Beyond Code Generation: Understanding AI-First Software Development in Practice

Ayumi Aoki
Federal University of Amazonas
Manaus, AM, Brazil
ayumi.santana@icomp.ufam.edu.br

Nabson Silva
Federal University of Amazonas
Manaus, AM, Brazil
nabson.paiva@icomp.ufam.edu.br

Márcia Lima
Amazonas State University
Manaus, AM, Brazil
msllima@uea.edu.br

Tayana Conte
Federal University of Amazonas
Manaus, AM, Brazil
tayana@icomp.ufam.edu.br

Abstract

The rise of AI-based coding assistants has accelerated the AI-first movement, in which AI evolves from a supporting tool to a central driver of software development activities. In this context, approaches such as Spec-Driven Development (SDD) position specifications as key artifacts for structuring interactions with AI and guiding implementation. However, empirical evidence on how this shift is reshaping software development practices remains limited. This study investigates how professionals use AI in AI-first development contexts, examining AI-first practices, levels of specification rigor, transformations in development processes, and perceived benefits and challenges. We conducted a qualitative survey with seven industry practitioners from different companies. Our findings show that SDD principles are already emerging in real-world AI-first practices, revealing varying levels of specification rigor across organizations. Beyond productivity gains, we observed broader transformations in software development, including shifts in developer responsibilities and increasing reliance on artifacts such as Product Requirements Document (PRD), prompts, and plans. We also identified emerging challenges related to cognitive overload, contextual limitations, validation bottlenecks, and organizational pressure. These early findings provide empirical evidence of how AI-first practices are reshaping software processes and expand our understanding of the role of specification rigor in this paradigm.

Keywords

Software development, AI-first, Spec-Driven Development, AI coding assistants

1 Introduction

Recent advances in generative AI are reshaping software engineering practices [3]. Tools such as GitHub Copilot [8] and Claude [2] support implementation, debugging, testing, and solution exploration [17]. AI is also moving beyond isolated assistance toward active participation in planning, execution, and validation [19]. AI-first development reflects this transition toward a more central role for AI in software work [4]. Unlike opportunistic prompt-based use, this paradigm involves sustained integration into development workflows [11]. Developers may define context, steer AI interactions, and validate generated outputs [20]. These changes reshape responsibilities and alter the coordination of software activities [3].

AI-first development introduces engineering challenges. Code generation depends heavily on input quality [12]. Ambiguous requirements can compromise reliability [21]. Contextual limitations affect AI performance in software tasks [10]. Spec-Driven Development (SDD) emphasizes specifications as central development artifacts used to guide implementation decisions [15]. PRDs, technical specifications, planning documents, and structured prompts may support AI-mediated development.

Existing AI-first proposals and SDD approaches describe structured workflows for software development [4, 15]. However, software development involves heterogeneous tasks, organizational constraints, and evolving practices that shape AI use in industry [3]. Consequently, limited empirical evidence exists on how software professionals operationalize AI-first development, how specification artifacts support these activities, and how AI-first adoption is transforming software development processes.

To address this gap, we conducted a Qualitative Survey [13] with seven software professionals across six companies. This study investigates how practitioners use AI in AI-first development contexts, focusing on adopted practices, specification rigor, transformations in development processes, and perceived benefits and challenges.

Our findings reveal diverse AI-first adoption patterns across the industry. Some professionals rely on ad hoc prompt-based interactions. Others adopt structured workflows using planning artifacts and persistent specifications. AI-first adoption shifts developer work toward supervision, validation, and context engineering. Technical and organizational challenges remain significant, including trust, contextual limitations, and workflow adaptation. The main contributions of this paper are early empirical evidence on how AI-first development is operationalized in industry; a characterization of specification rigor in AI-mediated development; the identification of changes in software development work; and technical and organizational challenges to AI-first integration.

2 Background

This section presents the key concepts that underpin this study. It discusses AI-first software development and SDD, an approach that structures software development around specifications.

