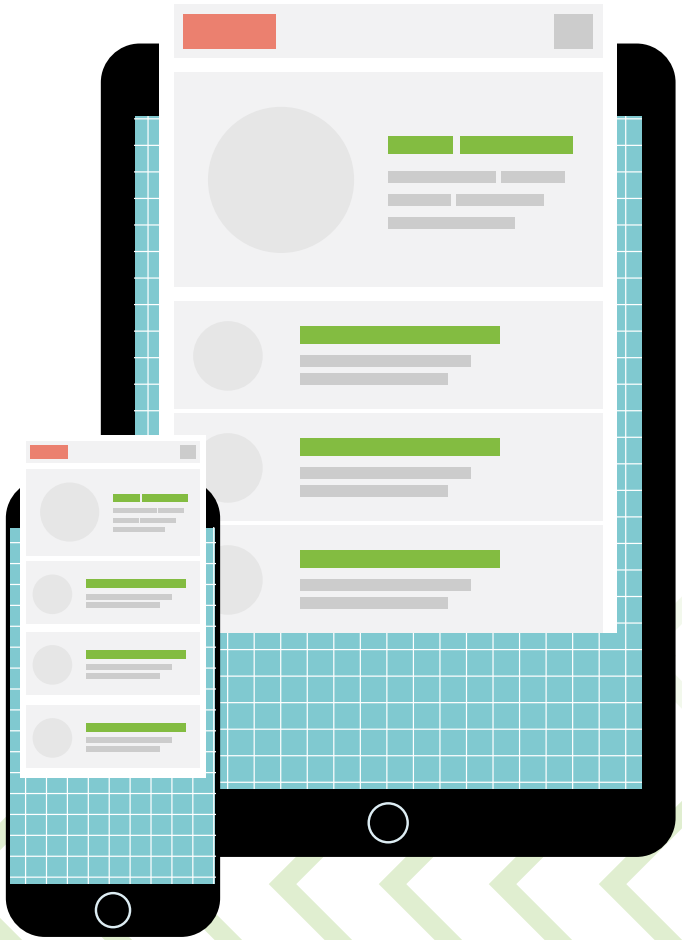
10 Test Automation Frameworks for Cross-Browser Testing

A Comparison Guide for 2017

Perfecto

Seek Perfection

Table of Contents



Introduction

The goal of this document is to offer web development teams a set of selection criteria for choosing the correct web testing frameworks for each project. The paper provides a market overview of cross-browser testing tools, including a summary of the ten leading web testing frameworks available on the market.

Key Findings:

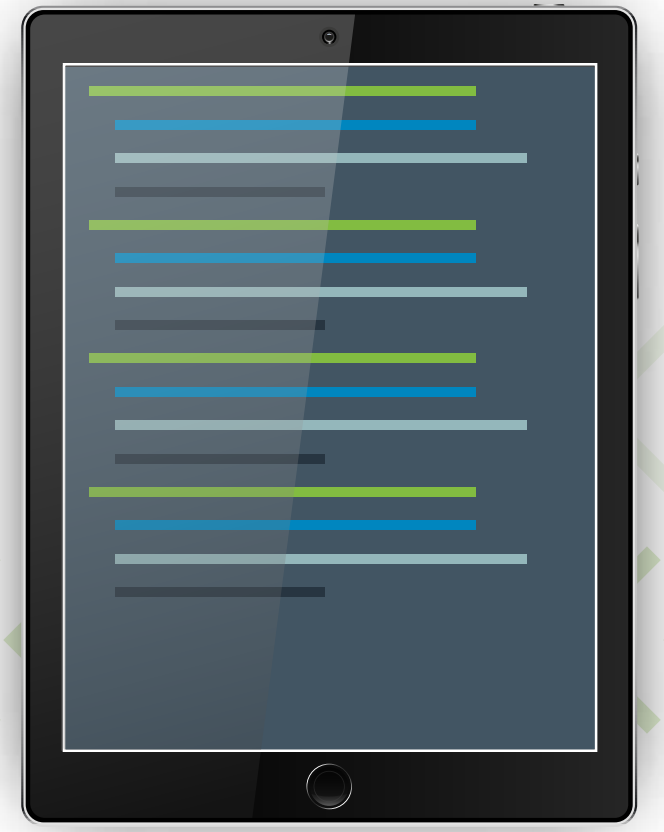
1. **The open-source community is very active and offers great code samples, innovation and support. New tools are introduced constantly and it's worth tracking tool adoption, contributions and features to stay ahead of the market.**
2. **JavaScript is the standard development language in web testing**
3. **Selenium is the core API for the leading web test automation frameworks**
4. **BDD is a clear trend in web test automation and is supported by the majority of tools**
5. **Protractor, WebDriver.IO, and NightwatchJS are the leading E2E web testing frameworks**
6. **For unit testing purposes, PhantomJS, CasperJS and JSDOM are the most common frameworks**
7. **To choose the right test framework, evaluate the six organizational fit and six technical fit criteria.**

JavaScript is The Leading Web Development Language

Before looking at the landscape, it is important to understand that in the desktop web market, JavaScript is the leading development language for both developers and testers. Many leading web development frameworks in the market are leveraging JavaScript. They include frameworks such as AngularJS, React, Aurelia, Vue.JS, and others.

