International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 02 | Feb - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

An Agentic, Workflow-Native System for Generating Structured Accessibility Annotations in Design Tools

Nilmani Kumar[1]

[1]*Thoughtworks*

neil.mani@thoughtworks.com

Abstract - Accessibility defects in modern software development frequently originate during the visual design stage, where dynamic interactions, focus management, and assistive technology expectations are insufficiently specified. Existing accessibility plugins within design environments predominantly function as static linters (e.g., color contrast checkers) or manual documentation kits, leaving critical behavioral accessibility guidance deferred to the engineering phase. This paper introduces a workflow-native, agentic system embedded within Figma and synchronized with developer workspaces (e.g., VS Code) that autonomously generates structured, screen-specific accessibility annotation reports. Guided by strict heuristic and WAI-ARIA Authoring Practices, the system utilizes a multi-model Large Language Model (LLM) engine to parse design artifacts and output deterministic Markdown documentation detailing keyboard interaction sequences, focus order, and screen reader announcements. Through an expert agreement analysis across 24 enterprise screens and a downstream quality assurance (QA) implementation study, we demonstrate that the system achieves substantial expert alignment ($\kappa = 0.76$) and reduces downstream accessibility defects by 32%. We argue that embedding AI-driven, documentation-first accessibility agents directly into the design-to-development pipeline effectively mitigates structural gaps in accessible product engineering.

Key Words: Web Accessibility, Large Language Models, Human-Computer Interaction, Design Workflows, Assistive Technology, WAI-ARIA.

1.INTRODUCTION

The operationalization of accessibility in enterprise software development remains highly inconsistent. While the Web Content Accessibility Guidelines (WCAG) provide universal compliance criteria, translating these normative guidelines into actionable, component-level specifications during the design phase is often an informal, high-friction process. Consequently, complex accessibility decisions—such as Document Object Model (DOM) focus trapping, dynamic ARIA state transitions, and contextual screen reader announcements—are frequently left to software engineers to interpret during active development.

This deferred specification inevitably leads to incomplete focus order documentation, ambiguous expectations for assistive technologies, and a heavy reliance on Quality Assurance (QA) remediation cycles. While the software engineering industry has heavily promoted "shifting left" (moving testing and specification earlier in the lifecycle), this has seen limited success regarding behavioral accessibility. Modern design tools feature various accessibility plugins, but these largely act as static evaluators, flagging contrast ratios or touch target dimensions rather than generating structured behavioral guidance for interactive User Interfaces (UI).

To address this systemic gap, this study investigates the following research question: *Can a prompt-governed, agentic system embedded within the design workflow accurately generate structured accessibility annotations that align with expert expectations and measurably reduce implementation defects?*

In this paper, we present a novel accessibility annotation agent integrated via a conversational interface within Figma (e.g., Figma Make). The system leverages a multi-model LLM architecture to parse structural design

topologies and produce deterministic Markdown documentation tailored for direct developer handoff.

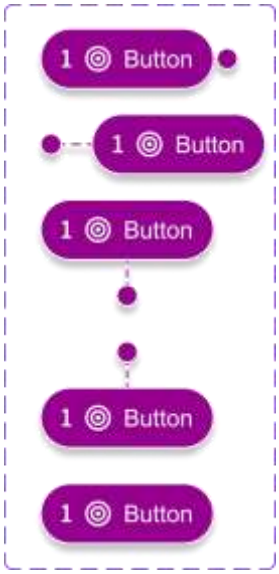
2. Related work

2.1 Automated Accessibility Evaluation

Standard automated tools (such as axe-core) operate on the rendered HTML DOM and the browser's accessibility tree. While highly efficient for identifying static rule violations (e.g., missing alternative text), they are inherently retrospective. They cannot analyze early-stage visual artifacts and are incapable of deducing the intended interactive behavior of a component before implementation.

2.2 Design-Stage Accessibility Tooling

The integration of accessibility into vector graphic editors (e.g., Figma, Sketch) has popularized plugins that simulate visual impairments or validate static constraints. Additionally, manual annotation libraries allow designers to drag and drop badges to indicate focus order. However, these solutions require significant manual labor and deep domain expertise, leading to low adoption for complex, multi-state application flows.

