Making Accessibility Accessible, Part I: A Care Framework for Developer-Centered Tooling Infrastructure

Lalitha A R
24f2006078@ds.study.iitm.ac.in

December 19, 2025

Abstract

Web accessibility guidelines provide clear compliance criteria, yet the majority of websites remain inaccessible. We argue this persistence reflects not developer indifference but a fundamental mismatch between accessibility tooling and developer contexts. Current tools optimize for enterprise environments with dedicated accessibility teams, legal departments, and substantial resources. This leaves the majority of web developers–individuals, students, nonprofits, and small teams building 80% of the web–without practical pathways to implementation. We introduce the **Care Framework**, which reframes accessibility tooling design around three principles: (1) *accessibility of accessibility*–tools must be adoptable by developers with limited time and expertise; (2) *context-awareness*–tools must adapt to varied resource constraints rather than assuming enterprise capacity; and (3) *empowerment over enforcement*–tools must guide and educate rather than merely audit and block. Through systematic ecosystem analysis and a case study demonstrating 5.7× adoption growth after developer-centered redesign, we show that accessibility tooling can and must prioritize developer experience as integral to accessibility outcomes. We identify critical boundaries between safe and unsafe automation, propose a composable utility infrastructure model, and establish research directions for building care-centered accessibility tools that serve all web creators, not only the well-resourced few.

1 Introduction

Twenty-six years after the Web Content Accessibility Guidelines (WCAG) were first introduced, accessibility remains an aspiration rather than reality for most of the web. The WebAIM Million report consistently finds that over 94% of home pages contain detectable WCAG failures [**?**]. This persistence occurs despite widespread awareness of accessibility's importance, legal mandates in many jurisdictions, and extensive documentation of implementation requirements.

The standard explanation attributes this gap to developer negligence or organizational deprioritization. We argue this framing obscures a fundamental systemic problem: **the tools and resources for implementing accessibility are themselves inaccessible to most web developers**. Current accessibility tooling implicitly assumes enterprise contexts–organizations with dedicated accessibility specialists, legal teams to interpret compliance requirements, substantial engineering resources to implement fixes, and budgets for testing services. This assumption fails the majority of web creators: individual developers, students learning web development, nonprofit organizations, open-source maintainers, and small teams building side projects, portfolios, and community resources.

Consider the developer experience of implementing a modern web framework versus implementing web accessibility. To build with React, a developer runs a single scaffolding command (`npm init react-app my-app`), follows tutorial documentation designed for progressive learning, and benefits from an extensive ecosystem of components, testing tools, and community resources. The cognitive load is managed through tooling that assumes no prior expertise. In contrast, implementing web accessibility requires reading a 200-page reference manual (WCAG) written for auditors, manually researching implementation approaches across fragmented blog posts and Stack Overflow threads, testing with screen readers or paid services, and iterating without clear feedback loops. The cognitive burden is not reduced by tooling–it is *created* by tooling designed for compliance verification rather than development support.

This mismatch is not theoretical. A developer articulating this frustration on Hacker News wrote: "The problem with accessibility is the need to think about it at all. It doesn't matter how easy it is to implement as the problem first and foremost is the mental and organizational bandwidth it consumes" [2]. This perception of accessibility as an inherent burden is not accidental; it is the direct result of tooling that creates complexity rather than reducing it.

This paper introduces the **Care Framework** for accessibility tooling design. Drawing on care ethics literature [3, 4] and infrastructure studies [5], we argue that moving "beyond compliance" toward more humane accessibility practices requires first making compliance itself accessible to achieve. Care, in this context, means attending to the lived constraints and contexts of developers rather than imposing abstract ideals that assume unlimited resources. It means building infrastructure–shared, maintained, public

utilities–rather than expecting each developer to construct their own solutions or each organization to purchase proprietary services.

Critically, prioritizing developer experience is not opposed to centering disabled users–it is a prerequisite to it. **Inaccessible tooling produces inaccessible websites**. Tools that prove unusable by most developers result in inaccessible web content, harming disabled users regardless of the tools' theoretical capabilities. Moreover, many disabled people are themselves developers [8]; inaccessible development tooling excludes them from participating in web creation. Caring for developers is caring for users.

Contributions

This work makes four contributions:

1. **The Care Framework**: A conceptual framework grounded in care ethics for evaluating and designing accessibility tooling, comprising three principles with explicit boundaries for safe automation.
2. **Ecosystem Analysis**: A systematic examination of current accessibility tooling revealing how design choices reflect implicit resource assumptions that exclude most developers.
3. **Empirical Validation**: A case study demonstrating 5.7× adoption growth (140 to 800+ monthly downloads) after redesigning a tool around care principles, providing evidence that developer-centered design measurably improves accessibility tool adoption.
4. **Research Agenda**: Identification of critical open problems including automation safety boundaries, composable utility design patterns, and sustainable infrastructure maintenance models.

2 Background and Motivation

2.1 The Compliance Paradigm

Web Content Accessibility Guidelines (WCAG) establish technical requirements for accessible digital content [6]. WCAG 2.1 comprises 78 success criteria organized under four principles (Perceivable, Operable, Understandable, Robust), with conformance levels ranging from A (minimum) to AAA (enhanced). Many jurisdictions mandate WCAG compliance for public sector websites and, increasingly, private sector services.

This compliance paradigm has been remarkably successful at establishing *what* constitutes accessibility. WCAG provides clear, testable criteria that enable consistent

evaluation across contexts. However, compliance documentation is written for *auditors*–specialists who evaluate existing implementations–rather than *builders* implementing accessibility from the start. The guidelines specify required outcomes (e.g., "contrast ratio of at least 4.5:1") but provide minimal guidance on implementation pathways, tool selection, or integration into development workflows.

The result is a substantial *knowing-doing gap* [7]. Developers may understand that contrast ratios matter but lack practical knowledge of how to achieve compliant colors while preserving brand aesthetics. They may recognize the importance of keyboard navigation but not know which ARIA patterns to apply or how to test them without extensive screen reader training. Compliance criteria establish the destination but provide no map for the journey.

2.2 The Resource Asymmetry Problem

Web development contexts exist on a spectrum of resource availability. At one extreme, large technology companies and enterprises maintain dedicated accessibility teams with specialists in screen reader testing, WCAG interpretation, automated testing infrastructure, and legal compliance. These organizations can afford proprietary auditing services, consulting engagements, and sustained engineering effort to implement comprehensive accessibility.

At the other extreme, an individual developer building a personal portfolio site, a student creating their first web application, a nonprofit updating their community resource page, or an open-source maintainer stewarding a small library operates under fundamentally different constraints. These developers typically:

- Work alone or in small teams without specialized roles
- Have limited time budgets (accessibility competes with feature development, bug fixes, and maintenance)
- Lack formal training in accessibility or disability studies
- Cannot afford paid testing services or consulting
- Must learn accessibility implementation through self-directed research
- Build on limited budgets or as unpaid labor

Current accessibility tooling optimization reflects the former context while the majority of web development occurs in the latter. This creates what we term **resource asymmetry**: tools designed for resource-rich environments prove impractical or unusable in resource-constrained contexts, yet web accessibility remains equally important regardless of developer resources.

