Web Testing and Selenium:

The Current State & Future Possibilities

TestComplete

TestComplete

TestComplete is a Single Tool for All Your Test Automation Needs

LEARN MORE ABOUT **TESTCOMPLETE**

Content

Web is undergoing dramatic changes which continue to alter the way automated tests are created and maintained. An example of the intricate nature of web is the increased usage of Web Components (Templates, Shadow DOM, and Custom Elements), which are often leveraged while building sophisticated UI. Add to that, browsers becoming increasingly dependent on JavaScript and other open web technologies — which often result in automated tests being slow, brittle, and hard to maintain.

Over the last year, we have also seen browser vendors placing a lot of emphasis on making web secure, stable, and power-efficient. And in that process, they have made it difficult for testers to bypass UI by restricting access through plug-ins and APIs.

With all these changes taking place, Selenium has become the default standard for web testing, as evidenced by a 300% increase in job postings over the past 3 years. It has come a long way since its inception with Selenium IDE (Integrated Development Environment) and Selenium 1 (Selenium Remote Control). The future of Selenium looks different than it did a year back.

The question then arises: what do all these trends mean for web testing and how can testers equip themselves to better handle these changes? Also, it is crucial to understand what the future of Selenium looks like. That's why we created this eBook.

In this eBook, we will look at:

- **Web Testing Trends in 2016:** What has happened in Web over the last year and how do these changes impact automated testing?
- **How Selenium is Evolving to Address These Challenges:** The numerous ways in which Selenium is accommodating to these trends.
- **Challenges with Selenium and Essential Solutions:** The challenges traditionally associated with using Selenium and different ways in which QA teams can overcome these challenges.
- **Tools for Scaling Your Selenium Tests:** How integrations with automated testing and test management tools help scale your Selenium tests?

Web Testing Trends in 2016

Web is undergoing dramatic changes which continue to alter the way automated tests need to created and maintained.

Let's start by looking at three of the biggest trends happening in Web in 2016:

1. Quality is becoming more important, leading to increased complexity for automated tests

One of the biggest trends we saw in 2015 was that ensuring high quality across multiple browsers became more complex for QA testers. This was primarily because of the fact that browser vendors such as Chrome, Firefox, etc. became increasingly locked down. A few examples of browser vendors placing a renewed focus on high quality by becoming locked down include:

Browsers block APIs: In an effort to improve security, many browsers began limiting access to specific APIs in 2015. One of the APIs which browsers started restricting is NPAPI API. In fact, if you were someone with an automation tool using NPAPI API to access your browser, your tests will have started breaking because of this change.

Browsers block technologies: On a similiar note, browsers also began limiting access to certain technologies that proved to be a security challenge. For example, one of the technologies which got blocked completely over the past year was Flash. Organizations, such as Google, took similiar steps when it blocked Flash Ads, and we saw Microsoft and Mozilla following the same trend.

Lengthening plugin approval processes: In order to improve quality and security, the approval process for plug-ins also got longer, and increasingly stringent. For example, with Firefox 42, developers are required to submit extensions for review and signing to Mozilla prior to development. More importantly, if you have any unsigned add-ons, these cannot be installed or used with Firefox.

2. Web continues to evolve

At the same time, there are many things happening which make web more challenging and cumbersome to test.

Influence of mobile on testing: It's no secret that the increased adoption of mobile is having a significant impact on all areas of the software world. This is especially true for teams responsible for testing any software. For example, since mobile-friendliness is a factor for website rankings, testers primarily responsible for testing desktop are now being asked to ensure high quality on mobile web as well. This means testers need to learn new skills specifically for testing mobile apps.

Open web standards: The way app development and web development is shaping up, developers are moving towards the use the same source code for web and app development. As a result, there is a move from pure HTML5 or JavaScript to a combined JSX, or even Facebook's React Native engine, just because these technologies allow developers to use the same source code across web and mobile apps. The use of these technologies means testers come under more pressure to test across devices in a reduced timeline.

Growing use of JavaScript: Browsers are increasingly becoming dependent on JavaScript. This is critical to understand because each of the browsers render JavaScript through runtimes in a different fashion. Since runtimes are different, ensuring test cases are working properly on each and every browser can become challenging.

3. UI gets increasingly sophisticated

While all these changes are taking place, UI continues to get much more sophisticated, and as a result, much more complex for testers.