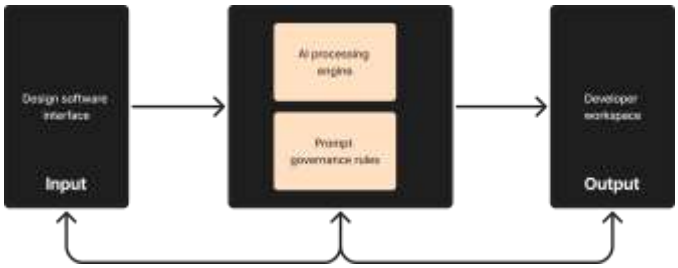


2.3 LLM-Assisted Accessibility

Recent research has explored the application of Large Language Models to accessibility, primarily focusing on interpreting WCAG rules against codebase snippets or automatically generating unit tests. However, these systems still operate on code artifacts. Our system diverges significantly by operating exclusively on design-stage interaction modeling, utilizing strict governance guardrails to output structured behavioral annotations directly into the developer's workspace.

3. System Architecture

The proposed system bridges the gap between the design canvas (Figma) and the engineering environment (VS Code). It operates via a four-layer architecture designed to translate raw vector nodes into actionable, developer-ready Markdown.



3.1 Architecture Overview

Layer 1: Workflow Integration & User Trigger The system is invoked natively within the design environment via a conversational chat UI. A designer selects a specific Figma frame and issues a prompt (e.g., "create annotation"). This prevents context-switching and anchors the agent to a specific visual boundary.

Layer 2: Structural Extraction & Normalization

To ground the LLM and mitigate hallucination, the system parses the underlying Figma node tree using strict heuristics. The system is programmed to assume specific design conventions:

- Top-level frames represent single interactive flows.
- Vertical auto-layout structures dictate the logical reading order.
- Interactive elements are explicitly defined as components, with text labels nested within.
- Interactive states (e.g., collapsed vs. expanded accordions) are represented by component variants.

Layer 3: Multi-Model Intelligence Engine

The extracted semantic tree is processed by a configurable LLM backend. The system interface allows

practitioners to experiment with different foundational models (e.g., Default, Gemini 3 Pro, Gemini 3 Flash, Claude Opus 4.6) to balance efficiency with complex reasoning capabilities. The engine operates under a strict system prompt (Guidelines.md).

Layer 4: Deterministic Output Generation

The final output is synchronized directly into the developer workspace as a Markdown (.md) file. The documentation explicitly details keyboard tab orders, specific ARIA attributes (e.g., aria-disabled="true"), and virtual cursor reading orders.

3.2 Governance and Prompt Engineering (Guidelines.md)

A critical innovation of this architecture is its rigid constraint system. The agent is explicitly instructed to:

1. Prioritize WAI-ARIA Authoring Practices and native HTML semantics.
2. Adopt the mental models of primary screen readers (NVDA, JAWS, VoiceOver).
3. **Refrain** from providing generic accessibility advice or simply listing WCAG criteria.
4. Focus exclusively on concrete interaction sequencing and auditory announcements.

4. Methodology

We evaluated the system through a two-phase empirical study: an expert validation of the generated artifacts, and an analysis of downstream QA metrics during software implementation.

4.1 Dataset and Implementation

We curated 24 complex enterprise application screens. These represented high-risk accessibility vectors, including multi-step application forms, dynamic data tables, financial authorization modals, and complex accordion-based document viewers. All assets were stripped of proprietary identifiers. Using the conversational trigger interface, the agent generated Markdown annotation files for each screen.

4.2 Expert Agreement Study

Three accessibility-specialized User Experience (UX) professionals independently evaluated the agent-generated reports. Each output was assessed using a standardized rubric evaluating:

- **Correctness (0-2):** Alignment with WAI-ARIA standards and native HTML semantics.
- **Completeness (0-2):** Coverage of all interactive elements within the selected frame.
- **Actionability (0-2):** Clarity and precision of the technical guidance for engineers.
- **Hallucination Presence (Yes/No):** The invention of UI elements not present in the source design.

