World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

Check for updates

Layer 1 of the AI-First SDLC: An agent-driven product and designer layer for autonomous discovery, requirement refinement, experience design and backlog generation

Ambar Nath Saha [1] and Debashis Patra [2,*]

[1] *Information Technology, Cognizant Technology Solutions Canada Inc, Nova Scotia, Canada.*
[2] *Computer Science, UT Austin, TX, USA.*

World Journal of Advanced Research and Reviews, 2026, 30(02), 1355-1364

Publication history: Received on 08 April 2026; revised on 16 May 2026; accepted on 18 May 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.30.2.1381

Abstract

The rising of Agentic AI has a very positive impact on software development and delivery, and organizations are improvising their processes very fast. In our earlier work, we introduced a thirteen-phase AI-First Software Development Lifecycle (SDLC) framework to make the process autonomous with mandatory and optional human-in-the-loop checkpoints, and in this paper, we are going to deep dive into the important entry point which we call the Product and Designer Layer.

Many organizations still follow a manual process to convert simple, high-level business ideas into clear requirements and usable designs in the form of stories and tasks, which leads to misunderstandings and also increases the chance of human error. Moreover, product managers or analysts spend days or weeks refining those high-level and ad-hoc epics before they become ready for engineering.

In this paper, we are going to explain the first module of the AI-First framework that we introduced in our earlier paper. This Product and Designer layer consists of four agents: Product Intent Agent, Research Agent, Design Agent, and Backlog Agent. These agents interact with each other and are managed by an orchestrator.

Starting from a natural-language epic, this layer enriches the initial idea with historical context and generates machine-readable design artifacts aligned with the organization's design system, and finally produces an engineering-ready backlog. The intention is not to replace product owners or designers, but to reduce the translation effort between business, design, and engineering. By doing so, the time required to move from idea to backlog can be reduced from weeks to minutes, while keeping humans firmly in control of the key decisions.

Keywords: Agentic AI; Product Discovery; UX Design Automation; Design Agents; Requirement Engineering; Backlog Generation; Multi-Agent Orchestration; AI-First SDLC; Design Systems; Human-in-the-loop

1. Introduction

Organizations have been implementing automation significantly in software development over the last couple of decades [1,7]. We have seen this especially on the DevOps side with the introduction of CI/CD pipelines and advanced monitoring solutions [7]. However, the very first phase of the lifecycle is still handled by humans, where a simple business idea is converted into clear requirements and usable designs [1,2]. Product managers, analysts, researchers, and designers manually try to convert those ambiguous epics into proper stories and tasks in the backlog, which takes a significant amount of time and human labor [2]. Since this is the starting point of the entire lifecycle, any ambiguity at

* Corresponding author: Debashis Patra

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0.

this stage usually creates bigger problems later, and we end up with buggy code, misunderstandings in the requirements, and sometimes missed deadlines [2,15].

In our earlier paper, we introduced a thirteen-phase AI-First Software Development Lifecycle (SDLC) framework governed by a Central Orchestrator Agent that coordinates specialized agents across the entire lifecycle, starting from requirement ingestion and continuing through to production monitoring [12]. That paper provided the overall blueprint, where each layer was described only at a high level [12]. In this paper, we are going to discuss the first layer in a more detailed way. We believe the first layer is the most important one to get right because it requires contextual understanding, empathy for the end user, and familiarity with the organization's visual and interaction language [4,8,9].

With the recent progress in agentic AI, retrieval-augmented generation, multimodal reasoning, and machine-readable design systems, it is now becoming practical to hand over a significant portion of product and design work to coordinated agents, provided that proper governance and human oversight are in place [5,10,11,16]. Although there are already tools such as Figma AI and Galileo AI that can generate diagrams from simple text, in most cases, these tools work in isolation. They are not aware of an organization's component library, accessibility standards, or how users actually interact with the existing product [4,8,9]. On the requirement engineering side, large language models have shown promising results on synthetic datasets, but there is still limited evidence of how effectively they perform within real enterprise environments that involve live backlog systems and strict audit requirements [3,6,10,15,16].