2.3 Developer Experience as Accessibility Barrier

Recent studies have documented specific barriers developers face in implementing accessibility. Trewin et al. [9] identified challenges including lack of awareness, inadequate training, and competing priorities through interviews with web developers at a large technology company. Potluri et al. [8] examined how developers without disabilities (and, importantly, many developers with disabilities) struggle to understand screen reader user needs, finding significant gaps between developer mental models and actual assistive technology usage. Ross et al. [10] analyzed mobile app accessibility epidemiologically, demonstrating that accessibility issues persist across application domains and developer experience levels.

However, much of this literature frames barriers as individual knowledge gaps or organizational culture problems requiring educational interventions. While education is necessary, this framing places responsibility on developers to overcome barriers rather than on tooling systems to not create barriers in the first place. We argue for a more structural analysis: developer experience barriers reflect *design choices in the tooling ecosystem* that assume resources most developers lack.

When accessibility tools assume users can read and interpret 200-page specifications, this is a design choice that excludes time-constrained developers. When tools focus on detecting violations without suggesting fixes, this is a design choice that assumes users have expertise to research solutions. When tools require screen reader proficiency to validate implementations, this is a design choice that assumes users can invest in learning assistive technologies. These choices make sense for enterprise auditing contexts but create insurmountable friction for resource-constrained developers.

The question is not "how do we educate developers to overcome these barriers?" but rather "how do we design tools that do not create these barriers in the first place?" This reframing shifts responsibility from individual developers to the tooling ecosystem–and makes tool design itself an accessibility intervention.

3 The Care Framework

We ground our framework in care ethics, a philosophical approach that emphasizes attention to context, relationships, and particular needs rather than application of abstract universal principles [3, 4]. Care ethics emerged as a critique of justice-based frameworks that assume autonomous, equal agents operating under ideal conditions. Instead, care ethics recognizes that people exist in webs of dependency and constraint, with unequal access to resources, and that moral action requires responding to actual conditions rather than imposing ideal standards.

In accessibility contexts, care ethics has primarily been applied to understanding the experiences of disabled people and critiquing ableist design [11]. We extend this lens

to *developer experience*, arguing that care requires attending to the actual constraints and contexts of those building accessible systems, not merely those using them. This is not to center developer convenience over user needs, but to recognize that **developer experience directly shapes accessibility outcomes**. Tools that prove unusable by most developers result in inaccessible websites. Moreover, many people with disabilities are themselves developers; inaccessible development tooling excludes disabled developers from participating in web creation [8].

3.1 Core Principles

The Care Framework comprises three interrelated principles for accessibility tooling design:

3.1.1 Principle 1: Accessibility of Accessibility

Tools for implementing accessibility must themselves be accessible to developers with varied expertise, time constraints, and learning styles. This principle has two dimensions:

Cognitive Accessibility: Tools should minimize the prerequisite knowledge required to achieve basic compliance. A developer should not need deep expertise in WCAG, assistive technology, or disability studies to make their color choices readable or their forms keyboard-navigable. While deeper expertise enables more sophisticated accessibility, basic compliance should not require specialization.

Adoption Accessibility: Tools should integrate seamlessly into existing development workflows rather than requiring workflow restructuring. If a tool demands developers learn new testing frameworks, change deployment pipelines, or restructure code architecture, adoption friction increases. Care-centered tools meet developers where they are.

This principle does not advocate for "dumbing down" accessibility or hiding its complexity. Rather, it recognizes that *different levels of engagement are appropriate for different contexts*. A developer building a personal blog needs different tooling than a team implementing a government service portal. Current tooling often provides only the latter, leaving the former without practical options.

3.1.2 Principle 2: Context-Awareness

Tools must adapt to varied resource constraints rather than assuming enterprise capacity. In prior work, we demonstrated this principle through mode-based optimization: strict mode for enterprise contexts requiring minimal brand deviation, recursive mode

for general development balancing accessibility and aesthetics, and relaxed mode prioritizing accessibility above all else [12]. This design explicitly acknowledged that different developers operate under different constraints and enabled users to select appropriate tradeoffs for their context.

Context-awareness manifests in tool design through:

Meaningful Customization: Tools should offer choices framed in terms developers understand, not technical parameters requiring domain expertise. Rather than exposing color science metrics like "delta-E thresholds," tools might ask: "Preserve exact brand colors (may fail for some pairs)" versus "Ensure all colors pass (may adjust brand colors slightly)." These options surface the actual tradeoff–brand fidelity versus universal compliance–in language accessible to developers unfamiliar with perceptual color spaces.

Constraint Transparency: When tools cannot accommodate certain constraints, they should communicate this explicitly rather than failing silently or producing low-quality results. A color optimizer that cannot fix certain pairs without significant visual change should explain this limitation and suggest alternatives, not simply return an error.

Progressive Enhancement: Tools should support incremental adoption where developers can address accessibility issues progressively rather than requiring comprehensive implementation at once. This aligns with how developers actually work–shipping iteratively, addressing issues as capacity permits, improving over time.

Context-awareness recognizes that "best practices" in enterprise settings may be impractical or inappropriate in other contexts, and that tool design can either accommodate or foreclose this contextual variation.

3.1.3 Principle 3: Empowerment Over Enforcement

Tools should guide and educate rather than merely audit and block. This principle reflects a pedagogical stance:

Explanation Over Error: When tools identify issues, they should explain *why* something is problematic (what user experience degrades, which WCAG criteria fail) and *how* to address it (suggesting concrete fixes with tradeoff explanations). Error messages that simply state "insufficient contrast" provide no actionable path forward.

Agency and Control: Developers should retain meaningful control over implementation choices rather than having fixes imposed automatically without understanding. This requires tools that surface decisions, explain options, and respect developer judgment about appropriate tradeoffs in their specific context.

Learning Through Use: Tools should be designed to teach accessibility concepts implicitly through use, not require mastery before use. A developer repeatedly seeing

explanations of why certain color combinations fail gradually builds mental models of contrast requirements without formal study.

Empowerment over enforcement does not mean permitting inaccessibility, but rather recognizing that sustainable accessibility practice requires developers to understand *why* certain practices matter and *how* to implement them, not merely follow automated instructions.

3.2 The Automation Boundary Problem

A critical dimension of the Care Framework is establishing **what can and cannot be safely automated**. We explicitly reject universal automation approaches–particularly overlays–that attempt to "fix" inaccessible sites through client-side JavaScript injection. The disability community has documented extensively how overlays harm users, provide false compliance, and remove developer agency [13, 14].

However, bounded automation serving specific, well-defined problems can reduce developer burden without compromising quality. The distinction is crucial:

Safe Automation Domain: Problems with clear technical solutions, minimal context-dependence, and low risk of unintended harm. Example: adjusting color lightness to meet contrast thresholds while preserving hue (brand color identity). The input space is constrained (RGB colors), the objective is measurable (contrast ratio), and the constraint is clear (hue preservation).

Unsafe Automation Domain: Problems requiring semantic understanding, context-dependent judgment, or risking harm through misapplication. Example: generating alternative text for images. Image meaning depends on surrounding context, user intent, and document purpose–information unavailable to automated systems. Automation here produces brittle, often incorrect results that harm screen reader users.