2.1 AI-First Software Development

Historically, software development has relied on process models such as Waterfall, Spiral, and Incremental development [1, 18],

collectively referred to in this study as traditional software development models. Within these models, activities such as specification, development, validation, and evolution are primarily carried out by human teams [20]. More recently, AI assistants have become part of development workflows, mainly as support tools. For example, Ross et al. [17] describe the use of AI assistants for code explanation, debugging, code generation, and exploring implementation alternatives. In this context, AI assistants support developer activities rather than assuming a central role in the development process.

The AI-first approach denotes a major shift away from traditional software development models. Rather than serving solely as a support tool, AI assumes a central role in the development process, participating in activities such as planning and testing [18]. Biel et al. [4] characterize this transformation as a progression in which AI shifts from an extension of human labor to the primary driver of software work. Likewise, Spera and Agrawal [19] defines AI-first systems as sociotechnical architectures in which AI agents perform tasks, coordinate processes, and make decisions, while humans retain supervisory and governance responsibilities.

It is also important to distinguish AI-first development from vibe coding. According to Kshetri and Voas [11], vibe coding is a prompt-driven approach focused on rapid code generation and experimentation, in which humans iteratively guide and refine generated outputs. In contrast, AI-first entails a redistribution of operational responsibilities, with AI serving as an active component of software execution rather than merely supporting isolated coding tasks [4, 18]. Although AI-first has gained attention as an emerging software development perspective [19], how it reshapes software development processes in practice remains less understood.

2.2 Spec-Driven Development

SDD, as defined by Piskala [15], is an approach in which specifications, understood as explicit descriptions of intended system behavior, guide implementation, verification, and maintenance activities. In this approach, specifications are treated as central development artifacts rather than auxiliary documentation. Piskala also characterizes SDD by levels of specification rigor, based on the role they play across the development lifecycle:

Spec-First: the specification is created before implementation to guide development, but may not be maintained afterward, allowing the code to become the primary artifact again.

Spec-Anchored: the specification remains synchronized with the code throughout the system lifecycle, requiring changes to update both artifacts and making the specification living and trustworthy documentation.

Spec-as-Source: the specification becomes the primary artifact edited by humans, while code is generated automatically. Behavioral changes require updating it and regenerating the implementation.

These levels illustrate how the role of specifications can vary in SDD. Across all of them, the central principle remains the same: shifting specifications from a passive artifact to an active mechanism for guiding development.

3 Study Design

Our goal is to understand how AI is being integrated into software development work in AI-first contexts. Therefore, we conducted a Qualitative Survey [13] with seven participants from different companies. To guide the study, we developed the following Research Questions (RQs):

RQ1: **What levels of specification rigor do developers adopt in AI-first contexts?**
RQ1 examines which levels of specification rigor developers adopt in their AI-first practices, allowing us to understand how closely these practices align with SDD principles and how they are adapted in practice.

RQ2: **How is AI-first development transforming the software development process?**
RQ2 investigates how AI-first development affects the software development process beyond code generation, focusing on broader changes in how software development is organized and performed in AI-first contexts.

RQ3: **What benefits and challenges do software professionals experience in AI-first development?**
Finally, RQ3 discusses the benefits and challenges associated with adopting AI-first development.

Data Collection. We conducted seven semi-structured interviews with software professionals. To guide the interviews, we developed an interview script with questions designed to characterize the participants and understand how their organizations adopt AI-first development practices. In addition to the interview questions, we created a checklist to ensure consistent coverage of all study topics across interviews. We created an initial version of the script and refined it through a pilot study with P6 (see Table 1), mainly to clarify the wording of the questions. Since P6's responses were not affected by the initial version of the script, we included this interview in the data analysis. Both versions of the interview script are available in the supplementary material (Section 7).

Participant Selection. We recruited participants through convenience sampling [23]. To characterize participants as working in AI-first contexts, we verified whether they regularly used AI in software development activities and whether their companies allowed, encouraged, or supported its use. Before each interview, participants signed a Consent Form describing the study purpose, interview procedures, voluntary and anonymous participation, the right to withdraw, and the use of the collected data in qualitative analysis and scientific publications. The Consent Form is available in our Supplementary Material (Section 7).