Selenium WebDriver is the market leader in adoption for testing web applications; however, it is being used in conjunction with a number of frameworks designed to boost productivity and it does have gaps with respect to what traditional testing tools provide (e.g., object repository).

Non Selenium WebDriver frameworks are out of scope.



Market Overview

Selenium and Protractor are seeing the largest number of downloads (Fig. 1). However, there are other parameters and solutions to consider as part of your tool selection. Sometimes the best solution is to build a tool stack strategy that includes a combination of frameworks, each serving a different purpose.

The tools that you select for your testing activities should support specific practices such as Behavior Driven Development (BDD), or back-end services, or other JavaScript frameworks (such as Node.JS). What we see in the market is that various tools best fit a specific practice, such as BDD, which can be technically supported through different capabilities. The goal is choosing tools matching both organizational and technical criteria.

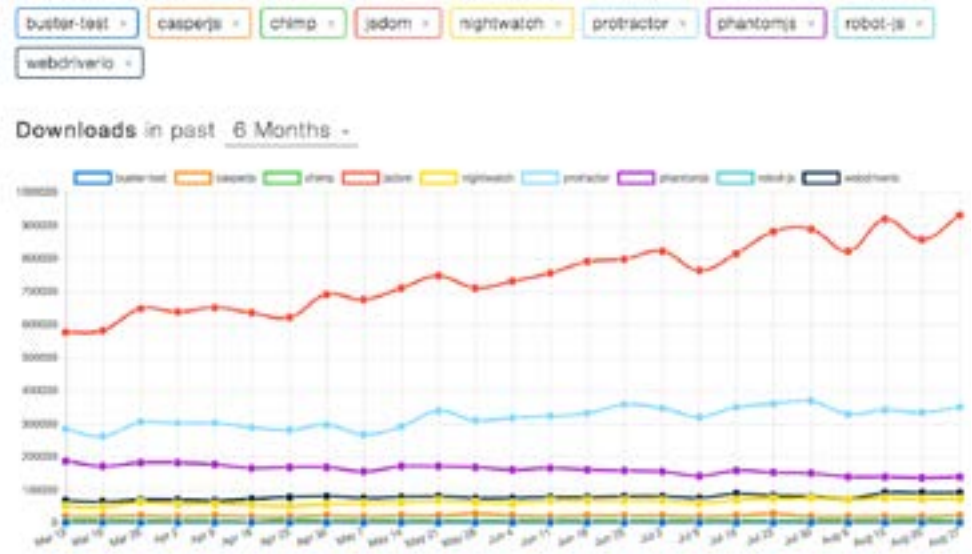
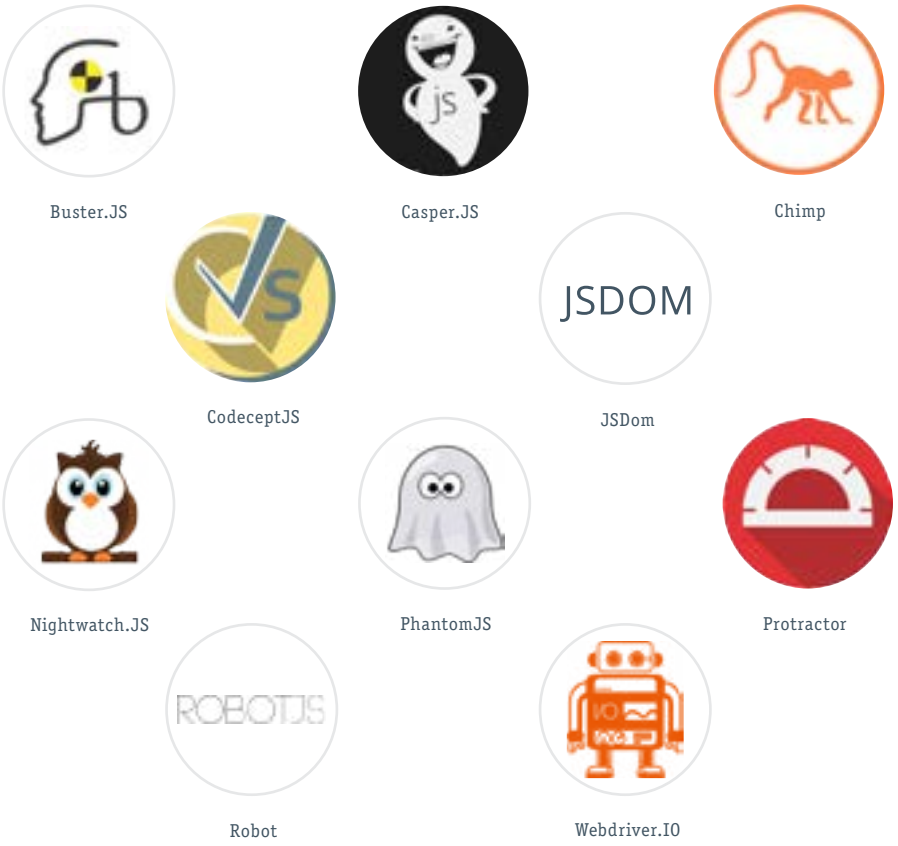