In this paper, we are proposing an agent-driven architecture where there will be four agents: the Product Intent Agent, the Discovery and Research Agent, the Experience Design Agent, and the Backlog Synthesis Agent [5,11,17]. These agents will be controlled by a Layer 1 Sub-Orchestrator, which finally reports to the Central Orchestrator that controls the entire SDLC [12]. All those agents will be interacting with each other, and together, they take a natural-language epic from the product owner, enrich it with historical insights from the Knowledge Layer, generate user journeys and wireframes aligned with the organization's design system, validate the outputs against accessibility and compliance policies, and ultimately produce a structured backlog that can be directly consumed by the downstream Architecture and Coding agents [4,5,11,17].

It is important to clarify that our intention is not to replace designers or product owners. Instead, all the product owners can spend time brainstorming new ideas, whereas the technical team and architects can invest their time in thinking of an optimum solution to solve the business goal [4,8,9]. To make this discussion more concrete, we use the same example as in our earlier paper—a user notification preference system that allows users to choose how they would like to receive alerts, such as email, SMS, or push notifications [12].

2. Related work

2.1. AI in Product Management

Nowadays, many product management tools such as Productboard, Aha!, and Linear have started adding AI features. These tools can summarize customer feedback, group similar feature requests, and even help in drafting release notes [1,2]. However, they are still mainly helper tools. They do not create engineering-ready artifacts, and the product manager still needs to spend a lot of time refining the requirements [2,3]. Some research has also explored the use of large language models for requirement elicitation, but most of these studies are not connected to real enterprise environments or live backlog systems like Jira [4,5].

2.2. AI in UX and Interaction Design

On the design side, tools like Figma AI, Uizard, and Galileo AI can generate wireframes and workflow diagrams from simple text prompts [6,7], but these tools usually work in isolation. They do not have any way to understand the organization's design system, component library, or accessibility standards [8,9]. In simple terms, those tools generate visual designs without knowing the organizational context or existing design and become useless for the engineering team. Recent advancements in multimodal reasoning and design-token-based generation show that this gap can be reduced, especially when these tools are connected to machine-readable design systems [10,11].

2.3. Multi-Agent Systems for Software Engineering

Many organizations or independent researchers have been exploring multi-agent models in software engineering in recent years [12,13,14]. ChatDev introduced an idea of a virtual company where agents play the roles of CEO, CTO, programmer, and tester [12]. MetaGPT again extended the same concept that ChatDev had and added standard operating procedures on top of that to reduce errors and improve coordination between agents [13]. SWE-Agent

showed that LLM-based agents can resolve real-world GitHub issues [14]. Cognition Labs introduced Devin as an autonomous software engineer to handle end-to-end development tasks.

Although these developments are extraordinary without any question, they mainly focus on the coding part [12,13,14]. We believe that coding is a subset of the entire Software Development Lifecycle. In our vision, an enterprise-ready autonomous system should also include product management, design, governance, and an organizational knowledge repository [15,16].

2.4. Design Systems as Machine-Readable Contracts

On a positive note, there have been several initiatives that make design systems easier for machines to understand [1,2,3]. Projects like Style Dictionary, Tokens Studio, and the World Wide Web Consortium Design Tokens specification allow design systems to be accessed programmatically [1,2,3]. In simple terms, an agent can ask the design system for an approved component, such as a primary button, and receive a structured response.

Our Layer 1 framework uses this concept directly [4]. Instead of generating random or free-form designs, the Experience Design Agent creates structured screen definitions that reference these approved components. As a result, this keeps the design consistent and ensures that everything follows the organization's design standards [1,3,5].

3. Positioning Within the AI-First SDLC

In this section, we will discuss about the first layer of the AI-First Software Development Lifecycle in more depth. This layer will be triggered when a product owner or business analysts write down an epic using natural language and the primary role of this layer is to transform the epic into two artifacts that are going to be used in the rest of the SDLC.