Execution. The first author conducted all interviews through online meetings. Initially, we invited eight practitioners to participate in the study. However, one professional declined due to time constraints, resulting in seven interviews. Each interview lasted approximately 20 minutes. During the interviews, the interviewer followed the semi-structured script and used the checklist to record the topics covered by each participant. With participants' consent, we recorded the interviews for transcription and analysis.

Data Analysis. Although the interview guide covered broader aspects of AI use in software development, the analysis reported in this paper focused on excerpts directly related to the RQs. Other questions were used to characterize the participants and their development contexts. We transcribed the interviews and conducted open and axial coding [6]. The first author selected excerpts relevant to each RQ and assigned open codes that captured their

meaning. Related codes were compared and grouped through axial coding into the categories highlighted in bold in the results. The co-authors reviewed and refined the codes and categories through peer-debriefing sessions. For the analysis of specification rigor, we interpreted the open codes using concepts from the software specification literature after the initial coding. The quotes, codes, and categories are available in our Supplementary Material.

4 Results

Table 1 characterizes the participants, identified as P1 to P7. The sample includes software developers, technical leaders, and one engineering manager, with professional experience ranging from four to twelve years. This role diversity allowed the study to capture AI-first practices from different perspectives, including implementation, technical coordination, and engineering management.

Table 1: Characterization of participants by role, years of experience, and company

Participant	Role	Exp.	Co.
P1	Engineering Manager	9 years	C1
P2	Software Developer	4 years	C2
P3	Software Developer	5 years	C3
P4	Software Developer	5 years	C4
P5	Software Developer	6 years	C5
P6	Technical Leader	6 years	C6
P7	Technical Leader	12 years	C6

The participants represented six companies (C1–C6), since two participants worked in the same organization. As shown in Table 2, the sample includes companies varying in size, time in market, and business domain. This diversity broadens the contextual coverage of the findings, as AI adoption practices may be shaped by organizational structure, scale, and development processes [3].

Table 2: Characterization of companies by sector, years in market, and approximate number of employees

Co.	Sector	Years	Employees
C1	Financial technology	8+	1,500+
C2	Legal technology	15+	1,000+
C3	Research and development	20+	1,000+
C4	Software engineering services	20+	10,000+
C5	Technology products	50+	20,000+
C6	Financial technology	10+	1,000+

In the following sections, we present the results and answers of each RQ.

4.1 Specification Rigor in AI-First Development

In light of the SDD rigor spectrum (see Section 2.2), the reported *AI-first* practices did not concentrate on a single specification level. The level of rigor varies according to the type of development activity, task complexity, and how AI is integrated into the workflow. As P1 explained: "*It depends on the task. If it is a very specific task (...) what I usually do is locate the source code (...) I use that as context and say: 'I need this to be changed (...)'. If it is a more complex task (...) I use planning models [an AI agent for planning] to generate an implementation plan and tasks, and then switch to an agent model [an AI agent for implementation] that executes that plan*".

***Ad hoc* practices** were observed mainly in isolated activities, such as interface adjustments and modifications to individual components. In these cases, the specification remains implicit in the context provided to the agent, and AI is invoked through direct prompts with minimal formalization. P4 exemplifies this scenario by stating: "*Let us suppose I have a very simple task (...) it is just creating a button (...) the AI can generate that component for me without many adjustments*". Similarly, P3 described a reactive use to solve a specific technical problem: "*I encountered the problem (...) opened the project in the agent (...) and asked it (...) to implement the connection using Wi-Fi Direct*". However, the lack of guidance may compromise the adequacy of the generated solution. As P4 observed: "*I asked it to generate a component, but I did not provide much context (...) it generated the component the way it thought it should be generated (...) but it did not have much functionality*".

***Spec-first* practices** characterized more complex activities, such as developing new features or workflows that required prior alignment. Unlike ad hoc practices, participants explicitly structured the intended solution before implementation. P5 illustrates this type of practice by describing the use of AI to transform available context into a structured Product Requirements Document (PRD): "*I provide all the context (...) I generate a very complete PRD (...) I use the LLM itself to make corrections until I have a PRD that I consider ideal*". Similarly, P7 described a process in which AI was used to "*develop a plan [for implementation]*" and that "*I kept validating it [the plan] (...) and then I actually asked it [the AI] to implement*". In both cases, the specification guides initial development, but the reports do not indicate its continuous use throughout execution.

