study in a large brazilian company, in: 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2025, pp. 330–341. doi:10.1109/ICSE-SEIP66354.2025.00035.
- [11] S. Mohsenimofidi, M. Galster, C. Treude, S. Baltes, Context engineering for AI agents in open-source software, in: Proceedings of the 23rd International Conference on Mining Software Repositories, ACM, 2026, pp. 194–198. doi:10.1145/3793302.3793350.
- [12] H. V. F. dos Santos, V. Costa, J. E. Montandon, M. T. Valente, Decoding the configuration of AI coding agents: Insights from Claude code projects, in: Proceedings of the 2026 International Workshop on Agentic Engineering, Association for Computing Machinery, 2026, pp. 63–67. doi:10.1145/3786167.3788412.
- [13] M. A. Abubakar, S. Mohsenimofidi, J. L. Lulla, J. M. Zhang, C. Treude, S. Baltes, M. Galster, An exploratory study of agent plans for agentic AI coding tools in open-source software, 2026. doi:10.48550/arXiv.2608.04661. arXiv:2608.04661, accepted at the 20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026), Emerging Results, Vision, and Reflection Papers Track.
- [14] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008), 2008, pp. 68–77. doi:10.14236/ewic/EASE2008.8.
- [15] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2009) 131–164. doi:10.1007/s10664-008-9102-8.
- [16] J. M. Morse, The significance of saturation, Qualitative Health Research 5 (1995) 147–149.

doi:10.1177/104973239500500201.

- [17] P. Mayring, *Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution*, Klagenfurt, 2014. URL: <https://nbn-resolving.org/urn:nbn:de:0168-ssolar-395173>.
- [18] S. Baltes, S. Diehl, Towards a theory of software development expertise, in: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018*, Association for Computing Machinery, New York, NY, USA, 2018, pp. 187–200. doi:10.1145/3236024.3236061.
- [19] D. S. Cruzes, T. Dybå, Research synthesis in software engineering: A tertiary study, *Information and Software Technology* 53 (2011) 440–455. doi:10.1016/j.infsof.2011.01.004.
- [20] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*, Springer, Berlin, Heidelberg, 2014. doi:10.1007/978-3-662-43839-8.
- [21] J. Jeuring, R. Groot, H. Keuning, What skills do you need when developing software using ChatGPT? (Discussion Paper), in: *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, ACM, 2024, pp. 1–6. doi:10.1145/3631802.3631807.
- [22] P. Vaithilingam, T. Zhang, E. L. Glassman, Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, in: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, Association for Computing Machinery, 2022, pp. 1–7. doi:10.1145/3491101.3519665.
- [23] S. Barke, M. B. James, N. Polikarpova, Grounded copilot: How programmers interact with code-generating models, *Proceedings of the ACM on Programming Languages* 7 (2023) 85–111. doi:10.1145/3586030.
- [24] P. Kruchten, What do software architects really do?, *Journal of Systems and Software* 81 (2008) 2413–2416. doi:10.1016/j.jss.2008.08.025.