Engineer

axe

the accessibility engine this site is audited with

What it is

axe, published as the open-source JavaScript library axe-core, is a set of automated accessibility rules that runs inside a web page, inspects what the browser has rendered, and reports each element that breaks one of those rules. Deque Systems, an accessibility company, maintains it under the Mozilla Public License 2.0. Other tools build on it: the accessibility category in Chrome's Lighthouse weights its audits by axe's impact ratings, and Playwright's own guide to accessibility testing runs axe through a small companion package.

The engine is a single script. A test runner or a browser extension loads it into the page under test and calls `axe.run`, and the engine walks the document, including nested frames and shadow DOM, applying every rule whose conditions match. Because it reads the page after the browser has laid it out, it can check things the source alone does not show: whether text is visible, which colour it sits on, whether an element is hidden. That is also why some rules need a real browser. The project says that its colour-contrast rule does not work in JSDOM, the simulated document that many unit-test setups use.

Each rule carries tags. One of them says which WCAG version and level the rule belongs to, such as `wcag2a`, `wcag2aa`, `wcag21aa` or `wcag22aa`, or marks it `best-practice` when no success criterion requires it. Other tags tie rules to EN 301 549, Section 508 or the W3C's conformance testing rules. The tags are how a team decides what a run checks.

What comes back is not a single pass or fail but four lists:

- `violations`: elements that failed a rule, each with an impact rated minor, moderate, serious or critical, a short summary of how to fix it, and a link to a longer explanation.

- `passes` : elements that passed, which is how a team can see that a rule actually ran.
- `incomplete` , which the documentation also calls “needs review”: elements the engine could neither pass nor fail with certainty, and which a person has to look at.
- `inapplicable` : rules that had nothing to test, such as the video rules on a page without video.

Use the concept in a decision

Put “What is axe?” to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The decision axe serves is which parts of accessibility a team hands to a machine, and whether a failure should stop a release. Two properties make it suited to that job. It is cheap to run, so it can check every build instead of one audit a year. And its makers’ stated aim is zero false positives, bugs aside: anything the engine is unsure of goes to the review list rather than the violations list. A check that cries wolf gets switched off; one that only reports what it can prove can fail a build without anyone resenting it.

The limit is coverage. Deque says axe-core finds on average 57 percent of WCAG issues automatically and returns uncertain results for manual review. The figure describes issues found, not the share of success criteria a machine can assess.

What stays out of reach is mostly judgement. axe can confirm that an image has alternative text, that a button has a name and that a form field has a label. It cannot tell whether the text describes the image, whether the name says what the button does, or whether the page makes sense when it is read aloud in order. Those questions need a person with a screen reader and a keyboard, and the time a machine saves on the mechanical checks is time that person can spend on them.

The choice of tags is a trade-off of its own. Running only the WCAG-tagged rules gives a result that maps cleanly to a conformance claim, and to nothing else. Adding the best-practice rules catches problems that WCAG does not name but users still meet, such as content sitting outside every landmark or a page with no top-level heading. The cost is a few failures that no conformance audit would cite.

What it looks like in practice

Two pages on this site show what the engine is trusted with.

In one platform's test suite it sits beside four layers of tests, and an axe-core gate asserts zero WCAG 2.0 and 2.1 A and AA violations, which turns accessibility into a build failure rather than an audit finding months later. The point is timing: a defect that fails the build is fixed by the person who just wrote it, while the change is still open.

On this site, the credits page describes the engine auditing a set of representative pages, one per template in every language, against WCAG 2.2 A and AA in both colour schemes, as part of the gate rather than as a report beside it. Running in both colour schemes means a contrast failure that appears only in the dark scheme can still stop the build. Sampling one page per template relies on pages built from the same template sharing their structure, which is what let 38 pages stand in for a site of 2,268 in October 2026. Automated rules do not cover every judgement WCAG requires, which is why the site keeps its record per success criterion rather than as a pass mark.

Where it goes wrong

Most axe failures are failures of trust: a run that checked less than it seemed to, or a result read as more than it says.

A tag list leaves rules out. Tags are not cumulative. `wcag2aa` holds only the rules for level AA criteria in WCAG 2.0, so a run limited to it skips every level A rule, and the API documentation says outright that both `wcag2a` and `wcag2aa` are needed. In the same way, `wcag21aa` holds only rules for criteria that 2.1 added, and a run limited to WCAG tags never executes a `best-practice` rule. The rule descriptions list the WCAG 2.2 rules as disabled by default, so a run given no tags leaves them out; naming their tag turns them on. Name every tag the run should cover, and keep the list beside the claim it supports:

```
const results = await axe.run(document, {
    runOnly: {
        type: "tag",
        values: ["wcag2a", "wcag2aa", "wcag21a", "wcag21aa", "wcag22aa"],
    },
});
```

A rule that appears in none of the four lists did not run at all, so read the lists once whenever the configuration changes.

A clean run is read as an accessible page. Zero violations means that the rules which ran could prove no failure. It says nothing about whether the alternative text is right, whether focus moves in a sensible order, or whether someone using a screen reader can finish the task. Treat the run as the floor, and schedule the checks it cannot make.

The review list is dropped. A gate that fails on violations alone lets the `incomplete` results through without anyone seeing them. Those are the elements the engine looked at and left undecided, which makes them the cases most in need of a person. Send them to someone who will read them, or fail the build on them too.

The page is tested in one state. The engine checks what is rendered at the moment it runs. A closed menu, an unopened dialog, an error message that appears only after a bad submission, and the other colour scheme are all outside a run against the page as it first loads. Drive the page into each state that matters, and run the engine again in each one.

The engine runs where nothing is drawn. A simulated document has no layout and no computed colour, and the project lists colour contrast as a rule that does not work there. Unit tests with axe still catch structural problems; contrast and anything else visual belong in a real browser.

A rule is switched off to get a release out. Disabling a rule for the whole run because one component trips it hides that rule’s failures everywhere else, including in pages written later. Exclude the one element instead, with a note saying why and when the exclusion should go.

Further reading

- axe-core — the engine's source, its stated principles and the list of supported browsers.
- Rule Descriptions — every rule with its impact and tags, grouped by WCAG version, best practice, AAA, experimental and deprecated.
- axe-core API — the tags, the `runOnly` option, how to include or exclude parts of a page, and the shape of the four result lists.
- Accessibility testing — Playwright's guide to running axe inside browser tests, including scanning a page after interacting with it.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is axe?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/axe/