***Spec-anchored* practices** were observed in activities involving technical refinement, team alignment, and architectural refactoring. Unlike spec-first practices, the specification continues to be used across system lifecycle. P1 described this practice by reporting that "*I used AI from the conception stage (...) planning models to ask the AI to challenge my architecture (...) I spoke with my team and presented the plan*". After this alignment, the participant stated that "*after that I used this refined plan to ask the AI to implement some generic hooks following that architecture*" and that "*I used this component as a reference together with my architecture document to carry out other refactorings in my codebase*".

Participants also reported adopting structured frameworks for AI-first development workflows. P5 described using Build More Architect Dreams (BMAD) [5] to support development: "*we use this framework to support everything from clarifying the initial idea to creating the PRD, and from there we move into development*". Similarly, P7 stated that the workflow is "*very focused on PRD, Tech spec, stack breakdown*". The participant also reported using specification-oriented frameworks such as Spec Kit [9] and Open Spec [14]: "*today we use Spec Kit a little for some things (...) I tried running an experiment using Open Spec (...) and it was really nice*".

Sufficient evidence was not identified to characterize ***spec-as-source* practices** in the analyzed cases. Although some reports indicate higher artifact-driven automation, such as the use of PRDs,

templates, or tool integrations, participants still described continuous human intervention and validation. This distinguishes these practices from a model in which the specification becomes the primary source for system generation and evolution.

RQ1 Summary: Developers vary in the specification rigor they apply to tasks. Ad hoc practices were more common in isolated activities, while spec-first practices appeared in complex tasks requiring prior alignment, such as new features or workflows. Spec-anchored practices emerged in activities involving technical refinement, team alignment, and architectural refactoring.

4.2 Transformations in Software Development

AI as a central development actor has become part of different stages of development, rather than being used only in isolated situations. As P3 reported, its use occurs in "*general (...) development, bug fixing, development of new features, and I have also used it for unit testing*". In AI-first scenarios, AI becomes central to development activities, making the main concern how to provide the context, constraints, and instructions it needs to complete tasks effectively. As P7 observed, "*the day-to-day process (...) changed, because we are no longer thinking so much about solving a problem directly (...) but rather (...) how the AI will solve that type of problem*". P3 further distinguished this process from chat-based use: "*It is different from ChatGPT (...), you ask a question, it gives you an answer. With this agent, no. You provide a prompt, it explains things to you, and then asks for confirmation before changing the file*". P2 also stated that they "*use it every day, all day, because that is what people at work recommend*". P5 also reported: "*today I do not code, I only talk to the LLM*". P2 reinforced this change by highlighting that "*we do not actually write code, we write the prompt that will write the code*".

The redistribution of responsibilities and the changing role of development artifacts result from the central role of AI in development workflows. P5 stated "*today I perform the role of the PO, the architect, the developer, and the tester*". The participant observed that "*the job market is still not prepared for this*", since "*the developer is taking on all this responsibility*". As responsibilities shift, the role of development artifacts also changes. PRDs, spikes, and prompts take an active role in development. P6 highlighted the PRD's centrality by stating that "*the PRD (...) is the definition of what needs to be done*". P4 described the use of spikes as a technical investigation stage: "*Based on the requirements, I create the spike (...) I will investigate how to implement this requirement within my existing code*". P2 added that "*nowadays we are translating this [product] idea into a very high-level algorithm (...) a step by step, for example, in Markdown, which will become code*".

Iterative planning and execution flows involve planning, validation, and AI-supported execution. P2 described this flow by stating that "*it [the AI] plans, I correct the plan, and then it executes*". P4 complemented this by highlighting that implementation may be preceded by validation of the generated plan: "*I am not simply going to tell the AI to implement this button. I will tell it: create a plan for how you are going to implement this button (...) I'm going to validate if this plan makes sense*".