Research-Needed Domain: Problems where automation boundaries are unclear and require empirical investigation. Example: form field labeling. Simple cases (input with type="email" and name="email") safely automate to "Email address." Ambiguous cases (input with name="q") require context (search form vs. quiz form) not available to tools. The 'Research-Needed' category highlights problems where safety is **contingent on unavailable context**. For form field labeling, the ambiguity cannot be resolved without knowing the form's purpose. The boundary is unclear because it depends on the tool's ability to reliably infer or request this missing contextual information. Categorizing a problem here signifies that automation is not *inherently* unsafe but is *currently* unsafe given technological limits and requires empirical investigation into context-capture methods, confidence thresholds, and user-in-the-loop designs.

The Care Framework requires tools to be explicit about these boundaries. Tools should automate what is safe, refuse to automate what is unsafe, and clearly communicate

uncertainty in research-needed domains. This transparency respects both developer agency and user safety.

3.3 Care vs. Compliance

The Care Framework does not oppose compliance. WCAG provides essential, well-researched standards representing decades of expertise from disability communities and accessibility professionals. Compliance is both legally necessary and ethically important as a baseline of access.

However, **compliance is necessary but insufficient for a flourishing accessibility ecosystem**. The Care Framework argues that achieving widespread compliance requires first making compliance implementation accessible. This is not a sequential relationship (first achieve compliance, then pursue care) but a dialectical one: care-centered tooling enables compliance, and compliance standards provide the foundation for care to improve upon.

The distinction is methodological. Compliance asks: "Does this implementation meet standards?" Care asks: "Does our tooling enable developers to meet standards given their actual constraints and contexts?" Both questions are essential; current accessibility discourse emphasizes the former while neglecting the latter.

4 Case Study: Evidence for Care-Centered Design

To validate that care principles measurably improve tool adoption, we examine the development trajectory of CM-Colors, an open-source library for automatic color contrast correction. This case study demonstrates how redesigning around developer needs increased adoption 5.7-fold.

4.1 Initial Design: Enterprise Assumptions

CM-Colors v0.1-0.4 implemented perceptually-aware color optimization in the OKLCH color space [15]. The algorithm successfully achieved minimal perceptual change while meeting WCAG contrast requirements, representing a technical advancement over tools operating in perceptually non-uniform color spaces (RGB, HSL).

However, the tool's design reflected implicit enterprise assumptions:

Expertise Assumptions: Documentation assumed familiarity with color science concepts (OKLCH, delta-E, perceptual uniformity). API parameters exposed technical details (delta-E thresholds, lightness/chroma adjustment strategies) requiring domain knowledge to configure appropriately.

Time Assumptions: The tool provided powerful optimization but required users to understand tradeoffs, read documentation, and experiment with parameters. This assumed users had time to invest in learning the tool.

Resource Assumptions: Documentation examples showed integration into large codebases with testing infrastructure, assuming organizational context.

Adoption plateaued at approximately 140 downloads per month–respectable for a specialized tool but suggesting limited reach beyond accessibility specialists and enterprise contexts with dedicated color system maintenance.

4.2 Redesign: Care-Centered Principles

Version 0.5.0 redesigned CM-Colors around care principles based on user feedback and usage analysis:

Accessibility of Accessibility:

- Simplified API to single function: `pair.make_readable()`
- Removed requirement to understand color spaces–system allows most of the standard color inputs like rgb, hex, hsl, etc.
- Default mode optimized for general use; no configuration required

Context-Awareness:

- Introduced three modes framing tradeoffs in developer-understandable terms [12]:
  - Mode 0 (Strict): "Minimal brand color change, lower success rate"
  - Mode 1 (Default): "Balanced–works for most color pairs"
  - Mode 2 (Relaxed): "Ensure compliance even if colors shift more"
- Mode descriptions explained *when* to use each, not technical implementation

Empowerment Over Enforcement:

- Tool returns both fixed color and success boolean, giving developers information to make decisions
- Documentation added visual examples showing before/after for common cases
- Error messages explained *why* certain pairs were difficult (e.g., "Colors too similar–even maximum adjustment insufficient")

Critically, these changes did not reduce technical capability–the underlying algorithm remained identical. The redesign focused entirely on *how users interact with the tool*. This redesign primarily embodies the principles of *Accessibility of Accessibility* and *Context-Awareness* by reducing cognitive load and framing choices. While the tool's design respects *Empowerment* by returning control and explanatory messages, its domain (color contrast) falls within the 'Safe Automation' boundary. Thus, this case strongly validates the framework's capacity to overcome adoption barriers and points the direction for tools addressing more complex, context-dependent problems where empowerment through guidance is even more critical.

4.3 Impact: 5.7× Adoption Growth

Following the v0.5.0 release, monthly downloads increased from 140 to over 800 within the same monthly period–a 5.7× growth. This surge coincided with the tool's feature in a prominent technical blog, which highlighted its utility for resolving contrast issues with minimal design alteration. Qualitative feedback in project issues shifted from configuration questions ("How do I set the right delta-E threshold?") and requests for documentation clarification to reports of successful usage ("This solved my color contrast problems") and feature requests building on successful adoption.

This case study provides empirical evidence that care-centered design–making tools accessible to developers with varied expertise and constraints–measurably improves adoption. The barrier to accessibility implementation was not lack of technical capability (the algorithm existed) but *inaccessibility of the tool itself*.

5 Barriers to Care: Analyzing Current Tooling

We analyze the current accessibility tooling ecosystem to identify how design choices create barriers for resource-constrained developers.

5.1 Checker Tools: Identification Without Remediation

Automated accessibility checkers (WebAIM's WAVE, Deque's axe, Google Lighthouse) scan websites to identify WCAG violations. These tools serve essential functions in enterprise testing workflows and have raised general accessibility awareness.

5.1.1 Design Assumptions

Checker tools implicitly assume:

- Users possess WCAG expertise to interpret findings
- Users have time and knowledge to research remediation approaches
- Violations are straightforward to fix once identified
- Organizations have dedicated personnel to address findings

5.1.2 Care Gaps

For resource-constrained developers, checkers present several challenges:

Interpretation Burden: An axe report might flag "ARIA attribute is not allowed for the element's role." A specialist recognizes this indicates incorrect ARIA usage and knows which patterns to apply. A developer learning accessibility sees an error message requiring research to understand, let alone fix.

Remediation Guidance Gaps: Most checkers provide links to WCAG documentation but minimal concrete guidance on implementation. A contrast error report states the measured ratio and required threshold but does not suggest which colors to adjust or how to preserve design intent.

False Positive Burden: Automated checkers cannot detect all accessibility issues (research suggests only 30-40% of WCAG criteria are automatable [16]), yet they generate substantial false positives requiring expert judgment to dismiss. Resource-constrained developers may waste time pursuing false leads or, worse, develop **learned helplessness** toward checker output, where they disengage from accessibility efforts entirely due to overwhelming and cryptic feedback.

Scale Mismatch: Enterprise contexts can pipeline checker output to accessibility specialists for triage. Individual developers receive raw reports requiring direct engagement with every finding.

These gaps do not make checkers useless–they remain valuable for enterprises and experienced practitioners. However, they illustrate how tools optimized for verification fail to support implementation.

5.2 Adjuster Tools and Overlays: The Automation Failure

Some tools attempt automatic remediation of accessibility issues. Overlay solutions inject JavaScript to provide accessibility features like text sizing, contrast adjustment, and content simplification. The disability community has extensively documented how overlays harm users through poor implementation, performance degradation, and false compliance [13, 14].

