

[Technologies ▾](#)[References & Guides ▾](#)[Feedback ▾](#)[Sign in](#) 

Introduction to cross browser testing

[↑ Overview: Cross browser testing](#)[Next →](#)

This article starts the module off by providing an overview of the topic of (cross) browser testing, answering questions such as "what is cross browser testing?", "what are the most common types of problems you'll encounter?", and "what are the main approaches for testing, identifying, and fixing problems?"

Prerequisites: Familiarity with the core HTML, CSS, and JavaScript languages.


Objective: To gain an understanding of the high-level concepts involved in cross browser testing.

What is cross browser testing?

Cross browser testing is the practice of making sure that the web sites and web apps you create work across an acceptable number of web browsers. As a web developer, it is your responsibility to make sure that not only do your projects work, but they work for all your users, no matter what browser, device, or additional assistive tools they are using. You need to think about:

- Different browsers other than the one or two that you use regularly on your devices, including slightly older browsers that some people might still be using, which don't support all the latest, shiniest CSS and JavaScript features.
- Different devices with different capabilities, from the latest greatest tablets and smartphones, through smart TVs, right down to cheap tablets and even older feature phones that may run browsers with limited capabilities.
- People with disabilities, who use the Web with the aid of assistive technologies like screenreaders, or don't use a mouse (some people use only the keyboard).

Remember that you are not your users — just because your site works on your Macbook Pro or high-end Galaxy Nexus, doesn't mean it will work for all your users — there's a whole lot of testing to be done!


 **Note:** [Make the web work for everyone](#) provides more useful perspective on the different browsers people use, their market share, and related cross browser compatibility issues.

We should explain a few bits of terminology here. To start with, when we talk about sites "working cross browser", we are really saying that they should provide an acceptable user experience across different browsers. It is potentially OK for a site to not deliver the exact same experience on all browsers, as long as the core functionality is accessible in some way. On modern browsers you might get something animated, 3D and shiny, whereas on older browsers you might just get a flat graphic representing the same information. As long as the site owner is happy with this, then you have done your job.

On the other hand, it is not OK for a site to work fine for sighted users, but be completely inaccessible for visually impaired users because their screen reader application can't read any of the information stored on it.

Second, when we say "across an acceptable number of web browsers", we don't mean 100% of the browsers in the world — this is just about impossible. You can make some informed calls as to what browsers and devices your users will be using (as we'll discuss in the second article in the series — see [Gotta test 'em all?](#)), but you can't guarantee everything. As a web developer, you need to agree a range of browsers and devices that the code definitely needs to work on with the site owner, but beyond that, you need to

code defensively to give other browsers the best chance possible of being able to use your content. This is one of the great challenges of web development.

 **Note:** We'll cover defensive coding later in the module too.

Why do cross browser issues occur?

There are many different reasons why cross browser issues occur, and note that here we are talking about issues where things behave different across different browsers / devices / browsing preferences. Before you even get to cross browser issues, you should have already fixed out bugs in your code (see [Debugging HTML](#), [Debugging CSS](#), and [What went wrong? Troubleshooting JavaScript](#) from previous topics to refresh your memory if needed).

Cross browser issues commonly occur because:

- sometimes browsers have bugs, or implement features differently. This situation is a lot less bad than it used to be; back when IE4 and Netscape 4 were competing to be the dominant browser in the 1990s, browser companies deliberately implemented things differently to each other to try to gain competitive advantage, which made life hell for developers. Browsers are much better at following standards these days, but differences and bugs still creep through sometimes.
- some browsers may have different levels of support for technology features to others. This is inevitable when you are dealing with bleeding edge features that browsers are just getting round to implementing, or if you have to support really old browsers that are no longer being developed, which may have been frozen (i.e. no more new work done on them) a long time before a new feature was even invented. As an example, if you want to use cutting edge JavaScript features in your site, they might not work in older browsers. If you need to support older browsers, you might have to not use those, or convert your code to old fashioned syntax using some kind of cross-compiler where needed.
- some devices may have constraints that cause a web site to run slowly, or display badly. For example, if a site has been designed to look nice on a desktop PC, it will probably look tiny and be hard to read on a mobile device. If your site includes a load of big animations, it might be ok on a high spec tablet, but might be sluggish or jerky on a low end device.

and more reasons besides.

In later articles, we'll explore common cross browser problems, and look at solutions to those.

