Shift-Left Accessibility for UX/UI: A Figma Preflight versus Code-Time Checks (Pilot)

Noelynn Faith Batalingaya

November 5, 2025

1 Introduction

Accessibility in technology is vital because digital tools shape nearly every part of modern life. People use websites and applications to study, work, receive healthcare, shop, and connect with others. When these systems are not accessible, individuals with disabilities are excluded from opportunities that others enjoy; accessibility is not an extra feature but essential for fairness and inclusion.

The lack of accessibility has real consequences. A student who relies on a screen reader may be unable to complete an online exam; a worker who cannot use a mouse may struggle to submit a job application; a person with low vision may miss critical health information if contrast is too low. Studies of real applications continue to find recurring problems, such as missing labels and weak support for assistive technologies [1].

Even with the Web Content Accessibility Guidelines (WCAG), many digital products remain inaccessible, in part because issues begin early in design [2]. Designers may pick colors with poor contrast, omit alternative text, or create layouts that are hard to navigate. Once encoded in a design system, these choices flow downstream into code, and fixes later are slower and costlier. Interviews with practitioners also surface barriers such as time pressure, limited training, and unclear ownership [3].

This project moves checks to the beginning of the process. It applies shift-left testing—testing earlier, when changes are cheaper and faster. Rather than waiting for code-time

tools (e.g., Lighthouse or axe-core) to detect problems, this pilot develops a Figma plugin that performs accessibility "preflight" checks while designs are being created. The plugin highlights problems such as poor color contrast or missing text alternatives and provides suggestions before any coding begins. The central research question is whether early design checks can reduce downstream barriers and help create more inclusive technology.

The following sections introduce key background concepts, describe the planned software deliverable, illustrate the workflow, and situate this project within related research before concluding with future directions.

2 Background

Accessibility aims to make digital products usable by people with a variety of abilities. Some rely on screen readers; others need captions, larger targets, or keyboard navigation. WCAG is often summarized by four principles: perceivable, operable, understandable, and robust [2]. Perceivable means people can see or hear content (e.g., captions and sufficient contrast). Operable means people can use the interface by different inputs such as a keyboard. Understandable means content and interaction are clear and predictable. Robust means content works across devices and assistive technologies.

WCAG also defines three conformance levels. Level A is the minimum. Level AA, a common organizational target, requires stronger contrast, captions, and more consistent navigation. Level AAA is strictest and includes requirements such as sign language for some video and even higher contrast. Because AAA is difficult for all content, many teams aim for AA as a practical standard.

Timing matters. In many workflows, accessibility is checked late, when products are almost finished. Fixes at that stage can require redesigns or code rewrites. Shift-left testing addresses this by catching issues early; it parallels successful practices in performance and security. Figma is well suited for this idea: it is widely used, collaborative, and supports plugins. Research shows that scanning Figma prototypes can reveal common issues early and reduce downstream cost [4, 5]. At the same time, some problems only surface at runtime; recognition and metadata generation still matter in code-time evaluation [6]. Together, this

suggests a workflow that begins in design and is confirmed in code.

Planned Software Deliverable

The main deliverable is a Figma plugin that runs design-time checks on real files. It scans text, shapes, frames, and components against a set of rules and shows results directly on the canvas. The checks cover color contrast, text alternatives for images and icons, focus order (design intent), target size for touch and click elements, and reliance on color alone for meaning. When a check fails, the plugin links the warning to the specific element and suggests a concrete fix—for example, proposing passing color pairs or prompting for an alt description. This creates a fast feedback loop that is both practical and educational.

3 Workflow Overview

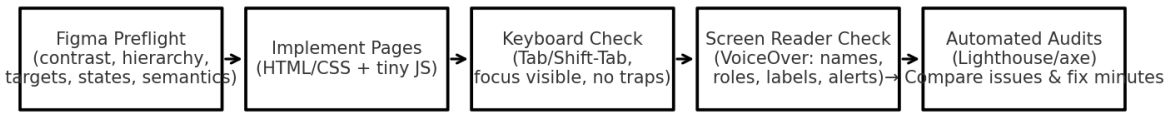

Figure 1: Shift-left accessibility workflow: integrating Figma preflight checks before code-time audits. Each stage—from color contrast and hierarchy checks to automated audits—represents how accessibility testing moves earlier in the design process.