Overlays represent the failure mode of automation without care:

Unsafe Automation: Overlays attempt to fix complex, context-dependent problems (semantic structure, navigation patterns, content meaning) through automated transformations. This violates the automation boundary principle–these problems require human judgment.

Agency Removal: Overlays remove developer agency entirely. Developers do not learn what was wrong or how to fix it; they apply a black-box solution. This precludes learning and creates dependency rather than capability.

User Harm: Poor overlay implementations can break existing accessibility, override user preferences, and create new barriers. The performative compliance they provide protects organizations legally while harming disabled users actually.

The overlay backlash teaches a crucial lesson: automation that seems to help developers ("instant accessibility!") but violates care principles (no learning, no agency, unsafe automation) ultimately fails both developers and users.

5.3 Documentation and Training: Expertise Assumptions

WCAG documentation, W3C tutorials, and accessibility courses provide implementation guidance invaluable for developers with time to engage them systematically.

5.3.1 Design Assumptions

Documentation assumes:

- Users can allocate substantial time to study (WCAG is 200+ pages)
- Linear learning pathways are practical (start with principles, progress through criteria)
- Reference manual format is accessible to all learning styles
- Users need comprehensive understanding before implementation

5.3.2 Care Gaps

Time Investment Barrier: A developer building a weekend side project cannot spend 20 hours studying WCAG before implementing basic accessibility. Documentation designed for comprehensive learning creates insurmountable barriers for time-constrained contexts.

Fragmentation: While WCAG provides authoritative standards, practical implementation guidance is scattered across blog posts, Stack Overflow threads, and outdated tutorials. Developers must curate their own learning resources, often encountering conflicting advice.

Task-Orientation Mismatch: Developers typically approach documentation with specific questions ("How do I make this dropdown keyboard accessible?"), not abstract principles ("What is WCAG's Operable principle?"). Reference manual structures serve compliance auditors better than developers seeking implementation guidance.

5.4 Enterprise Design Systems: Resource Prerequisites

Some organizations address accessibility through comprehensive design systems (Material Design, Carbon Design System) providing pre-built accessible components.

5.4.1 Design Assumptions

Design systems assume:

- Organizations can adopt complete design languages
- Applications can be built from provided components
- Teams have capacity to integrate and maintain dependencies

5.4.2 Care Gaps

Adoption Costs: Design systems require architectural commitment. Existing applications cannot incrementally adopt without significant refactoring.

Customization Constraints: Design systems make accessibility accessible only if developers accept the system's choices. Projects requiring custom design must implement accessibility independently.

Coverage Gaps: Even comprehensive systems cannot provide components for every use case. Novel interactions require developers to implement accessibility independently, returning to the original problem.

Design systems serve enterprises excellently but provide no pathways for resource-constrained developers or projects requiring customization.

6 Toward Care-Centered Infrastructure

The Care Framework implies a shift from compliance-optimized tooling toward developer-centered infrastructure. We propose the utility layer model and identify critical research problems.

6.1 The Utility Layer Model

Modern web development relies on composable utilities–small, focused tools solving specific problems that developers combine as needed. CSS utility frameworks (Tailwind), JavaScript utility libraries (Lodash), and React Hook collections exemplify this pattern. Developers do not adopt monolithic frameworks; they compose ecosystems from utilities matching their specific needs.

Accessibility tooling, by contrast, skews toward monolithic solutions: comprehensive checkers, complete design systems, all-or-nothing overlays. We propose a **utility layer for accessibility**–focused tools addressing specific accessibility challenges that developers can adopt incrementally and compose contextually.

6.1.1 Characteristics of Care-Centered Utilities

Focused Scope: Each utility solves one problem well. A color contrast optimizer handles colors; a focus management utility handles keyboard navigation; a heading hierarchy checker validates document structure. Developers adopt only the utilities addressing their specific needs.

Composability: Utilities work independently but integrate smoothly. A developer might use a color optimizer and a form accessibility utility together without requiring a shared framework.

Low Adoption Friction: Utilities should require minimal configuration, integrate with existing workflows, and provide immediate value. Installation should be a single command; basic usage should require no configuration.

Meaningful Customization: Utilities should expose choices framed in developer-understandable terms, not technical jargon. Rather than "set delta-E threshold," options might present: "Preserve exact colors (may fail)" vs. "Ensure compliance (may adjust colors)." This surfaces actual tradeoffs without requiring color science expertise.

Educational Design: Utilities should explain their actions and teach concepts through use. When a color optimizer adjusts a hue, it might note: "Darkened to increase contrast while keeping the color 'blue' (hue unchanged)." Repeated use builds mental models without formal study.

6.2 The Missing Ecosystem: Infrastructure Comparison

To illustrate the infrastructure gap, we compare the tooling ecosystems available for modern web development versus accessibility implementation. Table 1 reveals a stark asymmetry.

Development Concern	**Exemplary Tooling Ecosystem**
Scaffolding & Initialization	Vite, Create React App, Next.js templates
Component Libraries	Material UI, shadcn, Radix UI
Developer Experience	ESLint, Prettier, TypeScript
Testing Frameworks	Jest, Vitest, Testing Library, Cypress
Performance & Quality	Lighthouse, Web Vitals, bundle analyzers
Deployment & CI/CD	Vercel, Netlify, GitHub Actions
General Utilities	Lodash, date-fns, validator.js
Accessibility Implementation	**WCAG (reference manual), axe, Lighthouse (checkers)**

Table 1: Comparison of tooling ecosystems. Accessibility implementation lacks the layered, composable infrastructure available for other core development concerns.

This comparison reveals the fundamental disparity: frontend development benefits from layered, composable infrastructure built over two decades of open-source contribution. Accessibility implementation, by contrast, has compliance documentation and violation detection–but lacks the building, testing, and deployment infrastructure that makes modern web development tractable.

Critically, frontend frameworks did not emerge fully-formed. The ecosystem evolved through utility-first tools (CSS evolved into Sass/Less, then utility frameworks like Tailwind, then component libraries like Bootstrap and DaisyUI). Common usage patterns emerged, utilities were composed, and higher-level abstractions were built atop successful lower-level tools. This organic evolution occurred because developers building web applications *needed* these utilities and created them to reduce their own cognitive load.

The accessibility ecosystem has not followed this trajectory because the developers who most need accessibility utilities–resource-constrained individuals–lack capacity to build and maintain them. The result is that accessibility remains a "huge project to take on" (as one Hacker News commenter noted [2]) rather than a natural part of the development workflow. Care-centered infrastructure aims to catalyze this ecosystem development through intentional investment in foundational utilities.

6.3 Open Research Problems

Building care-centered accessibility infrastructure requires addressing several critical research problems. Solving these is the essential R&D agenda for realizing the utility layer model.

6.3.1 The High-Impact Problem Space: Evidence for Prioritization

The research agenda for a care-centered utility layer must be empirically grounded in the actual distribution of accessibility failures. Chronic data reveals a striking concentration: six specific failure types account for 96% of all detectable WCAG violations on home pages, a pattern that has remained consistent over the past five years [**?**]. Table 2 presents this epidemiological data.

WCAG Failure Type	**% of Home Pages**
Low contrast text	79.1%
Missing alternative text for images	55.5%
Missing form input labels	48.2%
Empty links	45.4%
Empty buttons	29.6%
Missing document language	15.8%

Table 2: Prevalence of the six most common WCAG 2 failures, constituting 96% of all detectable errors. Data from WebAIM Million (2025) [**?**].