The first artifact is the Product Requirement Bundle (PRB). This artifact captures the problem statement, user journeys, and both functional and non-functional requirements. It also highlights any open questions or assumptions that still need to be clarified.

Another important artifact that this layer is going to produce is the Design Artifact Bundle (DAB), which contains wireframes, structured mockups, and references to the organization's design, accessibility notes, and interaction specifications.

Both of the artifacts are going to be game-changing in the SDLC, as the quality of the product is highly dependent on these two outputs that this layer produces.

3.1. The Layer 1 Sub-Orchestrator

In our earlier framework, a single Central Orchestrator handled the coordination of all agents across the thirteen phases. However, in real-world situations, product and design work rarely follow a straight path. They often involve continuous back-and-forth discussions—clarifying ambiguities in the epic, refining personas, or revisiting wireframes when design changes occur. Routing each of these interactions through the Central Orchestrator can slow things down and introduce unnecessary delays.

To overcome this challenge, we introduce a **Layer 1 Sub-Orchestrator**, which takes care of the internal coordination within the Product and Designer Layer. It will have four specialized agents to complete specific work and it will always get in touch with the Central Orchestrator whenever necessary.

Layer 1 Sub-Orchestrator plays the role of a team lead, and after completing the entire workflow, it reports back to the Central Orchestrator with full transparency. When a new epic comes under its jurisdiction, it reviews the epic and determines what activities need to be performed to create two artifacts from it. After that, it kicks off the chain of activities and oversees the entire workflow. It monitors if there is any low-confidence output and finally stores the two artifacts in the Knowledge Layer. Also, it makes sure the entire process is transparent and fully audited.

3.2. Product Intent Agent

Layer 1 Sub-Orchestrator first activates the Product Intent Agent, and it plays a key role because this agent in the autonomous SDLC acts as a business analyst who reviews the epics and flags if there are any discrepancies. The agent's primary responsibility is to interpret the epic and transform it into a proper, structured, and unambiguous format. It

defines the desired output, targeted user base, business values and impact, and any business constraints that were never thought of while writing the epic.

Additionally, it scans the entire epic and finds out if there is any vague expression like "should be fast" or "similar to our competitor" that is really ambiguous in nature. Based on all the findings, it calculates an ambiguity score, and if the score exceeds a predefined threshold, it generates a list of clarifications and sends those to the product owner for further review. This is our first optional human checkpoint if a significant amount of clarifications is required before the agent goes further. Once all the clarifications are addressed by the product owner, the agent publishes a Normalized Epic Object into the Sub-Orchestrator's working memory.

3.3. Discovery and Research Agent

Our next agent is a Discover and Research Agent and this agent comes into picture when the epic has been normalized. This agent plays a combined role of a user researcher, market analyst and data analyst. Its first responsibility is to check whether this epic is similar to any existing personas from Knowledge Layer and if it doesn't find anything then it creates a new persona and flags it for human review. So, here are other human checkpoints if the entire feature hasn't been implemented in the organization before.

After that, the agent creates end-to-end user journeys, how the user is going to interact with the feature etc. The agent also uses telemetry data from tools such as Datadog, Amplitude, or Mixpanel to enrich the analysis with real usage insights. All the information that is generated by this agent is marked as either Retrieved or Assumed which allows the downstream agents to verify the details when necessary. The final outcome of this stage is the Discovery Brief which will be attached to the Normalized Epic Object.

3.4. Experience Design Agent

The Experience Design Agent transforms the product idea into real and usable design artifacts.

First, it identifies what screens are needed for this feature, and based on that, it fetches information from the organization's design repository and creates those components. One important aspect is that the agent does not create any new components. The agent only creates new components when it is explicitly authorized to do so.

Instead of producing static images, the agent generates structured wireframes in the form of JSON-based screen graphs. The designs can be brought to life in tools like Figma or viewed as HTML previews for easy review. This, in turn, enables downstream Coding Agents to use the artifacts directly, sparing the team from manual interpretation.