Figure 1: Cross-browser testing tools download stats (Source: NPM Trends)

Here are the top 10 test frameworks (in no particular order). There are at least a dozen more frameworks outside of these top ten. **We include Selenium in this list** but do not compare it to the rest, simply because Selenium is the basis for most of these solutions.

As can be seen in Figure 1, Protractor, from an E2E testing perspective, has the largest number of downloads, while JSDOM, from a unit testing perspective, is the most downloaded. These figures alone are not sufficient to drive a concrete decision but they can help to understand trends in the market. As we will see in the criteria section, a good decision needs to be based on a mixture of technical considerations and organizational fit.



> Often, teams find greatest benefit using a set of test frameworks rather than using only one.

JavaScript is King

JavaScript is, by far, the leading dev language for web. Most web development frameworks are based on JavaScript; it's not surprising that the associated test tools are also written in and support this language.

In a recently conducted survey, JavaScript was leading the preferred programming language of 62% of respondents.

Among the key benefits of using JavaScript for testing, these are the clearest:

1. Open-source is free

2. It's modular

3. It's backed by an active and vibrant community

4. Your client-server is written in JavaScript why not the tests?

a. This assumes that the test engineer has proper test automation skills

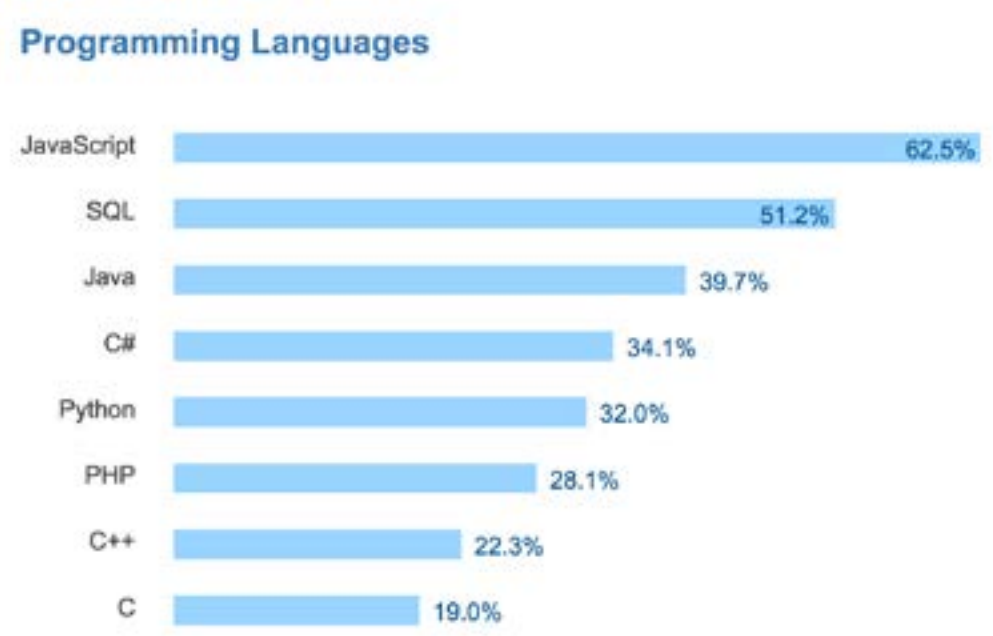


Figure 2: Preferred programing language survey (Source: Stack Overflow)

How to Select the Correct Test Framework for Your Needs?

When choosing a testing framework, consider both current organizational and technical fit.

The following are 6 organizational fit considerations and 6 technical-fit-related considerations to help make the best choice for your project

Organizational Fit

As part of the framework selection process, each team needs to assess the following 6 considerations and bake them in to the overall decision. The overall SDLC of a project is driven by existing and defined practices (BDD, TDD, waterfall etc.) and skills; therefore, the tool that is selected should suit them.

1. Project Complexity

- How **complex is the website** under test? The more complex the tool is, the more dependent your test automation architect will be upon APIs and extensions to be able to cover all functionalities and to manage test assets.

2. Resources

- Do you have within the team the right **skills** and **resources** to accomplish your objectives? Do you need to hire and train new resources for the selected framework?
- What are the **budget constraints**? Even though most web testing frameworks are open-source and free, people aren't; these tools test against real platforms like desktop browsers and mobile devices, and setting up a lab requires budget space.
- What is the **time frame for the project** and its release cadence? This will impact tool selection due to fast ramp-up, available support and documentation, and execution speed (e.g. going with headless browser vs. real browser, etc.) Also, how is reusability being leveraged?