4 Before and After Screenshots

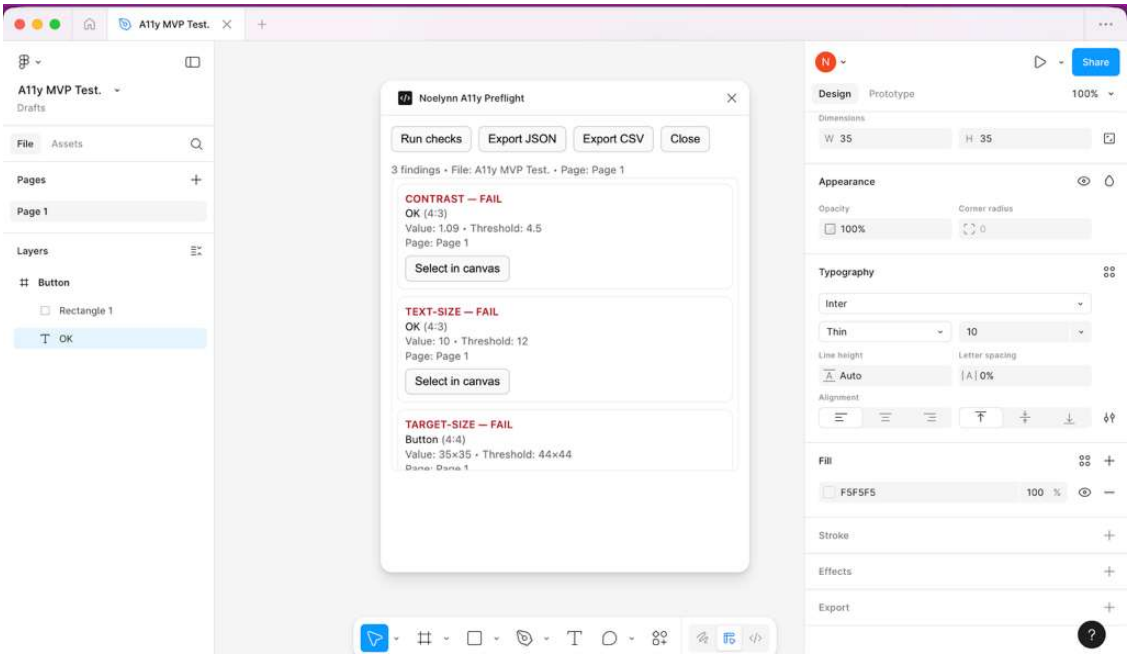

Figure 2: Before: accessibility issues detected by the A11y Preflight plugin (e.g., low contrast, incorrect text-size, wrong target size for the labels).

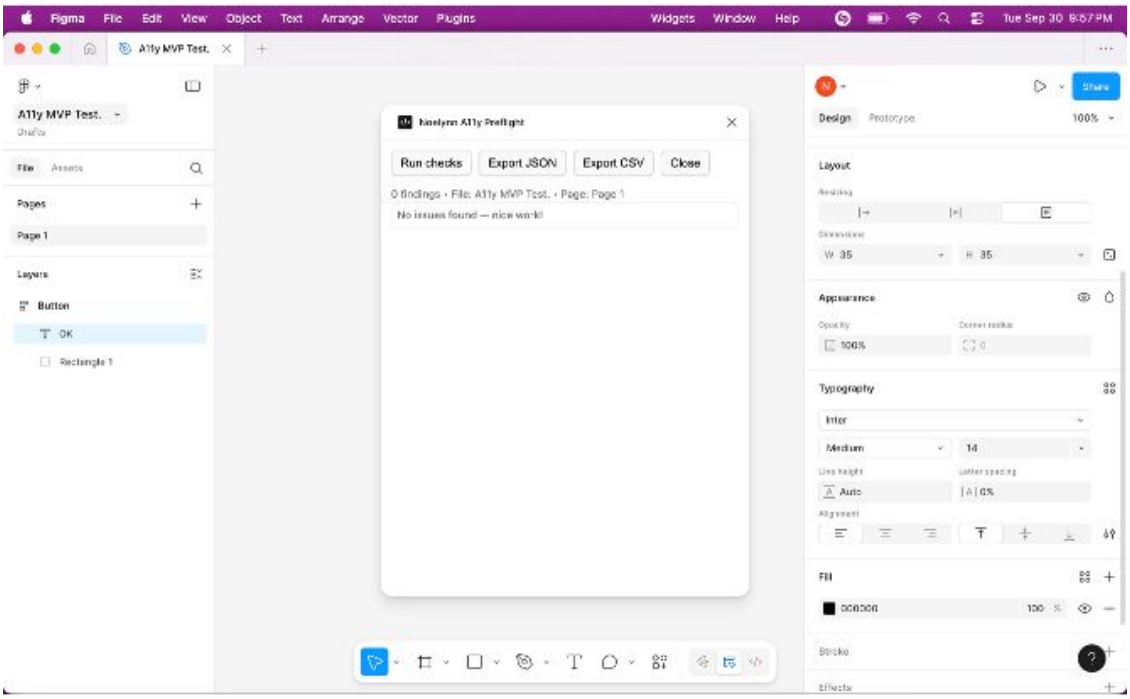

Figure 3: After: design improved following plugin feedback; flagged issues resolved.