Web components – move from DOM to Shadow DOM: An example of Shadow DOM could be when you embed a video on your website. In this case, you would have the source ID with your video ID, but when you are rendering the video on your website, each browser already has an embedded code in order to make the code work. For example, in the case of browsers this could be the play button, volume button, pause button, etc and the code for these buttons is primarily embedded in Shadow DOM. As you can see, Shadow DOM can be particularly useful while encapsulating the HTML mark up, CSS styles, and Javascript. It helps developers to decide what is rendered and what is not. However, it causes problems for testers, as accessing shadow DOM elements can be tough.

The impact of responsive UI

When releasing an application, you now have to test it on desktop and web, as well as mobile. Responsive UI makes things increasingly complicated for testers, because each and every mobile device has a different screen resolution and operating specs. The way pages render can also vary on different devices and browsers. One example of this is Canvas, which is part of HTML5 and is an element that's used to draw graphics on a web page. Canvas is powerful and offers new functionality, but it's hard to test, just like with Shadow DOM, with your traditional automation tactics.

CSS implementations vary depending on the browser and how you have structured your website. Say, for instance, you have to test in Internet Explorer 8, and you're using a CSS pseudo selector. While this would work in most modern browsers, in IE8 you could face challenges when testing your website. As a result, the implementations vary and that offers different challenges and different constraints depending on which browsers you care about.

The Evolution of Selenium: How Selenium is Adjusting to Current Trends

The big question remains for modern testers: how is Selenium accommodating to these different trends and challenges? In this section, we'll walk through each of these challenges as well as additional challenges facing testers today.

Mobile

One of the ways Selenium is evolving to meet the demands of mobile web and native apps is through its support of Appium, for both iOS and Android mobile, web and hybrid application. If you are someone who is testing Android native and hybrid applications, there is Selendroid, an automation framework which drives off the UI of Android native and hybrid applications and mobile web. For iOS native web and hybrid apps, there is iOS driver

Responsive UI

There are numerous open source and some commercial solutions that work with Selenium WebDriver, out of the box. These solutions help deal with the challenges of visual anomalies and responsive UI. These solutions can be added to your existing automation process to ensure that things look correct in the correct resolution, on the correct device, in the correct direct response size.

Here's how these solutions work:

A lot of these libraries can be added to your existing test suites, giving you the ability to do an image comparison. This visual testing library will take a baseline image of your application and then the next time it runs, it will look for anomalies based on that.

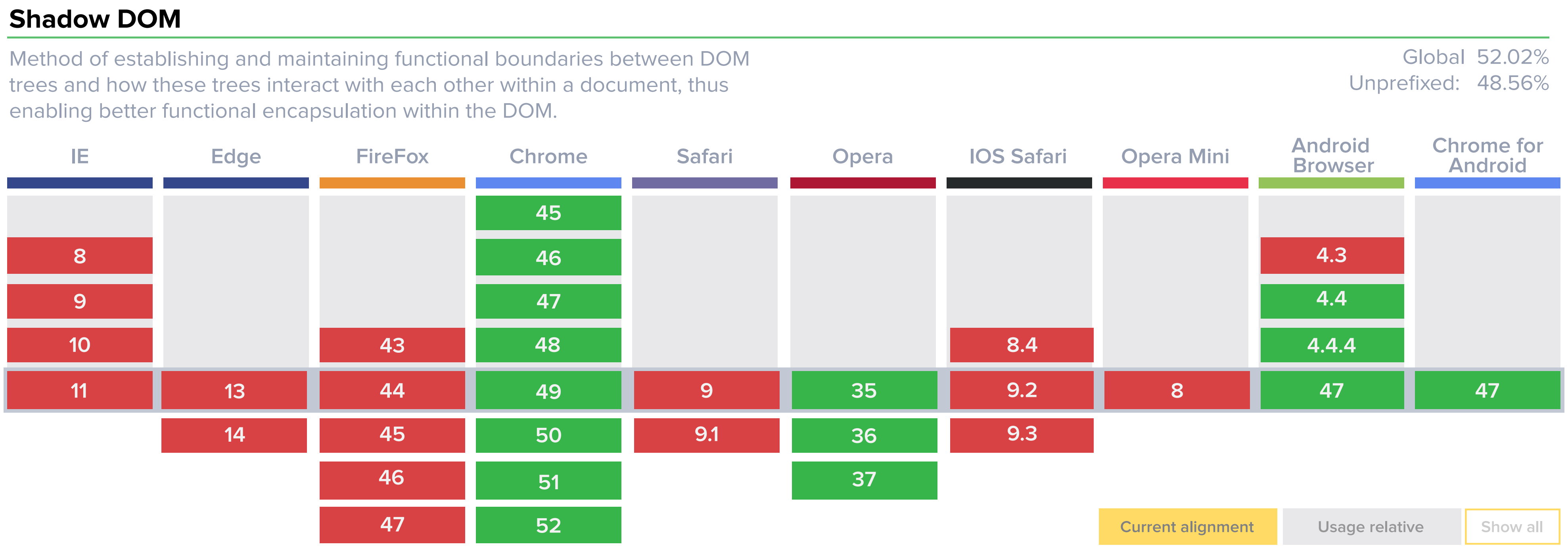
These libraries allow you to create a baseline of different checkpoints of your responsive UI. These checkpoints can come in handy when catching bugs, especially when you're changing the layout and want to ensure that it's accurate to the human eye.