Inter-rater reliability was measured using Cohen's kappa (κ).

4.3 Downstream QA Comparison

To assess practical efficacy, we monitored two agile development cycles utilizing identical engineering teams but contrasting specification methods:

- **Cycle A (Control):** Utilized standard, manual accessibility annotation processes.
- **Cycle B (Experimental):** Utilized the agent-generated structured Markdown annotations.

We measured the volume of accessibility defects flagged during formal QA testing, the frequency of ad-hoc clarification requests submitted to the design team, and the overall duration of accessibility review cycles.

5. Results

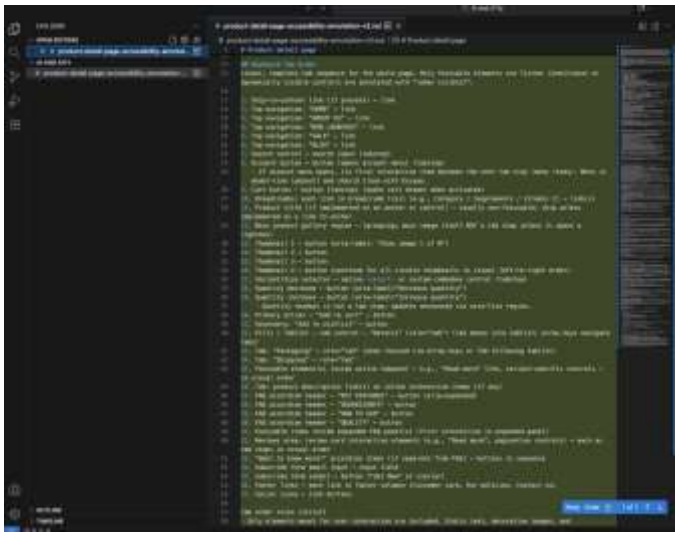
5.1 Artifact Quality and Expert Agreement

The expert evaluation revealed that the structured outputs were highly aligned with professional accessibility expectations.

Evaluation Metric	Mean Score
Correctness	1.82
Completeness	1.75
Actionability	1.88
Hallucination rate	<3%

Cohen's kappa indicated substantial agreement among the expert reviewers (κ = 0.76). The system excelled at defining sequential Tab orders and detailing specific

states, accurately generating pseudo-code guidance such as identifying toggle buttons, noting aria-disabled states, and differentiating between focusable elements and informational badges.



5.2 Downstream Engineering Impact

The introduction of the agentic annotations yielded significant, measurable improvements in engineering velocity and quality during Cycle B.

Table 2: QA and Implementation Impact (Cycle B vs. Cycle A)

Performance Metric	Cycle A (Control)	Cycle B (Control)	Measured improvement
Accessibility QA Defects	47 defects	32 defects	32% reduction
Developer Clarification Requests	39 requests	23 requests	41% reduction
Average A11y QA Review Time	4.5 hours/ ticket	3.2 hours/ ticket	28% faster

Qualitative feedback from the engineering cohort highlighted the utility of the ## Screen Reader Reading Order sections. By providing explicit virtual cursor expectations directly in the VS Code workspace, developers were able to implement robust DOM structures on their first pass, particularly for complex components like data tables and nested accordions.

6. Discussion

The findings of this research strongly advocate that enterprise accessibility failures frequently result from structural under-specification rather than a deficit in engineering capability. By algorithmically generating structured documentation at the visual design phase, the cognitive burden on developers is drastically minimized. Accessibility transitions from an implicit, highly subjective interpretation to an explicit, testable engineering contract.

The success of the agent relies heavily on its prompt governance. By strictly forbidding generic WCAG recitations and forcing adherence to WAI-ARIA authoring patterns, the generated Markdown (as shown in the study artifacts) bypasses abstract theory and delivers immediate technical utility. Furthermore, allowing practitioners to toggle between different LLM models (e.g., Gemini 3 Pro vs. Claude Opus 4.6) provides flexibility, allowing teams to balance token processing

speeds with the deep contextual reasoning required for highly complex interfaces.