3. Existing Tech Stack (Devops, COE, etc.)

- What kind of **development framework** is my web site built on? What other tools does the web site integrate with? What **current CI tools and test case/defect management** tools are being used within R&D?

4. Test Environment

- What are the **testing environment** requirements? Do you need to cover performance metrics? Do you need to capture network traffic data per execution? Do you need to mimic different locations, etc?
- What are **the platform coverage** requirements for the project? How many browsers? Is mobile also part of the mix?

5. Test Types / Practices

- What are the **quality criteria** for the project, e.g., which testing types are required for the project? This will impact the decision-making for the various tools (plugins, integrations)
- How good is the tool at building a robust **object repository** and identifying all object locators (Angular objects, React etc.)?
- Is the tool able to support things like visual validation and testing of responsive web?
- Which **execution engine** and test runners are supported with the test framework?

6. Reporting

- What kinds of **reports and dashboards** should be provided to developers, testers, and management?

Technical Fit

When the organizational fit items are well defined, it is time to match them against the relevant frameworks and assess their technical fit to the teams involved. Below, we define 6 technical fit considerations.

1. Community Size, Support and Documentation

The tool should be well documented, have an active community for resolving bugs, and commercial support.

While there are various frameworks in the open-source community, not all of them are equal with respect to level of support, continuous contribution, documentation, and commercial support – can you call someone when something doesn't work? The number of contributors, their skills and availability to support are important criteria, as well as how recently they contributed new code to the project. We have seen that frameworks such as Protractor and WebDriver.IO have, by far, a larger number of contributors, branches and support than tools such as Nightwatch.JS and Chimp; however, as part of your decision making, you need to look closely at what kinds of contributions have been made and when they were made.

2. SDLC Process Fit (Integration, Plugins, Dev Methodology fit)

The tool should support the SDLC practices for your web project, whether BDD or other methods are utilized. In addition, it should be easily extended and integrate into various other tools (CI, IDE's, Reporting, Defect Management, etc.)

What we've learned during tool research is that most tools have some level of BDD support. Some have better support than others. There are tools which allow running BDD-based test automation as part of their integration, while others implement their own unique BDD syntax. In addition, a tool should play nicely in the overall DevOps code pipeline. Whether the team uses CircleCI, Jenkins, or another continuous integration server, the test frameworks should support it seamlessly. Also, having the ability to extend the framework and build more capabilities on top of it through supported APIs is another consideration for teams when selecting a framework. Lastly, test automation is an ongoing project and, as with code, it requires continuous support; therefore, it is important to build the foundation on a framework that can grow with your project and add more capabilities as your project evolves.

3. Feedback Loop and Reporting

The tools should be able to provide actionable reports with sufficient artifacts for fast resolution of defects.

Dealing with a large amount of test data across many platforms is a challenge for most of the testing tools we evaluated. Some of these tools offer plugins to support robust reporters, such as Allure and others, and are also integrated into a variety of test management systems. Being able to get fast quality analysis and reports through the test framework is a critical requirement for fast release cadence, test management, data driven decision making, and efficiency.

4. Cloud and Automation at Scale

The tool should be able to support automation at scale across browsers and mobile devices through integrations with cloud providers.

Most web projects today are not only about desktop browsers. End users consume web content from multiple screens, locations, and in varying conditions. This reality requires a lab scaleable to support that can provide on-demand access to the latest browser version, beta, legacy web, mobile platforms, and various OS versions. These targets should be easily tested in parallel through CI, execution engines such as TestNG, or another grid configuration. Some of the frameworks that were evaluated have OOB integrations to cloud providers, such as Perfecto and others, and can be used to address these criteria.

5. Automation Coverage

The tool should provide enhanced test automation capabilities for your web site, including network monitoring, memory and performance profiling, visual navigation testing, and others.

As part of your test automation considerations, you need to make sure that the selected tool or tools support your JavaScript framework and can support additional types of testing other than functional test automation: tests such as performance, responsive design validations, internationalization and localization, and more. In many cases, commercial tools that wrap the underlying open-source framework can complement it with these additional capabilities which will also include future enhancements to the framework.

As a nice example of how to leverage from the browser vendors' built-in capabilities, Figure 3 shows how Google implemented extended testing capabilities within its browser which can also be used through WebDriver for your automation scripts. Such examples can add network monitoring, performance and code coverage capabilities to your testing.