This concentration defines a clear, high-impact roadmap for the Care Framework's utility layer. Rather than attempting to address all possible accessibility issues simultaneously, initial efforts should prioritize these six failure categories, which alone would significantly improve web accessibility. Our case study (Section 4) demonstrates this targeted approach for the most prevalent failure: *low contrast text*. The critical research challenge, therefore, is to design safe, effective, and adoptable utilities for the remaining high-frequency problems. This evidence-driven focus shifts the research agenda from abstract coverage to targeted intervention with maximal impact, directly informing the specific research problems that follow.

6.3.2 Defining Automation Safety Boundaries for High-Impact Problems

The Core Question: Given the failure distribution in Table 2, which of these high-impact problems can be safely automated, and which require human judgment?

This question has no universal answer–safety depends on context availability and risk tolerance. However, we can establish principles:

Safe Automation Criteria:

- **Well-defined input/output space**: Problem can be formulated with clear constraints
- **Measurable success criteria**: Outcomes can be validated objectively
- **Minimal context-dependence**: Solution does not vary substantially based on unavailable context
- **Low harm potential**: Incorrect automation produces detectable, non-harmful failures

Example: Color Contrast. Input space is constrained (RGB triplets), objective is measurable (contrast ratio $\geq$ 4.5:1), context-dependence is minimal (contrast requirements are universal), and failures are obvious (colors remain too similar). This aligns with its status as the most common failure.

Unsafe Automation Criteria:

- Requires semantic understanding of content
- Depends on context unavailable to tools
- Failures produce subtle, harmful user experiences
- No objective validation method exists

Example: Alternative Text. Requires understanding image purpose in document context, depends on author intent unavailable to algorithms, failures harm screen reader users through incorrect information, and validation requires human judgment of appropriateness. This explains why it remains a prevalent failure despite being the second most common.

Research Agenda:

- Systematically catalog accessibility problems–particularly high-prevalence ones from Table 2–along automation safety dimensions
- Develop heuristics for identifying safe automation domains within this high-impact set
- Create validation frameworks for evaluating automated fixes
- Establish confidence communication protocols (tools should state certainty levels)

This research is critical for building tools that help without harming–automating safely while refusing unsafe automation.

6.3.3 Perceptual Safety in Automated Adjustments

Automated adjustments improving one accessibility dimension can harm other users. High contrast benefits users with low vision but can trigger vestibular disorders causing dizziness and nausea. Simplified layouts aid cognitive accessibility but may reduce information density needed by expert users.

Research Questions:

- What parameter ranges balance competing accessibility needs?
- How can tools provide user control over accessibility modifications?
- What safe defaults exist when user preferences are unavailable?
- How do we empirically validate safety across diverse disability contexts?

6.3.4 Sustainable Maintenance Models

Accessibility utilities are public infrastructure requiring sustained maintenance as web standards evolve, assistive technologies change, and browser implementations shift. Yet open-source maintainers typically work unpaid, leading to burnout and abandonment.

Research Questions:

- What funding models can sustain public accessibility infrastructure?
- How can maintenance burden be distributed across stakeholder communities?
- What governance structures enable community stewardship of utilities?
- How do we prevent corporate capture of essential public tooling?

6.3.5 Evaluation Frameworks for Care

How do we measure whether tooling embodies care principles? Traditional metrics (usage statistics, GitHub stars) capture popularity but not care quality. We need frameworks evaluating:

- **Adoption accessibility**: How quickly can developers with no accessibility background achieve basic compliance?
- **Context-adaptiveness**: How well do tools function across resource-constrained contexts?
- **Learning transfer**: Do developers using tools gain accessibility knowledge generalizing beyond the tool?
- **Empowerment**: Do tools increase developer agency and capability?

7 Discussion

7.1 Care as Infrastructure Critique

The Care Framework positions accessibility tooling as an infrastructure problem, not merely a technical or educational challenge. Infrastructure studies [5, 19] emphasize that infrastructure is relational–what counts as infrastructure depends on one's position. For enterprise accessibility teams, WCAG compliance checkers are infrastructure enabling their work. For individual developers, these same tools are obstacles requiring additional infrastructure to use effectively.

This relational view explains the persistence of inaccessibility despite extensive tooling. The tools exist; they simply don't function as infrastructure for most developers. Care-centered design means building infrastructure that serves as infrastructure for resource-constrained contexts, not only resource-rich ones.

7.2 Developer Experience as Disability Justice

Focusing on developer experience risks centering technical convenience over disabled users' needs. We must be explicit: the Care Framework is *not* about making accessibility optional or reducing requirements. Rather, it recognizes that **developer experience is part of the accessibility problem**. Inaccessible tooling produces inaccessible websites, harming disabled people and people with disabilities. Improving developer experience is an intervention in accessibility outcomes, not a distraction from them.

Moreover, many people with disabilities are themselves developers [8]. Inaccessible development tooling–checkers requiring visual parsing of lengthy reports, documentation assuming specific reading styles, testing tools requiring hearing screen reader audio–excludes disabled developers from participating in web creation. Care for developer experience includes care for developers with disabilities navigating toolchains designed for non-disabled users.

We acknowledge the ongoing debate within disability communities about person-first ("people with disabilities") versus identity-first ("disabled people") language. Different communities and individuals have strong preferences. In this work, we use both constructions to respect this diversity, recognizing that language preferences reflect important differences in how people understand disability and identity.

7.3 The Limits of Automation

The Care Framework emphasizes making tools accessible, but we must be clear about automation's limits. Some accessibility work cannot be automated because it requires

understanding content meaning, user context, and domain-specific conventions. Alternative text for images, for instance, requires understanding image purpose in document context–a task requiring human judgment that automated systems cannot reliably perform.

Care-centered tools must be explicit about these limits. Rather than attempting universal automation (which fails) or disclaiming all automation (which abandons developers), tools should articulate *where* automation is safe, *where* it's uncertain, and *where* human judgment is essential. This boundary-drawing is itself a research problem requiring continued investigation, as Section 6.2.1 discusses.

The distinction between safe automation (color contrast adjustment preserving hue) and unsafe automation (alt text generation, overlay injection) is not always obvious. Care requires tools to communicate confidence: "I can safely fix this color pair" versus "I cannot determine appropriate alt text–this requires your judgment about image purpose." Transparency about automation boundaries respects both developer agency and user safety.

7.4 Institutional and Systemic Change

Individual tools, however well-designed, cannot alone resolve systemic resource asymmetries. The Care Framework requires institutional support, which must first confront the **misaligned incentives** that perpetuate the current ecosystem:

Tool Vendor Incentives: The prevailing business model relies on enterprise sales and audit services, not on empowering individual developers, directing investment away from low-friction public utilities.

Academic Incentives: Reward structures prioritize novel algorithms over the sustained maintenance of public infrastructure [20], which is essential but less visible work.

Standards Development Incentives: The necessary focus on precise, auditable compliance criteria can sideline the implementer's experience, creating a knowing-doing gap.

Realigning these incentives requires:

Funding Public Infrastructure: Governments, foundations, and corporations benefiting from the web should fund maintenance of public accessibility utilities as infrastructure investment, not charitable donation. Models like Sovereign Tech Fund, Mozilla Open Source Support, and the Chan Zuckerberg Initiative's Essential Open Source Software for Science program demonstrate that public infrastructure funding is possible and effective.