The agent also creates detailed interaction specifications and includes WCAG 2.2 accessibility annotations, so accessibility is taken into account from the very beginning. The final outcome of this stage is the Design Artifact Bundle (DAB).

3.5. Backlog Synthesis Agent

The Backlog Synthesis Agent is the final agent in Layer 1. Its role is to take the Normalized Epic Object, the Discovery Brief, and the Design Artifact Bundle and shape them into a backlog that the engineering team can start working on. Unlike the Planner Agent from our earlier framework, this agent benefits from a deeper understanding of both product and design context, which allows it to create user stories that are more meaningful and practical.

The agent then breaks down the epic into INVEST-compliant user stories and prepares Gherkin-style acceptance criteria. It also links the relevant non-functional requirements (NFRs) and suggests initial test scenarios. In addition, the agent provides effort estimates based on historical data. These estimates are meant to guide the team rather than act as firm commitments.

Through its integration with Jira, the agent creates all the necessary backlog items and links them to the PRB and DAB. Once this step is completed, control is handed back to the Central Orchestrator so that the next phase of the lifecycle can begin.

3.6. Human-in-the-Loop Checkpoints

In any autonomous SDLC, human judgement is very essential and that’s the reason, we have kept three optional and one mandatory checkpoint in this flow. Optional checkpoints include epic clarification, persona approval, and design review, each triggered under specific conditions such as high ambiguity or low design confidence. The mandatory checkpoint is

the backlog sign-off, where the product owner reviews and approves the generated backlog before the lifecycle proceeds to the architecture phase.

3.7. Knowledge Layer and Guardrails Interaction

All agents within Layer 1 interact continuously with the Knowledge Layer and the Guardrails. They retrieve information such as personas, design tokens, prior epics, APIs, telemetry schemas, compliance policies, and incident histories. Any new artifacts generated—such as personas, screen graphs, or requirement mappings—are written back to the Knowledge Layer with full provenance.

Before any read or write operation, the Guardrails ensure compliance and security. The Compliance Agent verifies adherence to data privacy regulations, while the Prompt Safety component detects potential prompt injection attempts. Additionally, the Audit Trail records all actions, including model versions and checkpoint outcomes, ensuring transparency and trustworthiness. By the time the lifecycle advances to the architecture phase, downstream agents can confidently rely on the PRB and DAB.