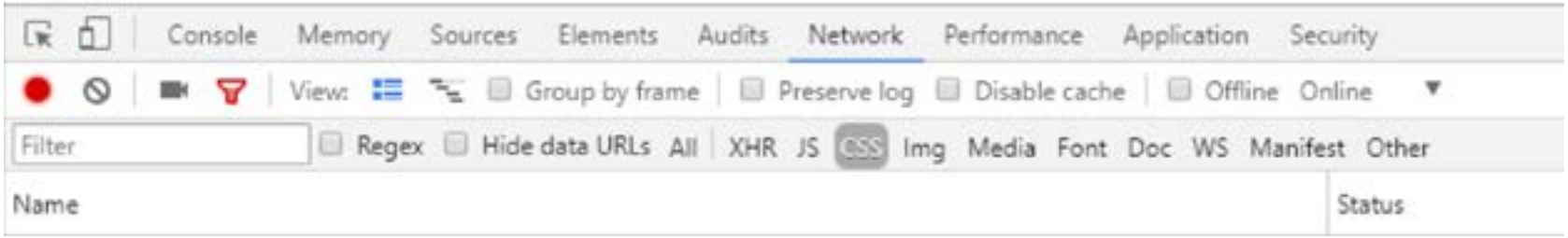


Figure 3: Google Chrome Dev Tools Options

6. Automation Robustness and Maintainability

The tool should support automation best practices such as page object model (POM), sync testing, and be easy to set up and maintain.

SYNC-based (e.g. Protractor, Codecept) tools that have an object repository allow for faster maintenance and easier to read scripts; in addition, having a synchronized test suite means that, from a testing perspective, you don't need to care about callbacks, or promises — no “wait states” make for more robust tests. A synchronized framework syncs the test steps with your application so that they will be executed properly, and at the right time, only when the application is ready and has processed the previous step. There is less overhead associated with waits, elements not found, and broken CI/automation builds when having this capability.

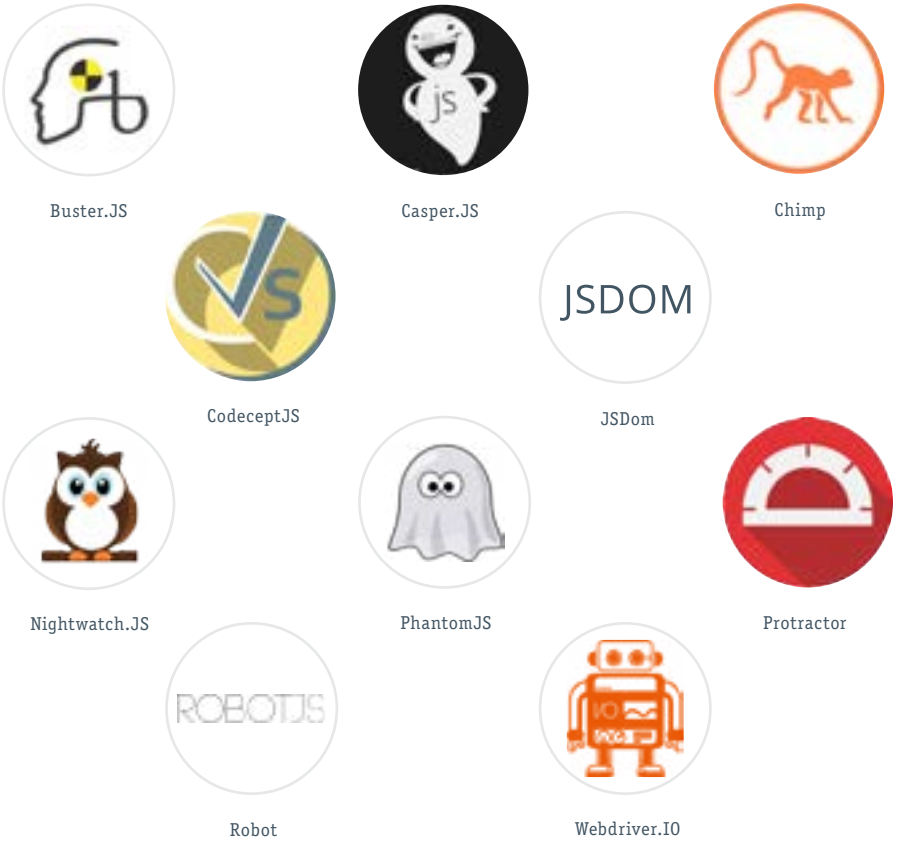
This does not mean that unsynchronized frameworks won’t deliver the same outcome, however; for that to happen, teams will need to implement better mechanisms based on implicit/explicit waits, assertions, and more.

From a maintenance perspective, tools that provide an automated wizard to quickly setup your environment, generate a config.js file for you, and more, obviously save time in the overall test automation development process, and reduce manual configuration errors. Lastly, tools that support different approaches — data-driven, keyword-driven, etc. — simplify the testing

Test Framework Summaries

Once both the organizational fit items and the technical fit are clearly set, let's examine these 10 frameworks at a high level and look at how they stack up against each other from a technical standpoint.

The table below represents the current state; however, with the market moving fast, keep in mind that even if your team has built a working strategy for their web product, it is always recommended to revise and validate that it is still the best choice considering the progress of the various communities and commercial tools.