Academic Recognition: Research institutions should recognize infrastructure maintenance and utility development as valid scholarly contribution, not only novel algorithms

or large-scale studies. Tenure and promotion criteria that value only "novel" research discourage sustained maintenance of essential tooling.

Educational Reform: Web development education should integrate accessibility from the start, treating it as foundational skill rather than advanced specialization. Curricula that teach accessibility in separate, optional courses perpetuate its marginalization.

Policy Alignment: Accessibility mandates should acknowledge resource asymmetries and provide support for small organizations and individual developers, not only impose requirements. Regulations that mandate compliance without providing implementation resources create impossible burdens for under-resourced developers.

The Care Framework provides conceptual foundations, but realizing its vision requires coordinated action across multiple stakeholder communities.

7.5 Toward Actionable Change

The question is not merely "what should exist?" but "how do we get there?" We propose concrete actions for different stakeholder groups to correct the identified incentive failures:

For Researchers:

- Investigate automation safety boundaries through systematic problem categorization
- Conduct participatory design studies with resource-constrained developers
- Develop evaluation frameworks measuring care dimensions, not only technical accuracy
- Recognize infrastructure maintenance as legitimate scholarly contribution and advocate for academic reward systems that value it

For Tool Builders:

- Design with resource-constrained contexts as primary users, not edge cases
- Frame customization in terms of developer-understandable tradeoffs, not technical parameters
- Be explicit about automation boundaries–communicate what tools can and cannot safely do
- Contribute to composable utility ecosystems rather than building monolithic solutions

For Funders:

- Fund public accessibility infrastructure as essential web utilities, not niche tools
- Support sustained maintenance, not only initial development
- Prioritize projects serving resource-constrained developers
- Recognize that accessibility tooling investment yields broad social impact

For Educators:

- Integrate accessibility throughout web development curricula
- Teach using care-centered tools that students can actually adopt
- Frame accessibility as standard practice, not advanced specialization
- Assign projects requiring accessible implementation, not just auditing

For Standards Bodies:

- Develop implementation guidance complementing compliance criteria
- Create task-oriented documentation for developers, not only auditors
- Collaborate with tool builders to ensure standards are implementable
- Consider resource asymmetries when developing new requirements

These actions are not exhaustive, but they provide starting points for translating the Care Framework from concept to practice.

8 Limitations

This work has several limitations. First, while we provide a case study demonstrating care principles' impact on adoption, we lack large-scale empirical validation across diverse tool categories and developer populations. Future work should involve participatory design studies with resource-constrained developers to validate and refine the framework.

Second, we focus on web accessibility, but the Care Framework likely applies to other accessibility domains (mobile applications, desktop software, embedded systems, physical environments). Extending the framework across domains requires further research examining domain-specific constraints.

Third, we emphasize developer experience but provide limited attention to how tool design affects disabled users' actual experiences. Care-centered tools must ultimately be evaluated by their impact on accessibility outcomes for disabled people, not merely adoption metrics. The connection between improved developer experience and improved user experience, while logical, requires empirical validation.

Fourth, our automation boundary analysis provides principles but lacks comprehensive problem categorization. Systematically mapping accessibility problems along automation safety dimensions would strengthen the framework's practical applicability.

Finally, we acknowledge potential misuse. "Care" rhetoric could be co-opted to justify lower accessibility standards ("we tried our best given constraints") or to excuse inaccessibility. We emphasize: care is not excuse but obligation. Care means providing pathways to compliance, not accepting inaccessibility as inevitable.

9 Related Work

9.1 Developer Barriers in Accessibility Implementation

Extensive research documents barriers developers face implementing accessibility. Trewin et al. [9] conducted interviews with web developers at a large technology company, identifying challenges including lack of awareness, inadequate training, and competing priorities. Their work established that even developers in resource-rich environments struggle with accessibility, suggesting barriers are not merely resource constraints but also knowledge and tooling gaps. Our work extends this by arguing that knowledge gaps reflect tooling design choices that assume expertise rather than building pathways to it.

Potluri et al. [8] examined how developers–including those with disabilities–misunderstand assistive technology usage patterns, finding significant gaps between developer mental models and actual screen reader user behaviors. This work is particularly important because it challenges the assumption that accessibility expertise naturally flows from disability experience. Even developers with disabilities may lack familiarity with assistive technologies they don't personally use, highlighting the need for tools that teach rather than assume expertise. Our Care Framework's "learning through use" principle directly addresses this finding.

Ross et al. [10] analyzed mobile app accessibility epidemiologically, demonstrating that accessibility issues persist across application domains, developer experience levels, and organizational sizes. This suggests systemic rather than individual failures. Their work supports our infrastructure critique: if accessibility problems are ubiquitous despite developer awareness and organizational mandates, the tooling ecosystem itself must be inadequate.

However, much of this literature frames barriers as individual knowledge gaps or organizational culture problems requiring educational interventions. While education is necessary, our work argues this framing places undue responsibility on developers to overcome barriers rather than on tooling systems to not create barriers in the first place. The Care Framework shifts the locus of intervention from individual education to systemic infrastructure redesign.

9.2 Accessibility Tool Evaluation and Effectiveness

Researchers have evaluated specific accessibility tools and their effectiveness. Abou-Zahra et al. [16] assessed automated accessibility testing tools, finding significant variation in coverage (which WCAG criteria tools detect) and accuracy (false positive/negative rates). They demonstrated that no single tool detects all issues, requiring developers to use multiple tools–a burden amplifying the adoption friction problem we identify.

Alshayban et al. [17] compared popular accessibility checkers for Android applications, demonstrating limited agreement between tools. When tools disagree about what constitutes violations, developers cannot trust any single tool's output. This necessitates manual expert review, which resource-constrained developers cannot perform. Our work complements these tool evaluations by providing a framework for understanding *why* tools designed for verification fail to support implementation–they assume expertise to interpret findings.

Vigo et al. [18] examined barriers to accessibility evaluation itself, including tool complexity and result interpretation challenges. They found that even accessibility specialists struggle with current tools, suggesting tools fail to serve even their primary audience effectively. Our Care Framework's emphasis on explanation over error and empowerment over enforcement directly addresses these evaluation barriers.

This body of work demonstrates that technical accuracy–the primary dimension tool evaluations measure–is necessary but insufficient. Our contribution is providing additional evaluative criteria beyond accuracy: accessibility of access, context-awareness, and empowerment. These dimensions capture whether tools serve as infrastructure for resource-constrained developers, not merely whether they detect violations correctly.

9.3 Care Ethics in HCI and Accessibility

Care ethics has informed recent HCI research, particularly in critical disability studies. Hamraie [11] applies care ethics to physical accessibility design, examining how design practices encode assumptions about bodies and abilities. Their concept of "access friction"–the unnoticed labor required to navigate inaccessible spaces–parallels our notion of cognitive load created by inaccessible tooling. Our contribution extends this analysis from physical spaces to digital development tooling.

Shinohara and Wobbrock [24] employed participant observation methods to understand design from the perspectives of people with disabilities, emphasizing the importance of diverse participation in design processes. While their work focuses on end-user participation, our extension argues that developer experience itself is a care concern–not as alternative to user-focused care but as prerequisite for it. Care requires attending to all actors in the accessibility ecosystem, including those building systems.