Review and validation practices are also being reformulated. P1 reported that, in *pull requests*, "*you have AI reviewing your code*" and that "*sometimes they catch really interesting things*". At the same time, P6 highlighted that, despite discussions about AI-supported review, an external human perspective remains important: "*you were not the one who typed it. But you were the one who told it to do it that way (...) it did what you told it to do. So, there is still value in an external perspective*".

RQ2 Summary: AI-first development shifts AI from assistance to task execution. Developers increasingly focus on guiding, reviewing, and validating AI-generated artifacts rather than directly implementing every solution. As AI becomes central to the workflow, responsibilities and role boundaries are redistributed, affecting practices such as specification, implementation, and code review.

4.3 Benefits and Challenges of AI-First Development

4.3.1 Technical Perspectives. **Technical benefits** include the acceleration of repetitive tasks, code generation, and improved understanding of the system. P1 stated that AI "*saves me a lot of time by delegating simple tasks*", while P3 highlighted that it "*greatly increases productivity for some activities*". Beyond implementation, P1 described using AI to understand product behavior from the code: "*it finds the implementation of that action and tells me exactly (...) like reverse engineering (...) it writes the product through the code*".

Technical limitations concern the reliability of AI-generated outputs and the ability of AI to interpret the development context. P3 observed that "*sometimes it generates redundant code, with code that does nothing*". P6 reflected on whether correcting these errors might require more effort: "*would I not take longer having to keep fixing it than if I had generated it correctly the first time?*". Beyond quality, participants highlighted limitations related to missing contextual and organizational knowledge. P1 pointed out that AI may lack important project background information: "*the AI will never have that information, and sometimes that information is important*". P2 illustrated how this limitation can affect development decisions: "*the AI was not able to detect that priority and order of task execution*". Similarly, P6 emphasized limitations in domain understanding, stating that "*it does not understand all the business rules*".

4.3.2 Cognitive and Human Perspectives. **Cognitive support** emerged from the use of AI to assist reasoning and solution exploration. P5 stated that "*we gain a lot in productivity for solving small problems and more complex things; we gained, perhaps, a reasoning partner*". Similarly, P3 highlighted the use of AI in situations where some guidance was needed, stating: "*I like using AI in situations like this, when we sometimes need some guidance*".

Cognitive load and human challenges were also associated with AI use. P2 stated that "*it is less manual work, but I think it is much more cognitive work*". P7 also highlighted that "*it generates a bit of (...) mental fatigue (...) there is too much new stuff at the same time*". This scenario was associated with the rapid evolution of tools and a sense of Fear of Missing Out (FOMO). As P7 reported, "*as a negative point (...) a bit of FOMO (...) every moment a new model appears (...) it becomes madness*". P6 added: "*I think the problem is that there are too many tools (...) it still creates this impostor syndrome that I cannot be completely productive*". Participants also reported difficulties in

prompt formulation. P2 mentioned difficulty in "*getting the thought (...) out of the head and into the prompt*", while P4 stated that "*the biggest difficulty is (...) being able to describe the problem*". P7 also highlighted difficulties in feature decomposition: "*they still need to (...) actually understand how to break features into deliverable pieces (...) because things are coming in too large*".

4.3.3 Process and Organizational Perspectives. **Process and organizational benefits** were mainly associated with increased delivery speed. As P5 highlighted, "*perhaps the business perspective is that we can also get things done and deliver faster*". P3 also reported AI use in high deadline-pressure scenarios: "*we need to deliver these POCs by Tuesday, and AI ends up being a huge help, especially for repetitive things that we (...) would just waste time developing (...) and could instead be spending time on something more complex*".

Process and organizational challenges indicate that faster implementation does not eliminate process bottlenecks. P5 stated that "*even though we can do several things at the same time, we still face the difficulty of having to review everything afterward*". P6 reinforced this concern by questioning whether speed gains remain meaningful when review effort is considered: "*it is not enough to just generate code if everyone still has to review that generated code, and that review takes time (...) generating the code quickly does not help much*". Quality concerns also emerged, as P5 observed that "*the rush to get the product quickly can end up compromising code quality*". P5 reported increased productivity pressure: "*the problem is that it became almost too easy, and then we end up taking on too many activities, we end up being pushed a bit more*".