CATEGORIES

		 Casper.JS	ROBOTJS Robot	 Buster.JS	 Nightwatch.JS	 WebDriver.IO	 Protractor	 Codecept.JS	 Phantom.JS	JSDOM JSDom	 Chimp
Automation Coverage	Visual navigation testing	No	No	No	With external libraries	Using external tools	Using external tools	No	No	No	No
	Take screenshots	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
	Network monitoring, Har File	No	No	No	Yes (through Google Chrome Driver)	Yes (through Google Chrome Driver)	Yes (through Google Chrome Driver)	Yes (through Google Chrome Driver)	Yes	No	Yes (through Google Chrome Driver)
	Memory and Performance Profiling	No	No	No	Yes (through Google Chrome Driver)	Yes (through Google Chrome Driver)	Yes (through Google Chrome Driver)	Yes (through Google Chrome Driver),	No	No	Yes (through Google Chrome Driver)
	Code Coverage Analysis	No	No	No	Jacoco	Jacoco	Jacoco	Jacoco	No	No	Jacoco
	Testing Types	Unit testing/Headless	E2E/Acceptance	Unit Testing	E2E/Acceptance	E2E/Acceptance	E2E/Acceptance	E2E/Acceptance	Unit, Headless	Unit, Headless	E2E/Acceptance
SDLC Process Fit	Plugins/Integrations	Jira, Junit, Cucumber, Gradle, Maven, Ant	Intellij, Maven, Jenkins, Other	eMacs, TextMate, xUnit	Cucumber, Intellij, Grunt, Jenkins, TeamCity, Hudson, Junit XML reporting built-in	Community Plugins available	Various community plugins, Jenkins CI, Jasmine, Mocha, Cucumber, Visual Studio, WebStorm, Grunt	Jasmine, Qunit, Travis CI, Jenkins, TeamCity, Buster.JS is built-in. This FW uses QtWebKit	Mocha for test execution	Karma	Mocha, Jasmine, Cucumber, Meteor, Most CI solutions, Simian, Chai, WebDriver.IO
	BDD/ATDD Friendly	Yes	Yes, KDT	Yes	Yes (Jasmine, Mocha, Cucumber)	Yes (Jasmine, Mocha, Cucumber)	Yes (Support Jasmine, Mocha, Cucumber)	Mocha JavaScript DSL language	No, Phantom tests can be triggered from BDD frameworks	No	Yes
	Dev Language Support	JavaScript	Java, Python	JavaScript	JavaScript	JavaScript	JavaScript, TypeScript	JavaScript	JavaScript	JavaScript	JavaScript
Automation Robustness and Maintainability	Config File Generation	No	No	No	No	Yes	No	No	No	No	No
	Page Object Model Creation	No	No	No	No	Yes	Yes	Yes	NA	Yes	No
	Execution Speed	Fast	Slower than headless	Slower than headless	Slower than headless	Slower than headless	Slower than headless	Slower than headless	Fast	Fast	Slower than headless
	Sync	No	No	No	No	Yes	Yes	Yes	No	No	Yes (via wrappers)
Feedback Loop and Reporting	Reporters	CMD, XUnit-XML	Rebot (XML)	HTML, Built-in reports, API for custom reporter	HTML, Allure plugin, Junit XML	Allure, Junit, HTML, XML, Perfecto	Jasmine2HTML, JUnit, Allure	CLI, XML, HTML	Jasmine reporters, Karma reports	Built in console for reporting	Relies on integrated FW reports like Mocha and others
Community Strength	Documentation	Casper.JS	Robot	Buster.JS	Nightwatch.JS	WebDriver.IO	Protractor	Codecept.JS	Phantom.JS	JSDom	Chimp
	Contributors	176 contributors	50 Contributors	20+ Contributors (not that active)	59 contributors	250 contributors	234 contributors	81 contributors	147 contributors	202 contributors	40 Contributors
Cloud and Automation at Scale	Cloud Support	No	Yes (web and mobile)	No	Yes (web and mobile)	Yes (web and mobile)	Yes (web and mobile)	Yes (web and mobile)	No	No	Yes

Casper.JS is A navigation scripting & testing utility for PhantomJS and SlimerJS headless browser tools. CasperJS allows you to build full navigation scenarios using high-level functions and a straightforward interface – this makes the solution very appealing and easy to start writing tests for folks who don't have a strong technical background. You can get started with this tool through this URL: http://docs.casperjs.org/en/latest/quickstart.html

Pro's

- **Automation Coverage:** Designed for unit testing activities
- **Automation Robustness:** Fast feedback due to fast execution time
- **Automation Coverage:** Seamlessly works with Phantom and SlimerJS headless browsers
- **SDLC Process Fit:** Easy ramp-up from a skillset perspective, BDD-based scripting