Light et al. [25] discuss care as analytical lens in HCI, particularly for understanding technology's role in care work and care relationships. They argue care is undertheorized in HCI despite its centrality to human experience. Our work responds to this call by applying care ethics specifically to infrastructure and tooling design, domains where care is typically not considered relevant. We demonstrate care's applicability beyond direct human-to-human interactions to the systems and tools mediating those interactions.

However, care ethics in accessibility has primarily focused on user experience and direct designer-user relationships. Our contribution is extending care to developer experience and infrastructure design, arguing that caring for those who build systems is inseparable from caring for those who use them.

9.4 Infrastructure Studies and Invisible Work

Star and Ruhleder [5] established infrastructure as relational concept–what functions as infrastructure depends on perspective and context. They identify properties of infrastructure including embeddedness (infrastructure sinks into other structures), transparency (infrastructure becomes invisible when working), and learned as part of membership (using infrastructure requires learning community practices). Our analysis reveals how accessibility tooling fails these properties for resource-constrained developers: it doesn't embed in existing workflows, remains highly invisible due to friction, and requires learning separate from general web development practice.

Bowker and Star [19] examined how infrastructure classifications embed values and assumptions, making some work visible while rendering other work invisible. Their concept of "residual categories"–people and practices that don't fit infrastructure's assumed classifications–applies directly to accessibility tooling: tools designed for enterprise contexts render resource-constrained developers as residual users whose needs are unmet. The Care Framework aims to center these residual categories in infrastructure design.

Ribes and Finholt [20] studied long-term scientific infrastructure maintenance, identifying tensions between initial development (rewarded, visible) and sustained maintenance (essential but invisible). This directly parallels open-source accessibility tooling, where creating novel tools receives recognition while maintaining them does not. Our call for sustainable maintenance models and institutional recognition of maintenance work builds on their analysis.

Star and Strauss [21] examined layers of work–foreground work (visible, rewarded) and background work (invisible, uncompensated). Accessibility implementation often becomes background work for developers: necessary but unrewarded, competing with visible feature development. Care-centered tooling that reduces cognitive load makes accessibility more feasible to address within existing time constraints, bringing it from background to foreground.

9.5 Critical Perspectives on Automation and Overlays

The disability community has extensively documented problems with accessibility overlays. Feingold [13], a disability rights lawyer, articulated how overlays provide legal cover while harming disabled users through poor implementation. WebAIM [14] analyzed overlay widgets, finding they introduce new accessibility barriers while claiming to fix existing ones. Overlay Watch [22], a community initiative, maintains a list of overlay-using websites to help disabled users avoid them.

This critical literature establishes that automation without care–attempting to fix accessibility through black-box JavaScript injection–fails users and removes developer agency. However, the backlash against overlays has created wariness toward *any* automation in accessibility. Our contribution is distinguishing safe, bounded automation (specific, well-defined problems with clear success criteria) from unsafe automation (context-dependent problems requiring semantic understanding). The Care Framework argues automation is not inherently problematic; automation that violates care principles is problematic.

Treviranus [23] discusses the dangers of applying machine learning to accessibility without considering diverse needs and edge cases. Automated systems trained on majority data fail minority users–precisely those accessibility aims to serve. This reinforces our automation boundary principle: tools must be explicit about what they can and cannot safely automate, communicating uncertainty and refusing to automate when confidence is low.

9.6 Developer-Centered Accessibility Approaches

Several researchers have recognized that accessibility implementation tools must serve developers better. Rubano and Vitali [26] argue that "non-disabled developers and designers have fundamental difficulty perceiving the impact of accessibility issues," proposing declarative frameworks (AX) that enforce accessible markup through static analysis. Their work demonstrates that technical abstractions can reduce developer burden—web components that "just work" accessibly without requiring deep WCAG knowledge.

Rubano [28] extends this with innovative testing methodologies that "map complex accessibility concepts onto visual rendering and mouse operability"—making accessibility issues visible to developers in ways they intuitively understand. Lengua et al. [27] further develop this approach, arguing that "persistent critical accessibility issues suggest the current approach to guidelines and tools is unsatisfactory for developers." This work aligns with our Principle 1 (Accessibility of Accessibility): tools must meet developers where they are cognitively.

However, these works focus primarily on *technical* solutions (frameworks, testing architectures) without examining the broader ecosystem gaps or establishing *why* such approaches succeed when they do. Our Care Framework provides the conceptual foundation explaining their effectiveness: they reduce cognitive load (Principle 1), adapt to

developer workflows (Principle 2), and make accessibility knowledge implicit in tooling (Principle 3). Where Rubano proposes specific tools, we propose *design principles* for evaluating and creating any accessibility tool.

Kelly, Sloan, and colleagues [29, 30, 31, 32] pioneered contextual approaches to accessibility, arguing that "holistic approaches are needed beyond mere adherence to guidelines" [29]. Their "Accessibility 2.0" framework [30] and Tangram model emphasize pluralistic, context-aware approaches recognizing diverse user needs and resource constraints. Kelly et al.'s work on developing contexts [32] explicitly addresses resource limitations in accessibility implementation, examining "how developers and practitioners deploy Web services within challenging contexts of limited resources and conflicting priorities."

This body of work is crucial but focuses primarily on *end-user* context—how accessibility must adapt to diverse user populations, technologies, and usage scenarios. Our contribution extends contextual thinking to *developer* context. Just as Kelly et al. argue users need adaptable accessibility, we argue developers need adaptable tooling. The Care Framework applies similar contextual reasoning to tool design: acknowledging varied developer resources, providing meaningful customization, and respecting implementation constraints.

In prior work, we proposed comfort mode frameworks emphasizing user autonomy and personalization [33]. That work established philosophical foundations (accessibility as care, attention to lived context) but proposed implementation approaches (overlay-based adaptation) that subsequent community feedback revealed as harmful [13, 14]. The current work refines those philosophical foundations, explicitly rejects unsafe automation approaches, and grounds the framework in empirical evidence of care-centered design's measurable impact on tool adoption.

These works collectively establish that both users and developers need context-aware, accessible approaches to accessibility. Our contribution synthesizes these insights into a unified Care Framework with explicit automation boundaries, empirical validation, and actionable research directions for building the missing developer-centered infrastructure.

10 Conclusion

Web accessibility's persistent implementation gap reflects not developer indifference but systemic tooling failures. Current accessibility tools optimize for enterprise contexts with dedicated teams, substantial budgets, and specialized expertise, leaving the majority of web developers–individuals, students, nonprofits, small teams building most of the web–without practical implementation pathways.

The Care Framework reframes accessibility tooling design around three principles: making accessibility accessible (tools must be adoptable by developers with limited expertise and time), context-awareness (tools must adapt to varied resource constraints), and empowerment (tools must guide and educate, not merely audit and enforce). We demonstrate through empirical case study that redesigning tools around these principles yields measurable adoption improvements–5.7× growth when CM-Colors shed enterprise assumptions and centered developer needs.

Critical to the framework is establishing automation safety boundaries: what can be safely automated (color contrast adjustment), what cannot (alternative text generation), and what requires research (form labeling). We reject both universal automation (overlays) and automation avoidance, instead advocating for tools that automate safely while transparently communicating limits.