4. Reference architecture diagram

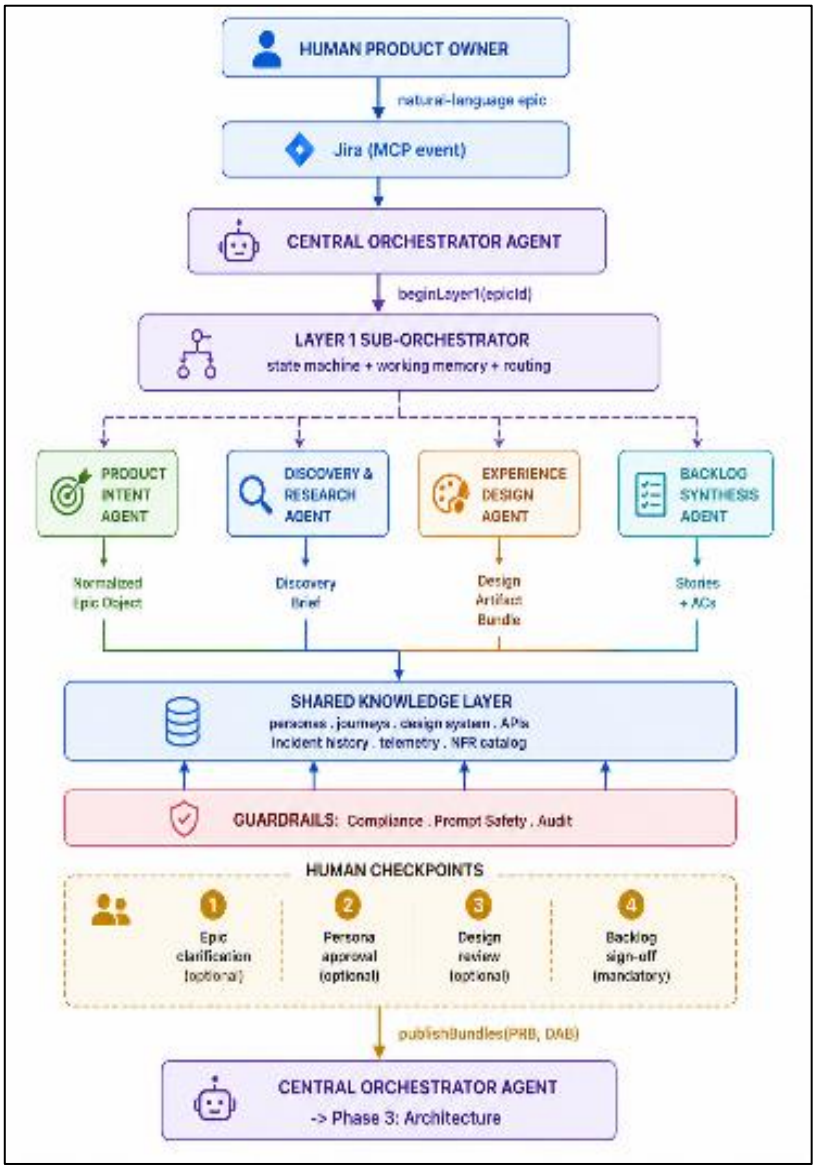

Figure 1 Layer 1 reference architecture: agents, sub-orchestrator, knowledge and guardrail layers, and human checkpoints

Please check Figure 1 above, which shows the reference architecture of Layer 1 as a simple block diagram. It covers the flow from the human product owner, through the Jira MCP connector, into the Central Orchestrator, and then into the Layer 1 Sub-Orchestrator and its four specialist agents. It also shows how every agent talks to the Shared Knowledge Layer and Guardrails stack, and finally the four human checkpoints before control is handed back to the Central Orchestrator for Phase 3.

4.1. Component Summary

Table 1 gives a quick summary of the four agents of Layer 1 and what each one produces. Table 2 lists the artifacts that flow through the layer, along with who produces and consumes them.

Table 1 Layer 1 agent summary.

Agent	**Powered by**	**Primary output**
Product Intent Agent	Reasoning LLM (e.g. Claude Sonnet)	Normalized Epic Object
Discovery & Research Agent	Reasoning LLM + telemetry MCP	Discovery Brief
Experience Design Agent	Multimodal model + design-token MCP	Design Artifact Bundle
Backlog Synthesis Agent	Reasoning LLM + Jira MCP	Stories, ACs, NFRs

Table 2 Primary artifacts flowing through Layer 1.

Artifact	Produced by	Consumed by
Normalized Epic Object	Product Intent Agent	Discovery, Backlog Synthesis
Discovery Brief	Discovery & Research Agent	Experience Design, Backlog Synthesis
Design Artifact Bundle (DAB)	Experience Design Agent	Backlog Synthesis, Phase 3 Architecture
Product Requirement Bundle (PRB)	Backlog Synthesis Agent	Phase 3 Architecture, Phase 4 Coding
Backlog (stories + ACs + NFRs)	Backlog Synthesis Agent	Jira, downstream phases

5. Example transformation flow

To illustrate the operational workflow of Layer 1, this section demonstrates how a simple notification preference epic is transformed into structured product, design, and backlog artifacts through coordinated agent interactions.