RQ3 Summary: AI-first development brings productivity, speed, and reasoning support. However, professionals also face unreliable outputs, increased review effort, cognitive overload, and process uncertainty. Thus, AI reduces effort in some tasks, but shifts work toward validating, refining, and governing AI-generated artifacts.

5 Discussion

This section discusses how AI-first development differs from AI-assisted use, how specification rigor varies across workflows, and the practical implications of these findings.

5.1 From AI-assisted to AI-first development

AI-first development represents a shift beyond using AI as a programming assistant. Rather than supporting isolated tasks, it reflects a broader integration of AI into software development workflows. Banh et al. [3] discuss how generative AI has already been incorporated into activities such as code generation, debugging, and solution exploration in Software Engineering. Our results extend this view by showing developers coordinating AI agents across multiple stages of development. In this setting, developers act less as direct implementers of every artifact and more as reviewers, refiners, and orchestrators of AI-generated specifications, code, tests, and fixes. Based on the business understanding provided by product roles, developers guide the AI to elaborate specifications, generate implementation alternatives, validate outputs, and iteratively correct problems. AI-first development therefore changes how software work is prepared, delegated, and verified.

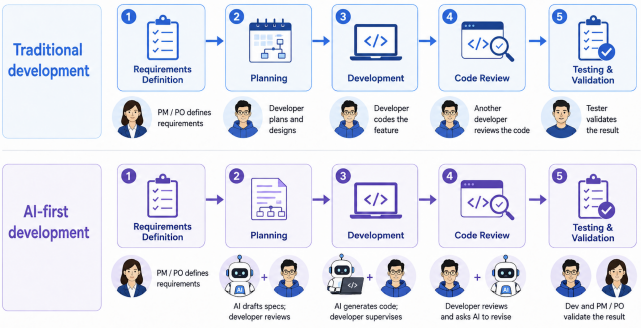

Figure 1: Comparison between traditional and AI-first development lifecycles. PM refers to Product Manager and PO to Product Owner.

Participants' reports indicate a redistribution of responsibilities in AI-first workflows. In more traditional workflows, activities such as specification, implementation, testing, review, and deployment were clearly distributed across roles [18]. In some reported cases, these boundaries became less defined. The implementation, test generation, artifact production, and exploration of technical alternatives were delegated to AI. In contrast, developers assumed more responsibility for context construction, instruction definition, result validation, and oversight of AI-supported decisions. As illustrated in Figure 1, this shift reshapes software development by redistributing responsibilities and reorganizing the workflow. Developers may move from direct execution toward orchestration, review, and governance of AI-supported work. These results resonate with the AI-first cycles proposed by Piskala [15], in which agents assume tasks across different stages of the software development lifecycle, and with the transformations in Software Engineering driven by closer human–AI integration discussed by Terragni et al. [20].

This redistribution of responsibilities was associated with both perceived benefits and new demands. Participants reported higher productivity, faster implementation, and support for understanding systems and generating artifacts. These findings corroborate results reported by Banh et al. [3] on productivity gains, time savings, and support for development activities through generative AI. In our results, however, reduced manual implementation was accompanied by greater involvement in planning, context construction, and validation. Participants described reviewing AI-generated plans before execution, refining instructions, and checking generated code and tests. Contextual limitations and unreliable outputs also required correction and additional review. Thus, AI did not simply remove effort from development activities; it shifted part of that effort from direct implementation to guiding, reviewing, and validating AI-generated artifacts.

Ullah et al. [22] show that AI-driven performance gains depend on factors such as collaboration, organizational capabilities, and workforce adaptation. Likewise, Dzhusupova et al. [7] shows that coordination, governance, and integration barriers can compromise AI initiatives. Our results reinforce this discussion by showing that *AI-first* adoption varies across teams and organizational contexts, reflecting differences in process maturity, institutional expectations, and supervision mechanisms. This suggests that the transition

toward *AI-first* development may depend less on advances in model technology and more on the organizational capacity to reconfigure practices and roles for consistent AI integration.