DOM to Shadow DOM

There's no official support for Shadow DOM in Selenium yet, and it's not for lack of trying. The core team has set down several times to hash out an implementation for the API for Selenium. One of the biggest roadblocks to consider is that the W3C specification for Shadow DOM is not complete.

Shadow DOM is supported in all versions of Chrome and Opera. It is not supported in IE, Edge, Firefox, or Safari. The support for shadow DOM across browsers looks something like this:

If you're using an automated testing tool, like TestComplete, Shadow DOM shouldn't be a testing issue. A tool like TestComplete interacts at the operating system level, rather than the web interface level. What that means is that TestComplete has the right access levels. The benefit is that your team is not reverse engineering and injecting JavaScript into the DOM and then making it dance. An automated testing tool like TestComplete is actually getting the full browser experience, and getting the JavaScript rendering for that browser. This enables you to use the same test across different browsers, irrespective of whether you are accessing DOM or Shadow DOM.



Shadow DOM

Method of establishing and maintaining functional boundaries between DOM trees and how these trees interact with each other within a document, thus enabling better functional encapsulation within the DOM.

Global 52.02%
Unprefixed: 48.56%

IE	Edge	FireFox	Chrome	Safari	Opera	IOS Safari	Opera Mini	Android Browser	Chrome for Android
			🟩 45						
🟥 8			🟩 46					🟥 4.3	
🟥 9			🟩 47					🟩 4.4	
🟥 10		🟥 43	🟩 48			🟥 8.4		🟩 4.4.4	
🟥 11	🟥 13	🟥 44	🟩 49	🟥 9	🟩 35	🟥 9.2	🟥 8	🟩 47	🟩 47
	🟥 14	🟥 45	🟩 50	🟥 9.1	🟩 36	🟥 9.3			
		🟥 46	🟩 51		🟩 37				
		🟥 47	🟩 52						

Future Challenges and Solutions: 2016 and Beyond

We have looked at the past and current trends and challenges, and how Selenium can accommodate each. But there are also trends on the horizon that could present challenges for testers in the years ahead. These are not the burning things which testers encounter on a day-to-day basis. But they will have a greater impact in the future.

Challenge: Handling dynamic objects

Let's assume you have a user ID field which has a username as Nikhil123. What happens when you record your test is that this field is automatically filled when the webpage is rendered. The problem is that when the webpage is rendered, the user ID field might be different. Rather than Nikhil123, it might be Nikhil1234. Obviously the test will start failing because the user field or the user ID has changed.

Solution:

The first question to ask is: is there another way to write this test, to interact with this element that does not require a dynamic locator? Ideally, there's some other locator that has semantic markup that's not likely to change and is helpfully named. If it's not on that specific thing you're trying to interact with, it will hopefully be the parent element for that object. You can scope to that, and then walk into it.

If that's not readily available, you could use something like a CSS pseudo-selector, which enables you to walk through the hierarchy of the DOM to scope to specific elements.

As Dave Haeffner of Elemental Selenium explains:

CSS Pseudo-classes work by walking through the structure of an object and targeting a specific part of it based on a relative number (e.g. the third <td> cell from a row in the table body). This works well with tables since we can grab all instances of a target (e.g. the third <td> cell from each <tr> in the table body) and use it in our test — which would give us all of the data for the third column.

The final option is for testers to work with dev to add a unique locator. The key to success here is making developers understand the specific use case. Typically, it's a fairly trivial thing for them to add once they understand what you're trying to do. Once that happens and you have more of those conversations over time, and you give more examples to the development team, they can start to see, "Oh, if I actually thought about this when building it, this would provide a shortcut for testing."

Challenge: Dealing with pop-up windows

There are two types of pop-up windows — web-based and window-based. Let's see how they differ:

You are on a website — let's use SmartBear.com as an example — and you click to download a tool, like TestComplete. Before you download the tool, you want to print the overview page, so that you can show it to your manager. If you right click and do the print command, that command is actually coming from the window, rather than the web.