Workflows for cross browser testing

All of this cross browser testing business may sound time consuming and scary, but it needn't be — you just need to plan carefully for it, and make sure you do enough testing in the right places to make sure you don't run into unexpected problems. If you are working on a large project, you should be testing it regularly, to make sure that new features work for your target audience, and that new additions to the code don't break old features that were previously working.

If you leave all the testing to the end of a project, any bugs you uncover will be a lot more expensive and time consuming to fix than if you uncover them and fix them as you go along.

The workflow for testing and bug fixes on a project can be broken down into roughly the following four phases (this is only very rough — different people may do things quite differently to this):

Initial planning > Development > Testing/discovery > Fixes/iteration

Steps 2–4 will tend to be repeated as many times as necessary to get all of the implementation done. We will look at the different parts of the testing process in much greater detail in subsequent articles, but for now let's just summarize what may occur in each step.

Initial planning

In the initial planning phase, you will probably have several planning meetings with the site owner/client (this might be your boss, or someone from an external company you are building a web site for), in which you determine exactly what the web site should be — what content and functionality should it have, what should it look like, etc. At this point you'll also want to know how much time you have to develop the site — what is their deadline, and how much are they going to pay you for your work? We won't go into much detail about this, but cross-browser issues can have a serious effect on such planning.


Once you've got an idea of the required featureset, and what technologies you will likely build these features with, you should start exploring the target audience — what

browsers, devices, etc. will the target audience for this site be using? The client might already have data about this from previous research they've done, e.g. from other web sites they own, or from previous versions of the web site you are now working on. If not, you will be able to get a good idea by looking at other sources, such as usage stats for competitors, or countries the site will be serving. You can also use a bit of intuition.

So for example, you might be building an e-commerce site that serves customers in North America. the site should work entirely in the last few versions of the most popular desktop and mobile (iOS, Android, Windows phone) browsers — this should include Chrome (and Opera as it is based on the same rendering engine as Chrome), Firefox, IE/Edge, and Safari. It should also provide an acceptable experience on IE 8 and 9, and be accessible with WCAG AA compliance.

Now you know your target testing platforms, you should go back and review the required featureset and what technologies you are going to use. For example, if the e-commerce site owner wants a WebGL-powered 3D tour of each product built into the product pages, they will need to accept that this just won't work in IE versions before 11. You'd have to agree to provide a version of the site without this feature to users of older IE versions.

You should compile a list of the potential problem areas.

 **Note:** You can find browser support information for technologies by looking up the different features on MDN — the site you're on! You should also consult caniuse.com, for some further useful details.

Once you've agreed on these details, you can go ahead and start developing the site.

Development

Now on to the development of the site. You should split the different parts of the development into modules, for example you might split the different site areas up — home page, product page, shopping cart, payment workflow, etc. You might then further subdivide these — implement common site header and footer, implement product page detail view, implement persistent shopping cart widget, etc.

There are multiple general strategies to cross browser development, for example:

- Get all the functionality working as closely as possible in all target browsers. This may involve writing different code paths that reproduce functionality in different ways aimed at different browsers, or using a Polyfill to mimic any missing support using JavaScript or other technologies, or using a library that allows you to write a

single bit of code and then does different things in the background depending on what the browser supports.

- Accept that some things aren't going to work the same on all browsers, and provide different (acceptable) solutions in browsers that don't support the full functionality. Sometimes this is inevitable due to device constraints — a cinema widescreen isn't going to give the same visual experience as a 4" mobile screen, regardless of how you program your site.
- Accept that your site just isn't going to work in some older browsers, and move on. This is OK, provided your client/userbase is OK with it.

Normally your development will involve a combination of the above three approaches. The most important thing is that you test each small part before committing it — don't leave all the testing till the end!

Testing/discovery

After each implementation phase, you will need to test the new functionality. To start with, you should make sure there are no general issues with your code that are stopping your feature from working:

1. Test it in a couple of stable browsers on your system, like Firefox, Safari, Chrome, or IE/Edge.
2. Do some low fi accessibility testing, such as trying to use your site with only the keyboard, or using your site via a screen reader to see if it is navigable.
3. Test on a mobile platform, such as Android or iOS.

At this point, fix any problems you find with your new code.