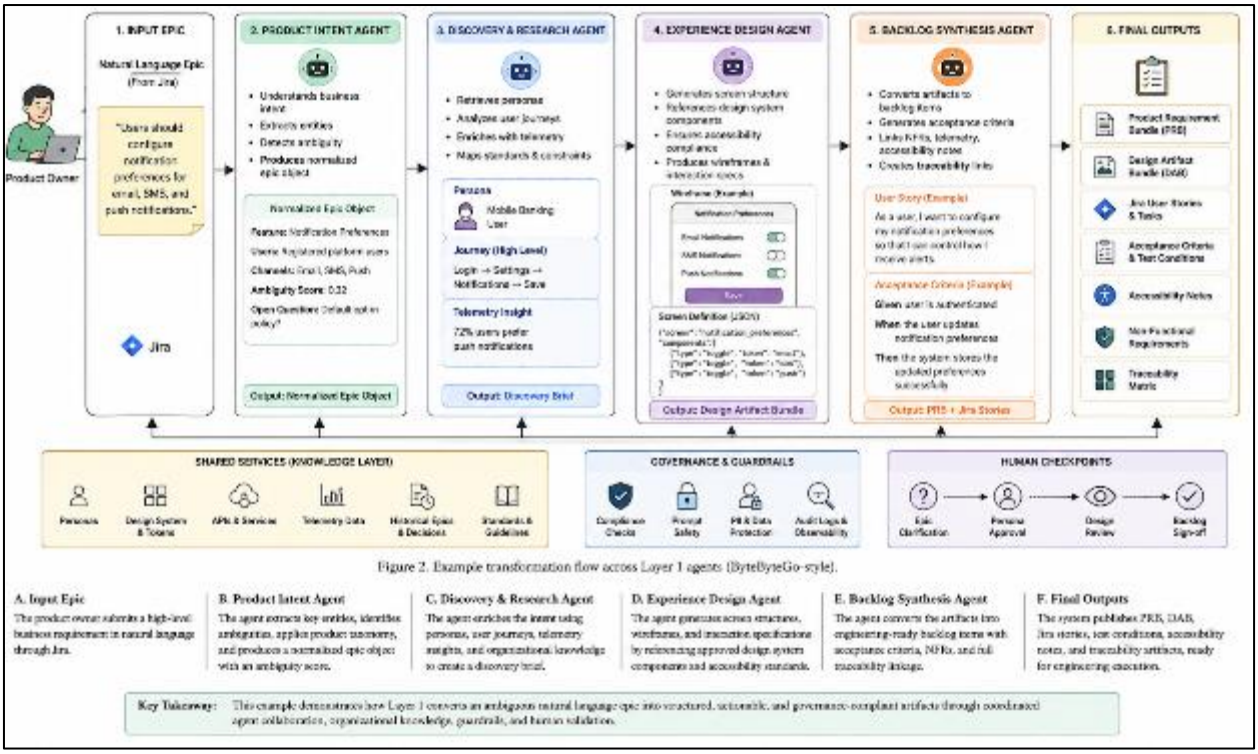

Figure 2. Example transformation flow across Layer 1 agents (ByteByteGo-style).

A. Input Epic
The product owner submits a high-level business requirement in natural language through Jira.

B. Product Intent Agent
The agent extracts key entities, identifies ambiguities, applies product taxonomy, and produces a normalized epic object with an ambiguity score.

C. Discovery & Research Agent
The agent enriches the intent using personas, user journeys, telemetry insights, and organizational knowledge to create a discovery brief.

D. Experience Design Agent
The agent generates screen structures, wireframes, and interaction specifications by referencing approved design system components and accessibility standards.

E. Backlog Synthesis Agent
The agent converts the artifacts into engineering-ready backlog items with acceptance criteria, NFRs, and full traceability linkage.

F. Final Outputs
The system publishes PRB, DAB, Jira stories, test conditions, accessibility notes, and traceability artifacts, ready for engineering execution.

Key Takeaway: This example demonstrates how Layer 1 converts an ambiguous natural language epic into structured, actionable, and governance-compliant artifacts through coordinated agent collaboration, organizational knowledge, guardrails, and human validation.

Figure 2 Layer 1 workflow of the framework

6. Architectural implications and expected benefits

6.1. Time-to-Backlog

In the traditional way of working, moving from an epic to an engineering-ready backlog with approved designs usually takes one to three weeks. Most of the time the developers sit idle and instead of the actual work they spend significant amount of time on waiting for meetings, feedback, and approvals. With Layer 1 operating in an autonomous manner, the same journey can be completed within minutes of wall-clock time, apart from the required human checkpoints. In most cases, the mandatory backlog sign-off remains the only primary time-consuming step. Similar reductions in turnaround time have also been observed in other multi-agent systems.