At the same time, there might be a web-based pop-up, which is basically an alert button which shows you whether you want to go ahead with a particular action, or whether you want to cancel a particular action. As another example, let's say you are on Amazon and you want to buy something. When you want to check out, they ask you whether this is the item you want to buy. That pop-up is coming from the web, rather than the window.

How can handle both of these pop-ups through Selenium, and which can be done well with Selenium?

Solution:

There is a limit to the things that Selenium can interact with, and since it's for driving the web browser, there are things outside of Selenium's control. Operation system-level pop-ups like print dialogs, file download, file upload dialogs can be a challenge for Selenium. However, handling browser-level popups are easy with Selenium.

Challenge: Multi-tab testing

The third part of the testing that is encountered often is multi-tab testing. Again, if we're testing SmartBear.com and want to open a separate tab to look at the TestComplete product page, you're not opening in the same window. Instead, you're opening it in the adjacent window. Again, it's important to think about the experience of the user. Users will right click and then open the tab in the adjacent window. How do you replicate that scenario when you go back to the product page when writing the test case through Selenium?

Solution:

If you truly need to test new tabs, the best you can do is issue keyboard keys to control tab (Ctrl + tab) to the next tab.

As Dave Haeffner explains on his website:

This can be done to a specific element, or generically with Selenium's Action Builder. Either approach will send a key press. The latter will send it to the element that's currently in focus in the browser (so you don't have to specify a locator along with it), whereas the prior approach will send the key press directly to the element found.

You can also write your tests so that they interact with new windows. If you open a link in a new window, Selenium knows that there are multiple windows for that session, so you can actually switch between them, and then once you do switch, then you can interact with that page. However, as evident, you have to switch back, if you want to go back to the previous window and test it.

Challenge: Slow, brittle, hard to maintain tests

Slow, brittle, and hard to maintain — these are the most common words that are associated with Selenium. The fact is that with Selenium IDE record and playback tools it's very easy to paint yourself into a corner — to end up with slow, brittle, and hard to maintain tests. That's a real challenge associated with Selenium unless you know how to mitigate it.

Solution:

The key to ensuring that your tests are easy to maintain is to follow a modular testing practice. For example, if you're building a webpage test — again, let's say, Amazon.com — the modular test design would be to look at the login button, log out button, and product recommendations as one module. By building tests with module design, you're able to reuse the modules in different sets or different test cases.

The other way to build a modular test design would be to not restrict it to the test cases, but actually building modular test data, as well as using it for test environments. For example, you can build modular test environments, depending on the browsers or operating system. We recently did a webinar on this topic which can be accessed here.

Challenge: **Scaling tests**

Once you figure out how to write tests that are easy to maintain and are actually performing well, how do you grow your testing practice so that you can cover potentially hundreds, or thousands, of use cases?

Solution:

The open source option for scaling your test infrastructure is using something like Selenium Grid, which is a built-in option within Selenium, which can spin up a complete network of machines that you coordinate and control test execution on. There are also third party solutions that offer Selenium Grid as a service and point at their cloud endpoint to run tests. There are commercial options, which include additional functionality that isn't available with an open source solutions. TestComplete's TestExecute gives testers the ability to execute Selenium tests, in a test lab, virtual machines, or even cloud.

TestComplete's integration with Selenium also enables teams to:

- Debug failed Selenium scripts quickly using screenshots.
- Get rapid feedback on product builds and save time by using out of the box plugins for CI systems.
- Use in-depth reports to get more insights into code coverage and quality
- Spend less time creating and maintaining Selenium scripts by reusing existing Selenium tests

Challenge: **Reporting**

Reporting can be a real challenge, because out-of-the-box, Selenium offers limited reporting. This is a challenge for testers, as ideally the goal of testing is to be able to find an issue and show it to the necessary people so they can understand where something failed.

Solution:

When it comes to reporting, there are two things to consider — human readable and machine readable reporting. For machine readability with something like your continuous integration server, you want metrics to check the health of a test job or tests over time as well as getting stacked trace information out of your test failures.

When it comes to *human readability*, you want to have something like an HTML report that has screenshots, videos, and additional visibility. Up until recently, this was a real challenge, something you might have had to create yourself. There is now an open source library, called the Allure framework, which is an HTML report generator for Selenium and that is language agnostic. It works for every primary programming language and major test framework. The Allure framework consumes data XML and screenshots then renders an HTML mini-app that's built using Angular.

Additionally, TestComplete offers a number of reporting features. TestComplete also generates detailed logs along with snapshots of all actions performed during automated testing. This in turn helps testers perform deep analysis of automated test results and quickly locate and fix errors. You can try TestComplete here.

