**Tools and Techniques for Testing**

**Automated Tools**

Automated tools scan web pages or apps to detect many common accessibility issues quickly. They can’t catch everything but provide a great starting point.

**Popular Automated Tools:**

* **Axe** (by Deque Systems)
  + Browser extension and API-based tool
  + Detects many WCAG 2.1 violations such as missing labels, ARIA misuse, color contrast issues
  + Integrates into CI/CD pipelines for continuous testing
* **Lighthouse** (by Google)
  + Built into Chrome DevTools
  + Audits accessibility along with performance and SEO
  + Highlights issues and provides suggestions
* **WAVE** (by WebAIM)
  + Browser extension and online service
  + Visualizes accessibility errors on page with icons and indicators
  + Good for spotting structural issues and missing alt text
* **tota11y** (by Khan Academy)
  + Lightweight browser bookmarklet
  + Visual overlays for checking accessibility features like ARIA, color contrast, labels

**Color Contrast Checkers**

Color contrast is critical for readability by users with low vision or color blindness. Automated tools may flag contrast issues but dedicated contrast checkers allow detailed analysis.

**Popular Color Contrast Tools:**

* **WebAIM Contrast Checker**
  + Enter foreground and background colors to verify WCAG 2.1 contrast ratios
  + Shows pass/fail for Level AA and AAA
* **Accessible Colors**
  + Suggests alternative color combinations that meet accessibility criteria
* **Color Contrast Analyzer (from TPGi)**
  + Desktop tool to analyze color contrast in apps or screenshots
  + Supports multiple formats and simulates color blindness

**Manual Testing Walkthrough**

While automated tools are helpful, manual testing is essential to evaluate user experience and complex interactions.

**Key Manual Testing Steps:**

1. **Use assistive technologies:** Screen readers, keyboard navigation, magnifiers.
2. **Test interactive components:** Forms, buttons, menus for operability and focus management.
3. **Evaluate content clarity:** Readability, instructions, error feedback.
4. **Check responsive design:** Behavior under different screen sizes and orientations.
5. **Review dynamic updates:** Use of ARIA live regions and announcements.

**How to Interpret and Prioritize Test Results**

1. **Categorize Issues by Impact and Frequency**
   * *Critical:* Blocks access or makes content unusable (e.g., missing form labels, keyboard traps)
   * *High:* Significantly degrades experience (e.g., poor contrast on key text)
   * *Medium:* Causes inconvenience but workarounds exist (e.g., minor focus order glitches)
   * *Low:* Cosmetic or rarely encountered (e.g., decorative images missing alt text)
2. **Align Issues with WCAG Levels and User Impact**
   * Focus first on *Level A* and *AA* criteria since they represent baseline and recommended standards.
   * Prioritize fixes that impact core user tasks and content accessibility.
3. **Cross-Reference Automated and Manual Findings**
   * Automated tools catch many low-hanging fruits but miss context-dependent issues.
   * Manual testing uncovers real user experience problems and edge cases.
4. **Plan Remediation Iteratively**
   * Fix critical and high-impact issues first to improve immediate accessibility.
   * Schedule medium and low priority fixes in future sprints or updates.
5. **Document and Track Issues Clearly**
   * Use issue trackers with detailed descriptions, WCAG references, screenshots, and reproduction steps.
   * Include guidance or code snippets for developers where possible.

**Real-World Example**

A retail website audit using Axe and WAVE found missing labels on form inputs and poor contrast on navigation links (automated). Manual testing revealed the shopping cart modal was not keyboard accessible, and screen reader users missed dynamic updates when items were added.

**Prioritization:**

* Immediate fix: Add labels and fix contrast issues (Level A and AA).
* Next sprint: Improve keyboard focus in modal and add ARIA live announcements.

**Summary Table of Tools and Techniques**

| **Tool/Technique** | **Purpose** | **Strengths** | **Limitations** |
| --- | --- | --- | --- |
| Axe | Automated WCAG checks | Detailed, integrates with CI/CD | Misses context-specific issues |
| Lighthouse | Automated audit + performance | Built-in Chrome, easy to use | Limited ARIA and dynamic tests |
| WAVE | Visual error highlighting | Good for structural review | Some false positives |
| tota11y | Visual overlays | Lightweight, educational | Basic scope |
| Color Contrast Checkers | Verify color contrast ratios | Precise ratio checks | Does not check context |
| Manual Testing | User experience validation | Finds real-world, complex issues | Time-consuming, requires expertise |
| Accessibility Tree Inspection | Debug browser exposure | Deep insight into accessibility roles | Requires developer knowledge |