Con's

- **Automation Coverage:** Not the best fit for E2E testing scenarios
- **SDLC Process Fit:** Uses older headless browser technologies, compared to recent Chrome Blink- based headless browser
- **Automation at Scale:** Doesn't cover real desktop browsers OOB, needs to be complemented by an E2E solution

Automation Coverage	
Visual navigation testing	No
Take screenshots	Yes
Network monitoring, Har File	No
Memory and Performance Profiling	No
Code Coverage Analysis	No
Testing Types	Unit testing/Headless
SDLC Process Fit	
Plugins/Integrations	Jira, Junit, Cucumber, Gradle, Maven, Ant
BDD/ATDD Friendly	Yes
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	No
Execution Speed	Fast
Sync	No
Feedback Loop and Reporting	
Reporters	CMD, XUnit-XML
Community Strength	
Documentation	Casper.JS
Contributors	176 contributors
Cloud and Automation at scale	
Cloud Support	No

Robot Framework is a test automation framework for acceptance testing and acceptance test-driven development (ATDD). It has easy-to-use tabular test data syntax and utilizes a keyword-driven testing approach. Its testing capabilities can be extended with test libraries implemented either in Python or Java, and users can create new higher-level keywords from existing ones using the same syntax that is used for creating test cases.

Pro's

- **Automation Coverage:** Designed for acceptance testing activities (ATDD)
- SDLC Process Fit: Leverages keyword-driven testing method making it easy to develop readable scripts
- **Community and Support:** Large community behind it, plenty of plugins and extensions
- **Automation at Scale:** Cross platform support — supports testing of Android, MongoDB and more.

Con's

- **SDLC Process Fit:** Not fully embedded into dev workflows, suitable more to QA
- **SDLC Process Fit:** Mostly based on Python, less JavaScript focused – may be a skillset issue
- **Automation Robustness:** Uses WebDriver as an external test library rather than fully designed around it
- **Feedback Loop and Reporting:** Limited reporting
- **SDLC Process Fit:** KDT is a less common practice than BDD (source)

Automation Coverage	
Visual navigation testing	No
Take screenshots	Yes
Network monitoring, Har File	No
Memory and Performance Profiling	No
Code Coverage Analysis	No
Testing Types	E2E/Acceptance
SDLC Process Fit	
Plugins/Integrations	IntelliJ, Maven, Jenkins, Other
BDD/ATDD Friendly	Yes, KDT
Dev Language Support	Java, Python
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	No
Execution Speed	Slower than headless
Sync	No
Feedback Loop and Reporting	
Reporters	Rebot (XML)
Community Strength	
Documentation	Robot
Contributors	50 Contributors
Cloud and Automation at scale	
Cloud Support	Yes (web and mobile)

Buster.JS is a **Node.JS testing** toolkit that is highly extensible for reporting and other purposes. This toolkit is built in to headless browser tools such as Phantom. JS and JSDom. This tool fits testing of browsers and Node apps.

It is still being positioned as Beta — unclear if this will evolve. That means less community behind it, perhaps technical limitations, no IDE plugins and potentially some integration issues etc.

Buster.JS comes with built-in assertions and DSL support for **adding app-specific custom assertions.**

In addition, Buster.JS **comes with a few front-end plugins such as XUnit, as well as a BDD plugin** that enables automation engineers to write scenarios.

Pro's

- **Automation Coverage:** Supports headless browser testing and unit testing for Node and browsers
- **SDLC Process Fit:** Friendly DSL-based scripting technology that supports assertions
- **SDLC Process Fit:** Extensible and integrates with many CI and reporting tools

Con's

- **Automation Robustness/Community and Support:** Still growing, currently in Beta – might mean less experience, less stability, less functionality
- **Automation Robustness and Maintainability:** Async testing tool
- **SDLC Process Fit:** Requires setting up a proxy server to cover testing against an application server from your web tests
- **Automation Coverage:** Doesn't fit a complete E2E functional testing objective

Automation Coverage	
Visual navigation testing	No
Take screenshots	Yes
Network monitoring, Har File	No
Memory and Performance Profiling	No
Code Coverage Analysis	No
Testing Types	Unit Testing
SDLC Process Fit	
Plugins/Integrations	eMacs, TextMate, xUnit
BDD/ATDD Friendly	Yes
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	No
Execution Speed	Slower than headless
Sync	No
Feedback Loop and Reporting	
Reporters	HTML, Built-in reports, API for custom reporter
Community Strength	
Documentation	Buster.JS
Contributors	20+ Contributors (not that active)
Cloud and Automation at scale	
Cloud Support	No

Nightwatch.js is an easy to use *Node.js*-based End-to-End (E2E) testing solution for browser-based apps and websites. It uses the powerful *W3C WebDriver API* to perform commands and assertions on DOM elements. The tool has a few built-in plugins for Junit XML reporting that make it easy to send steps reports to CI servers such as Jenkins, TeamCity and Hudson, as well as a Grunt plugin for simple task execution. You can execute your tests against a local Selenium server or against a cloud server such as Perfecto and others. This solution is one of the main competitors to Protractor.