Next, you should try expanding your list of test browsers to a full list of target audience browsers and start concentrating on weeding out cross browser issues (see the next article for more information on [determining your target browsers](#)). For example:

- Try to test the latest change on all the modern desktop browsers you can — including Firefox, Chrome, Opera, IE, Edge, and Safari on desktop (Mac, Windows, and Linux, ideally).
- Test it in common phone and tablet browsers (e.g. iOS Safari on iPhone/iPad, Chrome and Firefox on iPhone/iPad/Android),
- Also do tests in any other browsers you have included inside your target list.

The most low fi option is to just do all the testing you can by yourself (pulling in team mates to help out if you are working in a team). You should try to test it on real physical devices where possible.

If you haven't got the means to test all those different browser, operating system, and device combinations on physical hardware, you can also make use of emulators (emulate a device using software on your desktop computer) and virtual machines (software that allows you to emulate multiple operating system/software combinations on your desktop computer). This is a very popular choice, especially in some circumstances — for example, Windows doesn't let you have multiple versions of Windows installed simultaneously on the same machine, so using multiple virtual machines is often the only option here.

Another option is user groups — using a group of people outside your development team to test your site. This could be a group of friends or family, a group of other employees, a class at a local university, or a professional user testing setup, where people are paid to test out your site and provide results.

Finally, you can get smarter with your testing using auditing or automation tools; this is a sensible choice as your projects get bigger, as doing all this testing by hand can start to take a really long time. You can set up your own testing automation system ([Selenium](#) being the popular app of choice) that could for example load your site in a number of different browsers, and:

- see if a button click causes something to happen successfully (like for example, a map displaying), displaying the results once the tests are completed
- take a screenshot of each, allowing you to see if a layout is consistent across the different browsers.

You can also go further than this, if wished. There are commercial tools available such as [Sauce Labs](#) and [Browser Stack](#) that do this kind of thing for you, without you having to worry about the setup, if you wish to invest some money in your testing. It is also possible to set up an environment that automatically runs tests for you, and then only lets you check in your changes to the central code repository if the tests still pass.

Testing on prerelease browsers

It is often a good idea to test on prerelease versions of browsers; see the following links:

- [Firefox Developer Edition](#)
- [Edge Insider Preview](#)
- [Safari Technology Preview](#)
- [Chrome Canary](#)
- [Opera Developer](#)

This is especially prevalent if you are using very new technologies in your site, and you want to test against the latest implementations, or if you are coming across a bug in the

latest release version of a browser, and you want to see if the browser's developers have fixed the bug in a newer version.

Fixes/iteration

Once you've discovered a bug, you need to try to fix it.

The first thing to do is to narrow down where the bug occurs as much as possible. Get as much information as you can from the person reporting the bug — what platform(s), device(s), browser version(s), etc. Try it on similar configurations (e.g. the same browser version on different desktop platforms, or a few different versions of the same browser on the same platform) to see how widely the bug persists.

It might not be your fault — if a bug exists in a browser, then hopefully the vendor will rapidly fix it. It might have already been fixed — for example if a bug is present in Firefox release 49, but it is no longer there in Firefox Nightly (version 52), then they have fixed it. If it is not fixed, then you may want to file a bug (see [Reporting bugs](#), below).

If it is your fault, you need to fix it! Finding out the cause of the bug involves the same strategy as any web development bug (again, see [Debugging HTML](#), [Debugging CSS](#), and [What went wrong? Troubleshooting JavaScript](#)). Once you've discovered what is causing your bug, you need to decide how to work around it in the particular browser it is causing problems in — you can't just change the problem code outright, as this may break the code in other browsers. The general approach is usually to fork the code in some way, for example use JavaScript feature detection code to detect situations in which a problem feature doesn't work, and run some different code in those cases that does work.

Once a fix has been made, you'll want to repeat your testing process to make sure your fix is working OK, and hasn't caused the site to break in other places or in other browsers.

Reporting bugs

Just to reiterate on what was said above, if you discover bugs in browsers, you should report them:

- [Firefox Bugzilla](#)
- [EdgeHTML issue tracker](#)
- [Safari](#)
- [Chrome](#)

- [🔗 Opera](#)
-

Summary

This article should have given you a high-level understanding of the most important concepts you need to know about cross browser testing. Armed with this knowledge, you are now ready to move on and start learning about Cross browser testing strategies.

[↑ Overview: Cross browser testing](#)

[Next →](#)

In this module

- Introduction to cross browser testing
 - Strategies for carrying out testing
 - Handling common HTML and CSS problems
 - Handling common JavaScript problems
 - Handling common accessibility problems
 - Implementing feature detection
 - Introduction to automated testing
 - Setting up your own test automation environment
-