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

Noelynn Faith Batalingaya

October 15, 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 just an additional feature; it is essential for fairness and inclusion.

The lack of accessibility has real-world 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 not be able to read important health information if the text contrast is too low. These examples show that inaccessible design blocks access to education, employment, and essential services. Studies of real applications continue to find common problems, such as missing labels and weak support for assistive technologies [1].

Even with the Web Content Accessibility Guidelines, many digital products remain inaccessible. A major reason is that many issues start early in the design process. Designers may pick colors with poor contrast, forget alternative text for images, or create layouts that are hard to navigate. Once these choices are built into design systems, they carry over into code, and fixing them later becomes slow and expensive. Interviews with practitioners also show barriers such as time pressure, limited training, and unclear ownership [8].

This project uses a different approach by moving checks to the beginning of the process. It applies shift-left testing, which means testing earlier in the workflow when changes are cheaper and faster. Instead of waiting for code-time tools like Lighthouse or axe-core to find problems, this study develops a Figma plugin that performs accessibility "preflight" checks while designs are still 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: can early design checks reduce accessibility barriers later and help create more inclusive technology?

2 Background

Accessibility means making digital products usable by people with a variety of abilities. Some rely on screen readers, others on captions, larger buttons, or voice commands. Accessibility ensures that technology supports different ways of interacting. The Web Content Accessibility Guidelines are often summarized by four ideas: perceivable, operable, understandable, and robust.

Perceivable means people can see or hear the content, for example through 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 the content works across devices and assistive technologies.

The guidelines also define three levels of compliance. Level A is the minimum. Level AA is the most common organizational target and requires stronger contrast, captions, and more consistent navigation. Level AAA is the strictest and adds advanced requirements such as sign language for some video and even higher contrast. Level AAA is ideal but hard for every product to reach, so many teams aim for Level AA as a practical standard.

Timing matters. Many teams check accessibility near the end, when products are almost finished. Fixing problems at that stage means rewriting code or redesigning screens. Shift-left testing solves this by catching issues early. It already works well in areas like performance and security. If a designer chooses accessible colors and labels from the start, developers do not need to redo the work later.

Figma is a good platform for this idea. It is widely used, supports real-time collaboration, and has a plugin system. Some design-time tools exist already. For example, Stark checks color contrast and Able helps with text alternatives. Recent research shows that scanning Figma prototypes can reveal common issues early and reduce downstream cost [6, 2]. At the same time, some problems only appear when an application is running; runtime recognition and metadata generation still matter in code-time evaluation [9]. Together, this supports a workflow that starts early in design and then confirms at runtime.

2.1 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 on the canvas.

The plugin checks:

- **Color contrast:** Ensure text and background colors meet common thresholds.
- **Text alternatives:** Flag images and icons that do not have helpful descriptions.
- **Focus order (design intent):** Encourage a logical order for interactive elements in the layout.
- **Target size:** Highlight buttons or touch areas that are too small to tap comfortably.
- **Use of color:** Warn when color alone conveys meaning without text or symbols.

When the plugin runs, it lists issues and links each warning to the exact element. Designers can click, fix the issue, and re-run the checks. If text fails a contrast test, the plugin suggests color pairs that pass. If an icon lacks a text alternative, it prompts the designer to add one. This creates a feedback loop that is both practical and educational.

3 Related Work

Most tools today focus on code-time checks. Axe-core, Lighthouse, and WAVE scan finished websites or apps and report problems such as missing labels, weak heading structure, or broken landmarks. These tools are powerful but reactive, since they find issues late.

Design-stage tools exist but are narrower. Stark checks contrast; Able supports adding text alternatives. Research on Figma shows that static designs can be scanned for issues like low contrast and missing labels, which improves quality when done early [6, 2]. Other work examines apps built from Figma templates and finds that many issues come from templates themselves, which supports early, design-time review [7].

Practice studies show why this matters. Interviews with user experience professionals describe time pressure, unclear roles, and limited training as barriers to accessibility integration [8]. These findings motivate tools that fit directly into design workflows and provide explanations, not only error flags. In parallel, automated critique with large language models shows promise but still has limits and requires careful prompting [3, 4]. Runtime recognition research reminds us that some issues only appear when the interface is live [9]. Overall, prior work supports a combined approach: strong checks during design and confirmation at runtime.

Finally, broader design perspectives argue that accessibility should reflect empathy and equity, not only rule compliance. Tools that explain *why* an issue matters help designers build lasting habits and deliver more inclusive results [5].

3.1 Future Work

This pilot prepares the ground for a larger study. Next steps include expanding rules to cover motion sensitivity, animation, and device orientation; running user studies to measure time saved and issues prevented; and integrating with code-time tools (for example, axe-core) to build a continuous pipeline from design to development. The plugin can also include short learning tips that explain the reason behind each rule, supporting students and early-career designers. These steps will provide stronger evidence that shift-left accessibility reduces problems and improves inclusion.

4 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 using a Figma plugin to find issues during design. The tool highlights common problems like poor contrast, missing labels, and unclear focus order while screens are still being shaped.

The contribution is both practical and conceptual. It delivers a working tool that fits normal design work and shows how shift-left testing can help accessibility. It also reinforces a simple idea: inclusive design is good design. This junior project sets a path for a senior independent study that can extend the plugin, run studies with design teams, and collect data on impact.

References

[1] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. *Accessibility Issues in Android Applications: State of Affairs, Sentiments, and Ways Forward.* In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, IEEE/ACM, 2020.

[2] Yufei Chen, Han Wu, and Ruijie Wang. *Assessing Accessibility Levels in Mobile Applications from Figma Templates.* In *CHI Conference on Human Factors in Computing Systems (CHI '24)*, ACM, 2024.

[3] Wentao Duan, Yiheng Zhou, Chen Wu, and Xiang Anthony Cao. *Generating Automatic Feedback on UI Mockups with Large Language Models.* In *CHI Conference on Human Factors in Computing Systems (CHI '24)*, ACM, 2024.

[4] Wentao Duan, Yiheng Zhou, Chen Wu, and Xiang Anthony Cao. *UICrit: A UI Critic Agent*

and Critique Dataset. In *ACM Symposium on User Interface Software and Technology (UIST '24)*, ACM, 2024.

[5] Rony Ginosar, Hila Kloper, and Amit Zoran. *Parametric Habitat: Virtual Catalog of Design Prototypes.* In *Designing Interactive Systems Conference (DIS '18)*, ACM, 2018.

[6] Jingyi Huang, Walter S. Lasecki, and Liangjie Feng. *Beyond the Guidelines: Assessing Accessibility in Figma Prototypes.* In *CHI Conference on Human Factors in Computing Systems (CHI '24)*, ACM, 2024.

[7] Júlia Holanda Muniz, Daniel Mesquita Feijó Rabelo, and Windson Viana. *Assessing Accessibility Levels in Mobile Applications Developed from Figma Templates.* In *International Conference on Pervasive Technologies Related to Assistive Environments (PETRA '24)*, ACM, 2024.

[8] Wei Shi and Leah Findlater. *"It Could Be Better, It Could Be Worse": Understanding Accessibility in UX Practice with Implications for Industry and Education.* In *CHI Conference on Human Factors in Computing Systems (CHI '23)*, ACM, 2023.

[9] Hanxiao Zhang, Anhong Guo, Xiang Anthony Chen, and Yang Li. *Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels.* In *CHI Conference on Human Factors in Computing Systems (CHI)*, ACM, 2021.