Closing the accessibility gap requires building care-centered infrastructure: composable utilities, focused on specific problems, with low adoption friction, that teach through use. This infrastructure does not yet exist. Table 1 illustrates the disparity–frontend development benefits from layered, composable tooling built over decades, while accessibility implementation has compliance documentation and violation detection but lacks the building, testing, and deployment infrastructure that makes development tractable.

This is not a call to lower accessibility standards or accept inaccessible implementations. Rather, it recognizes that achieving widespread accessibility requires first making accessibility achievable for those building the web. **We are not failing at accessibility because developers don't care. We are failing because we have not yet built the tools that make caring *possible* within the real constraints of building the web.** Care is not accommodation or excuse–it is the practical foundation upon which compliance becomes possible.

The path forward requires coordinated action: researchers investigating automation boundaries and evaluation frameworks; tool builders creating composable utilities; funders supporting sustained maintenance of public infrastructure; educators integrating accessibility as foundational skill; standards bodies developing implementation guidance. Each stakeholder has specific actions they can take to realize the Care Framework's vision.

Beyond compliance is care. But care must begin with making compliance itself accessible. Only then can we move toward a web that welcomes everyone–not through mandate and enforcement alone, but through infrastructure that makes accessibility the natural path forward, the option requiring least resistance rather than most effort. When accessibility is as easy as running `npm install`, it will cease being an afterthought and become simply how we build for the web.

Acknowledgments

My deepest gratitude to Mr. Krishna, whose constancy forms the foundation upon which all my work, including this, quietly rests.

We thank the open-source community for discussions that informed this framework, particularly those who engage daily with the challenges of implementing accessibility in resource-constrained contexts.

Statements and Declarations

10.1 Funding Declaration

No funding was received to assist with the preparation of this manuscript.

10.2 Author Contribution

L.A.R. was responsible for all aspects of this manuscript, including conceptualization, methodology, writing the original draft, and review and editing.

10.3 Competing Interests

The author developed the open-source cm-colors library referenced in this work specifically for the purposes of the research. The library is freely available under GNU GPLv3.0 license with no commercial implications or financial benefits to the author.

References

[1] WebAIM, "The WebAIM Million - An annual accessibility analysis of the top 1,000,000 home pages," 2025. [Online]. Available: https://webaim.org/projects/million/

[2] Hacker News, "What's Been Your Experience Implementing Web Accessibility?" Dec. 2024. [Online]. Available: https://news.ycombinator.com/item?id=46172855

[3] J. C. Tronto, *Moral Boundaries: A Political Argument for an Ethic of Care*. Routledge, 1993.

[4] V. Held, *The Ethics of Care: Personal, Political, and Global*. Oxford University Press, 2006.

[5] S. L. Star, "The Ethnography of Infrastructure," *American Behavioral Scientist*, vol. 43, no. 3, pp. 377-391, 1999.

[6] W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," World Wide Web Consortium, 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/

[7] J. Pfeffer and R. I. Sutton, *The Knowing-Doing Gap: How Smart Companies Turn Knowledge into Action*. Harvard Business School Press, 2000.

[8] V. Potluri, A. Guo, and J. P. Bigham, "Examining Visual Semantic Understanding in Blind and Low-Vision Technology Users," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023.

[9] S. Trewin, B. Cragun, C. Swart, J. Brezin, and J. Richards, "Accessibility challenges and tool features: an IBM Web developer perspective," in *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (WAA)*, 2010.

[10] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock, "Epidemiology as a Framework for Large-Scale Mobile Application Accessibility Assessment," in *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, 2020.

[11] A. Hamraie, *Building Access: Universal Design and the Politics of Disability*. University of Minnesota Press, 2017.

[12] L. A R, "Context-Adaptive Color Optimization for Web Accessibility: Balancing Perceptual Fidelity and Functional Requirements," *arXiv:2512.07623 [cs.HC]*, 2025.

[13] L. Feingold, "Honor the ADA: Avoid Web Accessibility Quick-Fix Overlays," 2020. [Online]. Available: https://www.lllegal.com/2020/08/quick-fix/

[14] WebAIM, "Should I Use an Accessibility Overlay?" 2021. [Online]. Available: https://webaim.org/blog/overlay-fact-sheet/

[15] L. A. R., "Perceptually-Minimal Color Optimization for Web Accessibility: A Multi-Phase Constrained Approach," *arXiv:2512.05067 [cs.HC]*, 2025.

[16] S. Abou-Zahra, E. Brewer, and S. Harper, "Web Accessibility Evaluation Tools: A Survey and Some Improvements," in *International Conference on Web Engineering*, Springer, 2005, vol.157(2), pp.87-100

[17] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 1323-1334.

[18] M. Vigo, J. Brown, and V. Conway, "Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests," in *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, 2013.

[19] G. C. Bowker and S. L. Star, *Sorting Things Out: Classification and Its Consequences*. MIT Press, 2000.

[20] D. Ribes and T. A. Finholt, "The Long Now of Technology Infrastructure: Articulating Tensions in Development," *Journal of the Association for Information Systems*, vol. 10, no. 5, pp. 375-398, 2009.

[21] S. L. Star and A. Strauss, "Layers of Silence, Arenas of Voice: The Ecology of Visible and Invisible Work," *Computer Supported Cooperative Work*, vol. 8, pp. 9-30, 1999.

[22] Overlay Watch, "List of Overlay-Using Websites," 2023. [Online]. Available: https://overlaywatch.org/

[23] J. Treviranus, "The three dimensions of inclusive design: A design framework for a digitally transformed and complexly connected society," PhD thesis, University College Dublin, 2018.

[24] K. Shinohara and J. O. Wobbrock, "In the Shadow of Misperception: Assistive Technology Use and Social Interactions," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 705-714.

[25] A. Light, I. Leach, and J. Frauenberger, "(Re)Searching the Heart of HCI: Rediscovering and Revitalizing Care," in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018.

[26] V. Rubano and F. Vitali, "Making accessibility accessible: strategy and tools," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2021, pp. 1–6.

[27] C. Lengua, V. Rubano, and F. Vitali, "Aligning accessibility design to non-disabled people's perceptions," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2022, pp. 1–6.

[28] V. Rubano, "On making web accessibility more accessible: strategy and tools for social good," PhD thesis, Alma Mater Studiorum - Università di Bologna, 2023.

[29] D. Sloan, A. Heath, F. Hamilton, B. Kelly, H. Petrie, and L. Phipps, "Contextual web accessibility—maximizing the benefit of accessibility guidelines," in *Proceedings of the 2006 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, 2006, pp. 121–131.

[30] B. Kelly, D. Sloan, S. Brown, J. Seale, P. Lauke, S. Ball, and S. Smith, "Accessibility 2.0: Next steps for web accessibility," *Journal of Access Services*, vol. 6, no. 1–2, pp. 265–294, 2009.

[31] B. Kelly, L. Nevile, D. Sloan, S. Fanou, R. Ellison, and L. Herrod, "From web accessibility to web adaptability," *Disability and Rehabilitation: Assistive Technology*, vol. 4, no. 4, pp. 212–226, 2009.

[32] B. Kelly, S. Lewthwaite, and D. Sloan, "Developing countries; developing experiences: approaches to accessibility for the real world," in *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, 2010, pp. 1–10.

[33] L. A. R., "Beyond Compliance: A User-Autonomy Framework for Inclusive and Customizable Web Accessibility," *arXiv:2506.10324 [cs.HC]*, 2025.