5.2 Specification in AI-first Development

Although AI is often associated with faster implementation and reduced manual effort, our findings indicate that its adoption may require more structured specification practices. Specification rigor varied across the reported workflows. Direct prompts were mainly used for isolated tasks, while complex features, architectural decisions, and refactoring involved PRDs, technical specifications, acceptance criteria, plans, and structured prompts. In some cases, these artifacts remained active during implementation, indicating spec-anchored practices. The level of rigor varied with task complexity and AI integration.

Larbi et al. [12] show that ambiguous, incomplete, or contradictory descriptions compromise the correctness of generated code, even in advanced models. When the model's interpretation diverges from the specification's intent, the implementation may deviate from expected behavior [21]. In our study, participants addressed these risks through task decomposition, prompt refinement, additional context, and structured templates. Some also reviewed and corrected AI-generated plans before implementation, placing validation within the specification process. Therefore, requirements engineering and specification-centered practices may become increasingly important in AI-first software development.

In the reported cases, specifications not only aligned human stakeholders and guided developers but also instructed AI agents to generate, revise, and validate code. Developers translated the business understanding provided by product roles into explicit and actionable specifications for AI-supported implementation. This indicates that SDD may become more relevant as AI assumes greater implementation responsibility. Frameworks such as Spec Kit and OpenSpec illustrate attempts to organize development around this logic. However, we found no clear evidence of *spec-as-source* practices, since specifications still required continuous human interpretation, refinement, and validation. This may reflect limitations in tool maturity, trust, integration, or organizational support.

5.3 Practical Implications

The findings have practical implications for teams adopting AI-first development. First, context preparation and specification should be treated as part of implementation, not as optional documentation. Explicit requirements, constraints, acceptance criteria, and architectural decisions can more clearly guide AI agents. Second, verification should occur throughout the workflow, covering specifications, plans, generated code, and tests. Third, organizations should define who reviews and approves AI-generated artifacts. Finally, reusable prompts, instructions, agent configurations, and validation criteria should be managed as project artifacts to improve consistency and traceability.

6 Limitations

One limitation of this study concerns the formulation of the interview questions. There was a risk that the questions would not adequately address the aspects investigated in this research or that participants would not properly understand them. To mitigate this limitation, a senior researcher reviewed the interview script, and we conducted a pilot study to refine the questions. Another limitation concerns the interpretive nature of qualitative analysis, which may introduce researcher bias. To mitigate this risk, the researchers conducted iterative discussions and peer-debriefing sessions to review, refine, and validate the generated codes.

The number of participants can also be considered a limitation. To mitigate this limitation, we selected participants from six different software organizations and included professionals with different roles. Furthermore, according to the ACM SIGSOFT Empirical Standards [16], in Qualitative Surveys, statistical generalization is not expected. Thus, the sample of seven participants is suitable for generating preliminary insights into AI-first development practices.

7 Conclusion

This paper investigated how software professionals use AI in software development within AI-first contexts. We conducted a Qualitative Survey with seven professionals from six organizations. We analyzed AI-first practices, specification rigor, transformations in development processes, and perceived benefits and challenges.

Results show that AI is moving beyond isolated coding assistance to execute tasks across development workflows. As AI assumes a broader role, teams use different levels of specification rigor aligned with SDD principles, depending on the activity and context. This shift also changes responsibilities, artifacts, and workflows, while introducing technical, cognitive, and organizational trade-offs. Therefore, this study provides evidence on how *AI-first* development and SDD are being adopted in industry and how specifications support coordination between developers and AI.

Future work should investigate how these practices evolve over time, whether teams progress toward higher levels of specification rigor, and which artifacts and contextual information best support reliable AI-generated outputs. Further studies should also examine the effectiveness and trade-offs of specification-driven frameworks and the technical and organizational barriers to more autonomous practices, particularly *spec-as-source*.

Artifact Availability

The interview scripts, consent form, and codebook are available at https://zenodo.org/records/21526157.

Acknowledgements

We thank the study participants and USES Research Group members for their support. We also acknowledge the use of OpenAI's ChatGPT, based on the GPT–5.5 Thinking model, to generate Figure 1. All generated content was reviewed by the authors, who assume responsibility for the final manuscript, in accordance with CNPq's Policy on Integrity in Scientific Activity (Ordinance No. 2.664/2026). CAPES supported this study-Finance Code 001; CNPq processes 314797/2023–8, 443934/2023–1, and 445029/2024–2; the Amazonas State Research Support Foundation (FAPEAM) through POSGRAD 2026—2027; and Amazonas State University through the Academic Productivity Program 01.02.011304.026472/2023–87.