Additionally, QAComplete can also be used to get detailed reports on Selenium tests. It allows you to easily link Selenium tests with user stories and associated defects. As a result, you can leverage end-to-end traceability reports for one view of your testing efforts. Full two-way integration with Jira and other tools helps you get reporting data from other systems as well. Try QAComplete here.

Challenge: Test Management

Even if you're writing automated tests with Selenium, you're traditionally also doing some manual testing as well, just to ensure that you're covering the areas which cannot be tested through automation. Additionally, you might be using API tests to test the backend. The question then arises: How do you manage these tests? How do you get end-to-end traceability of tests with requirements and defects? Most importantly, how do you decide when to ship the product? And finally, how do you know what your test coverage look like?

Solution:

If you're using Selenium for testing the front-end of a mobile app from a website, and then on the back-end you might be testing through an open source solution like SoapUI. In such a case, you might want to consider investing in a solution for test management. With a tool like QAComplete, you'll get real-time insights into your testing state through built-in traceability across requirements, tests, and defects. This allows you to manage a complex testing process by getting full visibility into test changes and their impact with versioning. You can try QAComplete here.

Challenge: Limited Selenium support

There's no 1-800 number for Selenium. So, when there's an issue, you don't really have someone to call. The best you can do is find people in the community who are interested in helping. It's a great part about the community, but it is a challenge. You have to know where to find these people and this information. Support can be a challenge if you're just getting started.

Solution:

The biggest challenge with Selenium is information. There is a strong community within the Selenium Project. If you're just looking for answers as quickly as you can, you can also use the Selenium IRC chat channel. This is the place where the core practitioners hang out and answer question and ask questions about Selenium. It's the best resource that you can find with regards to Selenium.

SMARTBEAR TestComplete

Overcome your test automation challenges

Whether you're looking to get the most out of your Selenium tests or manage the limitations that come with an open source solution, integrating Selenium with an automated testing tool like TestComplete can help.

TestComplete Web helps you create automated functional tests for websites, web apps, and mobile web applications.

With TestComplete's integration to Selenium WebDriver, developers and testers can:

- Perform end-to-end testing: Combine Selenium with API tests to go beneath the user interface and account for unpredicted changes made to back-end services.
- Fix issues quickly: Screenshots captured during test runs enable you to find and fix failed Selenium WebDriver scripts swiftly.
- Reduce Setup and Maintenance Costs: Prevent hidden costs associated with setting up, maintaining, and scaling Selenium Grid for only $499 a year.
- Scale Selenium Tests: Execute Selenium WebDriver tests from the cloud or virtual machines. Run multiple tests in parallel to reduce testing time.
- Integrate with CI systems: Get rapid feedback on product builds and save time by using out of the box plugins for CI systems.
- Get Actionable Results: In-depth reports along with console logs help you get a single view of different Selenium WebDriver tests run and detect failures quickly.

LEARN MORE ABOUT **TESTCOMPLETE**

Additionally,

Selenium's integration to test management platform QAComplete provides the following:

- Full Visibility into Selenium tests: Get real-time insights into your testing state through built-in traceability across requirements, Selenium tests, and defects. Manage complex testing process by getting full visibility into changes and their impact with versioning.

- End-to-End-View: Plan, track, and proactively manage manual, automated, and API tests in one repository to mitigate risk. Launch with certainty by getting up-to-date information about your open source Selenium and SoapUI tests along with TestComplete and Ready! API tests.

- Make informed decision: Use reports to get better understanding of test coverage, defect trending, and sprint status. Isolate, resolve, and automate the resolution of testing bottlenecks by defining escalation rules and getting instant notifications.

QAComplete

LEARN MORE ABOUT **QACOMPLETE**

SMARTBEAR

TestComplete Platform: Testing for Desktop, Mobile, & Web Applications

TestComplete

TRY IT FOR **FREE**

SMARTBEAR

Over 4 million software professionals and 25,000 organizations across 194 countries use SmartBear tool

4M+	25K+	194
users	organizations	countries

See Some Succesful Customers >>

API READINESS

Functional testing through performance monitoring

SEE API READINESS PRODUCTS

TESTING

Functional testing, performance testing and test management

SEE TESTING PRODUCTS

PERFORMANCE MONITORING

Synthetic monitoring for API, web, mobile, SaaS, and Infrastructure

SEE MONITORING PRODUCTS

CODE COLLABORATION

Peer code and documentation review

SEE COLLABORATION PRODUCTS

TestComplete
Se