6.2. Requirement Churn and Design-to-Code Fidelity

If system can address all the ambiguity with the help of agents, developer doesn't have to worry about the requirement changes during their sprint. The clarification pack generated by the Product Intent Agent encourages the product owner to make decisions while the context is still fresh.

In addition, the Experience Design Agent prepares wireframes in a structured way so that the Coding Agent from Phase 4 can use them directly. This means developers no longer need to translate designs manually, reducing the common gap between design and engineering and minimizing UI mismatches. Earlier research also indicates that refining AI-generated outputs through automated feedback improves their accuracy and stability.

6.3. Accessibility and Compliance by Construction

Another important advantage of this layer is that accessibility and compliance checks are embedded from the very beginning rather than being treated as an afterthought. By the time a user story reaches the Coding Agent, it already includes accessibility annotations, ensuring that the resulting code naturally adheres to these standards instead of being retrofitted later.

Compliance validation is handled by the same Compliance Agent introduced in the parent framework, which is extended in this layer to include design-specific policies. Taking this action early really helps organizations meet regulatory and accessibility requirements more consistently.

6.4. Preserving Designer Authorship

Our approach is not going to reduce the role of human designers and their creativity. The framework is designed in such a way that the Experience Design Agent doesn't create a new component on its own or it doesn't have capability to override design system. If the agent detects any high-visibility screen or if the confidence is low then it will send those to human designer for the review.

Designers are not losing their place in the process; instead, their role is evolving. Rather than manually creating every single screen, they can focus on curating the design system, approving new components, and reviewing complex or edge-case scenarios. In our view, this shift allows designers to contribute in a more strategic and high-value way compared to the traditional approach.

6.5. Knowledge Accumulation and Reuse

Accumulating knowledge in a compounding fashion is the most astonishing part of this framework. Each time a feature is being developed, all the agents will connect to the Central Knowledge Layer to get the organizational context, coding standards, experience, any known issues, and architecture. When the feature is developed, all the new experiences and information will be fed back to the Knowledge Layer. Gradually, the Knowledge Layer will become robust with all the information, and the agents will become more effective in producing higher-quality outputs with fewer iterations, as they are going to get more sophisticated and mature knowledge from the Knowledge Layer.

This kind of organizational learning is something that stateless AI coding assistants are unable to offer, making the framework increasingly valuable as it matures.

6.6. Limitations

So far, we've talked about all the advantages of this framework, but it is also important to recognize its limitations.

Let's begin our discussion with the probabilistic nature of Large Language Models. It means that Layer 1 inherits this probabilistic nature, and there is always a chance of getting error-prone outputs.

For example, the Product Intent Agent might misinterpret an epic, or the Experience Agent might select an inappropriate component, etc. Although these risks are reduced through confidence scoring, guardrails, and human checkpoints, they cannot be completely eliminated.

Another important consideration is the availability of a mature, machine-readable design system. If such a system is not in place, the Experience Design Agent may end up creating too many new components, leading to inconsistencies. In such cases, organizations would benefit from strengthening their design systems while adopting this framework.

Cost is also a practical concern. Multimodal design generation is typically the most expensive part of this layer on a per-token basis. To keep these expenses under control, organizations can adopt strategies such as caching and reusing design artifacts wherever possible.

7. Conclusion

In this paper, we explored what the first layer of an AI-First SDLC could look like in a real-world setting. Layer 1 acts as the entry point of the entire lifecycle, where a simple business idea is gradually shaped into structured artifacts that guide all the downstream phases. Because of this, we believe that agentic automation can create the most value at this stage, even though it is also the most sensitive part of the process.

To address this, we proposed a four-agent architecture consisting of the Product Intent Agent, Discovery and Research Agent, Experience Design Agent, and Backlog Synthesis Agent. These agents are coordinated by a Layer 1 Sub-Orchestrator and supported by the shared Knowledge Layer and governance guardrails from the parent framework.