References

[1] Adel Alshamrani and Abdullah Bahattab. 2015. A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. *International Journal of Computer Science Issues (IJCSI)* 12, 1 (2015), 106.

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

[3] Leonardo Banh, Florian Holldack, and Gero Strobel. 2025. Copiloting the future: How generative AI transforms Software Engineering. *Information and Software Technology* 183 (2025), 107751.

[4] Stefan Biel, Alexander Hahn, Volker Bilgram, and Helen Rogers. 2025. From hybrid to AI-first innovation management. *IEEE Engineering Management Review* 53, 5 (2025), 58–64.

[5] BMAD Method. 2026. Welcome to the BMad Method. https://docs.bmad-method.org/

[6] Juliet Corbin and Anselm Strauss. 2014. *Basics of qualitative research: Techniques and procedures for developing grounded theory.* Sage publications.

[7] Rimma Dzhusupova, Jan Bosch, and Helena Holmström Olsson. 2022. Challenges in developing and deploying AI in the engineering, procurement and construction industry. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. 1070–1075. doi:10.1109/COMPSAC54236.2022.00167

[8] GitHub. 2026. GitHub Copilot. https://github.com/features/copilot

[9] GitHub. 2026. Spec Kit. https://github.github.com/spec-kit/

[10] Nam Huynh and Beiyu Lin. 2025. Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications. *arXiv preprint arXiv:2503.01245* (2025).

[11] Nir Kshetri and Jeffrey Voas. 2026. Is Vibe Coding the Future of Software? *Computer* 59, 3 (2026), 100–104.

[12] Maya Larbi, Amal Akli, Mike Papadakis, Rihab Bouyousfi, Maxime Cordy, Federica Sarro, and Yves Le Traon. 2025. When prompts go wrong: Evaluating code model robustness to ambiguous, contradictory, and incomplete task descriptions. *arXiv preprint arXiv:2507.20439* (2025).

[13] Jorge Melegati, Kieran Conboy, and Daniel Graziotin. 2024. Qualitative surveys in software engineering research: definition, critical review, and guidelines. *IEEE Transactions on Software Engineering* 50, 12 (2024), 3172–3187.

[14] OpenSpec. 2026. OpenSpec. https://openspec.dev/

[15] Deepak Babu Piskala. 2026. Spec-Driven Development: From Code to Contract in the Age of AI Coding Assistants. *arXiv preprint arXiv:2602.00180* (2026).

[16] Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, et al. 2021. ACM SIGSOFT Empirical Standards Released. *SIGSOFT Softw. Eng. Notes* 46, 1 (Feb. 2021), 19. doi:10.1145/3437479.3437483

[17] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th international conference on intelligent user interfaces*. 491–514.

[18] Ambar Nath Saha and Debashis Patra. 2026. AI-First Software Development Lifecycle: An Agent-Driven Framework for Autonomous Planning, Coding, Testing, and Deployment. *ESP Journal of Engineering & Technology Advancements* 6, 1 (2026), 131–139.

[19] Cosimo Spera and Garima Agrawal. 2025. Reversing the Paradigm: Building AI-First Systems with Human Guidance. *arXiv preprint arXiv:2506.12245* (2025).

[20] Valerio Terragni, Annie Vella, Partha Roop, and Kelly Blincoe. 2025. The future of ai-driven software engineering. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–20.

[21] Zhao Tian and Junjie Chen. 2025. Aligning Requirement for Large Language Model's Code Generation. *arXiv preprint arXiv:2509.01313* (2025).

[22] Aziz Ullah, Xiaoming Sun, Wang Yalan, and Adnan Ali. 2026. AI-driven performance in organizations: Unveiling the role of team dynamics, innovation, and commitment using a hybrid approach. *International Journal of Information Management* 89 (2026), 103059.

[23] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.