It is critical to note that this system is not designed to replace human accessibility expertise. Instead, it serves as an intelligent scaffold. It handles the rote documentation of tab indices and basic semantic roles, allowing human accessibility specialists to focus their efforts on auditing highly complex, bespoke interactions.

7. Limitations

The primary limitation of this system is its inherent reliance on the structural integrity of the source design file. Because the agent heuristic assumes that vertical auto-layout represents reading order and that interactive elements are properly componentized, poorly constructed Figma files (e.g., flattened layers, disorganized groupings) will result in degraded or inaccurate output.

Additionally, while the system effectively specifies intended behavior, it cannot test runtime logic, API latency effects on state, or dynamic data rendering issues. The generated documentation provides a blueprint but does not legally guarantee WCAG compliance post-compilation. Finally, while the 24-screen sample size is robust for heuristic validation, wider cross-organizational replication is necessary to generalize the impact metrics.

8. Implications for HCI and Accessibility

This research introduces a paradigm shift in accessibility tooling: the advent of *Documentation-First Accessibility Agents*. Historically, accessibility software has focused on post-hoc validation. By operationalizing accessibility specification upstream, we reframe accessibility not as a compliance hurdle appended to the end of development, but as a core, structured design artifact.

Future HCI research should investigate how these deterministic Markdown artifacts can be automatically ingested by End-to-End (E2E) testing frameworks (e.g., Cypress or Playwright) to enable fully automated, test-driven accessible development pipelines.

9. Conclusion

This paper presented an agentic, workflow-native accessibility annotation system embedded within the design-to-development pipeline. By leveraging a multi-model LLM engine constrained by strict heuristic extraction and WAI-ARIA governance rules, the system autonomously generates highly accurate, developer-ready Markdown documentation. Through expert analysis and downstream implementation metrics, we demonstrated that this approach significantly aligns with expert judgment and yields a 32% reduction in post-development accessibility defects. The results validate that AI agents, when properly constrained and integrated into native workflows, can profoundly enhance accessible software engineering practices without entirely displacing human oversight.

References

1. World Wide Web Consortium (W3C). (2018). *Web Content Accessibility Guidelines (WCAG) 2.1*.
2. W3C Web Accessibility Initiative (WAI). (2023). *ARIA Authoring Practices Guide (APG)*.
3. Vigo, M., Brown, J., & Conway, V. (2013). Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests. *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility (W4A)*.
4. Brajnik, G. (2008). Beyond conformance: The role of accessibility evaluation methods. *International Conference on Web Information Systems Engineering*.
5. Wang, Z., et al. (2023). Enabling Conversational Interaction with Mobile UI using Large Language Models. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.
6. Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1), 37-46.
7. Ross, A. S., et al. (2020). Examining the shift-left approach to accessibility in agile software development. *ACM Transactions on Accessible Computing (TACCESS)*, 13(3), 1-28.
8. Hou, X., et al. (2023). Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology*.
9. Morris, M. R., et al. (2024). The promise and peril of Generative AI for accessibility. *Communications of the ACM*, 67(2), 44-53.
10. Bigham, J. P., et al. (2017). Human-Computer Interaction and Accessibility. *Foundations and Trends in Human-Computer Interaction*, 10(4), 281-366.
11. Deque Systems. (2023). *axe-core: Accessibility engine for automated Web UI testing*.
12. Feng, J., & Wang, Y. (2024). Auto-generating ARIA attributes for UI components using prompt

engineering. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*.

13. Lunn, D., et al. (2022). Closing the designer-developer gap: The impact of explicit behavioral documentation on UI implementation. *Journal of Systems and Software*, 191, 111354.

14. Yesilada, Y., et al. (2015). Accessibility in the wild: how are people evaluating web accessibility? *World Wide Web*, 18(4), 1011-1027.

15. Alshayban, A., et al. (2020). Accessibility issues in Android apps: state of the art, challenges, and future directions. *Empirical Software Engineering*, 25(6), 4645-4684.