Finally, our aim is not just to replace humans by introducing an autonomous software delivery process. But in our legacy human-driven system, we waste most of our time in repetitive and often frustrating back-and-forth discussions between business, design, and engineering teams. This system will reduce those unnecessary discussions so that everyone can invest their time in more innovative tasks. This constant translation is what makes the early stages of many projects slow and difficult. We believe that Layer 1 is where this friction is the highest and where an agent-driven approach can

make the biggest difference, while still keeping the product owner and design lead in control of the decisions that truly matter.

Compliance with ethical standards

Acknowledgments

The authors would like to thank the engineering, product, and design communities whose published practices around agile delivery, design systems, and multi-agent collaboration have helped shape this work. We would also like to thank the reviewers of our earlier framework paper *(5)*, whose feedback directly motivated this deeper look into Layer 1.

Disclosure of conflict of interest

The authors declare no competing interests

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Author contributions

A. N. Saha and D. Patra jointly conceived the Layer 1 architecture, defined the responsibilities of each agent, and drafted and revised the manuscript. Both authors approved the final version.

Data and materials availability

This paper introduces an architectural framework and does not report empirical data. All referenced public tools, specifications, and prior work are cited in References and Notes.

References

[1] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. Agile Alliance.

[2] Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley.

[3] Endres, C., Abdelnabi, S., Greshake, K., Holz, T., Fritz, M., & Mishra, S. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security* (pp. 79–90). Association for Computing Machinery. https://doi.org/10.1145/3605764.3623985

[4] Garrett, J. J. (2010). *The elements of user experience: User-centered design for the Web and beyond* (2nd ed.). New Riders.

[5] Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yao, S. K. S., Wu, Z., & Ding, L. (2023). *MetaGPT: Meta programming for a multi-agent collaborative framework*. arXiv. https://doi.org/10.48550/arXiv.2308.00352

[6] Kang, S., Yoon, J., & Yoo, S. (2023). Large language models are few-shot testers: Exploring LLM-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)* (pp. 2312–2323). IEEE. https://doi.org/10.1109/ICSE48619.2023.00194

[7] Kim, G., Humble, J., Debois, P., & Willis, J. (2021). *The DevOps Handbook* (2nd ed.). IT Revolution Press.

[8] Nielsen, J. (1994). *Usability engineering*. Morgan Kaufmann.

[9] Norman, D. (2013). *The design of everyday things* (Revised ed.). Basic Books.

[10] OpenAI. (2023). *GPT-4 technical report*. arXiv. https://doi.org/10.48550/arXiv.2303.08774

[11] Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)* (pp. 15097–15118). Association for Computational Linguistics. https://doi.org/10.48550/arXiv.2307.07924

[12] Saha, A. N., & Patra, D. (2026). AI-first software development lifecycle: An agent-driven framework for autonomous planning, coding, testing, and deployment. *ESP Journal of Engineering Technology Advances, 6*, 131–139.

[13] Debashis Patra, Ambar Nath Saha , 2026. "Preparing the Enterprise for AI-Driven Software Development: A Readiness Framework for Organizational Transformation", *ESP Journal of Engineering & Technology Advancements* 6(2): 30-36.

[14] Sauro, J., & Lewis, J. R. (2016). *Quantifying the user experience* (2nd ed.). Morgan Kaufmann.

[15] Vaithilingam, P., Zhang, T., & Glassman, E. (2022). 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 (CHI '22)* (pp. 1–7). Association for Computing Machinery. https://doi.org/10.1145/3491101.3519665

[16] Wooldridge, M. (2009). *An introduction to multiagent systems* (2nd ed.). Wiley.

[17] Yang, J., Pate, K., Sheng, A. S., Katz, D. M., Celi, L. A., Cliff, R. J. M., & Ebright, R. Y. (2024). *SWE-agent: Agent-computer interfaces enable automated software engineering*. arXiv. https://doi.org/10.48550/arXiv.2405.15793