5 Related Work

Design-Stage Tools

Most existing accessibility tools focus on code-time evaluation. Frameworks like axe-core, Lighthouse, and WAVE examine finished websites or mobile applications for structural issues such as missing Accessible Rich Internet Applications (ARIA) labels, weak heading structure, or broken landmarks. These tools are powerful but reactive, since they identify issues late. Design-stage tools are fewer and narrower. Stark checks contrast; Able supports adding alternative text. Studies of Figma reveal that static designs can be scanned for low contrast and missing labels, improving quality when checks occur early [4, 5]. Work examining apps built from Figma templates finds that many issues originate in the templates themselves, reinforcing the value of early, design-time review [7].

Organizational and Educational Barriers

Practice studies describe organizational barriers—time pressure, unclear roles, and limited training—that motivate tools embedded directly in design workflows and that explain why issues matter [3]. These findings show that accessibility is not only a technical task but also a cultural one that depends on shared responsibility.

Automation and Artificial Intelligence Approaches

Automated critique with large language models shows promise yet has accuracy limitations and must be applied carefully [8, 9]. Runtime recognition research reminds us that some problems only appear when the interface is live [6]. Overall, prior work supports a combined approach: strong checks during design and confirmation at runtime. Broader design perspectives also argue for empathy and equity, not only rule compliance, as goals of accessible practice [10].

Future Work

Future iterations can expand rules to include motion sensitivity, animation, and device orientation; integrate with code-time tools (e.g., axe-core) for a continuous pipeline; and evaluate the plugin with user studies to measure time saved and defects prevented. Short in-plugin learning tips can reinforce reasoning behind each rule for students and early-career designers.

6 Conclusion

Accessibility should not be left to the end. When teams ignore it early, barriers multiply and exclude real people. This project moves accessibility to the left in the timeline by providing a Figma plugin that detects common problems—poor contrast, missing labels, unclear focus order—while screens are still being shaped. The contribution is both practical and conceptual: a working tool that fits normal design work and evidence that shift-left testing can reduce downstream issues and support more inclusive outcomes.

References

[1] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in android applications: State of affairs, sentiments, and ways forward," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, pp. 1323–1334, IEEE/ACM, 2020.

[2] World Wide Web Consortium (W3C), "Web content accessibility guidelines (wcag) 2.2." `https://www.w3.org/TR/WCAG22/`, 2023. Accessed 2025-10-19.

[3] W. Shi and L. Findlater, "It could be better, it could be worse: Understanding accessibility in ux practice with implications for industry and education," in *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '23)*, ACM, 2023.

[4] J. Huang, W. S. Lasecki, and L. Feng, "Beyond the guidelines: Assessing accessibility in figma prototypes," in *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*, ACM, 2024.

[5] Y. Chen, H. Wu, and R. Wang, "Assessing accessibility levels in mobile applications from figma templates," in *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*, ACM, 2024.

[6] H. Zhang, A. Guo, X. A. Chen, and Y. Li, "Screen recognition: Creating accessibility metadata for mobile applications from pixels," in *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*, ACM, 2021.

[7] J. H. Muniz, D. M. F. Rabelo, and W. Viana, "Assessing accessibility levels in mobile applications developed from figma templates," in *Proceedings of the 17th International Conference on Pervasive Technologies Related to Assistive Environments (PETRA '24)*, pp. 316–321, ACM, 2024.

[8] W. Duan, Y. Zhou, C. Wu, and X. A. Cao, "Generating automatic feedback on ui mockups with large language models," in *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*, ACM, 2024.

[9] W. Duan, Y. Zhou, C. Wu, and X. A. Cao, "Uicrit: A ui critic agent and critique dataset," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '24)*, ACM, 2024.

[10] R. Ginosar, H. Kloper, and A. Zoran, "Parametric habitat: Virtual catalog of design prototypes," in *Proceedings of the 2018 Designing Interactive Systems Conference (DIS '18)*, (New York, NY, USA), pp. 1121–1133, ACM, 2018.