Pro's

- **Automation Coverage:** Good for E2E functional testing
- **Feedback Loop and Reporting:** Good reporting plugins such as Allure and JUnit
- **Automation at Scale and Cloud:** Easily integrated into cloud testing solutions (e.g. Perfecto)
  - **Uses Magellan.json for scaled browser automation**
- **SDLC Process Fit:** BDD friendly – comes with a built-in BDD FW based on Chai, and also supports Jasmine, Mocha and Cucumber

Con's

- **Community and Support:** Community of contributors is relatively small compared to Protractor and WebDriver.IO
- **Automation Robustness and Maintainability:** Async testing tool compared to Protractor which is a sync-based framework (built-in waits)
- **SDLC Process Fit:** Complex setup with a lot of pre-requisites
- **SDLC Process Fit:** Nonstandard BDD assertions compared to WebDriver.IO, which supports Jasmine, Mocha etc. assertions that are more common.

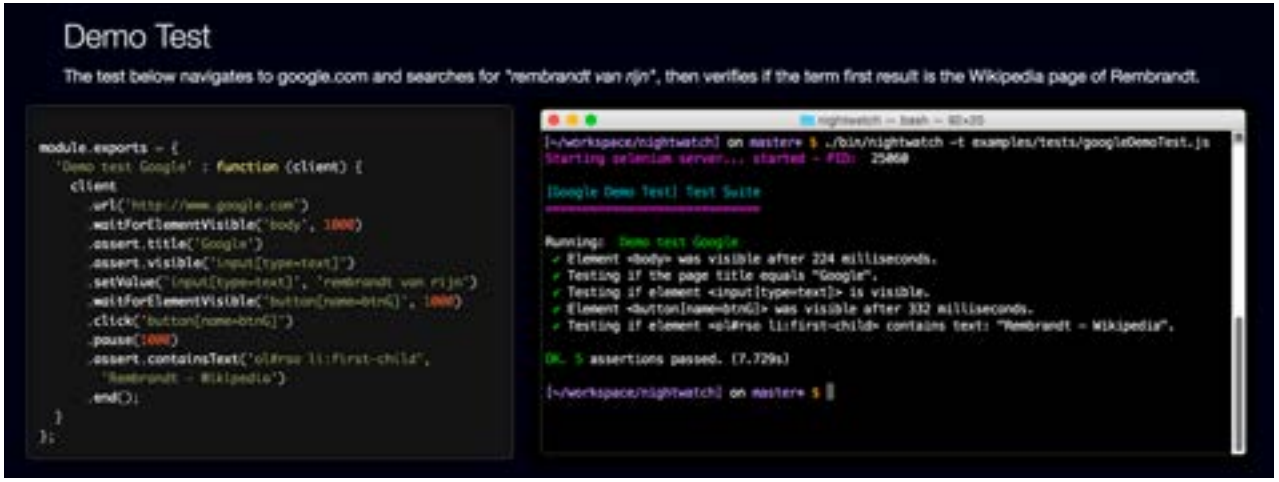


Figure 4: Nightwatch.JS sample code snippet & execution console side by side (source: nightwatch.js)

Automation Coverage	
Visual navigation testing	With external libraries
Take screenshots	Yes
Network monitoring, Har File	Yes (through Google Chrome Driver)
Memory and Performance Profiling	Yes (through Google Chrome Driver)
Code Coverage Analysis	Jacoco
Testing Types	E2E/Acceptance
SDLC Process Fit	
Plugins/Integrations	Cucumber, IntelliJ, Grunt, Jenkins, TeamCity, Hudson, Junit XML reporting built-in
BDD/ATDD Friendly	Yes (Jasmine, Mocha, Cucumber)
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	No
Execution Speed	Slower than headless
Sync	No
Feedback Loop and Reporting	
Reporters	HTML, Allure plugin, Junit XML
Community Strength	
Documentation	Nightwatch.JS
Contributors	59 contributors
Cloud and Automation at scale	
Cloud Support	Yes (web and mobile)

WebDriver.IO is the leading WebDriver binding for Node.JS. The framework basically sends requests to the Selenium server via the WebDriver protocol and manages the responses. Requests are wrapped in useful commands for ease of development and reuse for multiple test scenarios of your web site and more.

The integrated test runner lets you write asynchronous commands in a synchronous way so that you don't need to worry about how to handle a promise to avoid race conditions. Additionally, it takes away all the cumbersome setup work and manages the Selenium session for you.

Working with elements on a page is very easy due to its synchronous nature. When fetching or looping over elements you can use just native JavaScript functions. With the $ and $$ functions, WebdriverIO provides useful shortcuts which can also be chained to move deeper in the DOM tree without using complex xPath selectors.

The WDIO framework is easily integrated into many tools; therefore, during the config file setup, test automation engineers can specify their tool stack (Cucumber, Mocha, Jasmine, local vs. cloud, Selenium grid, and more).

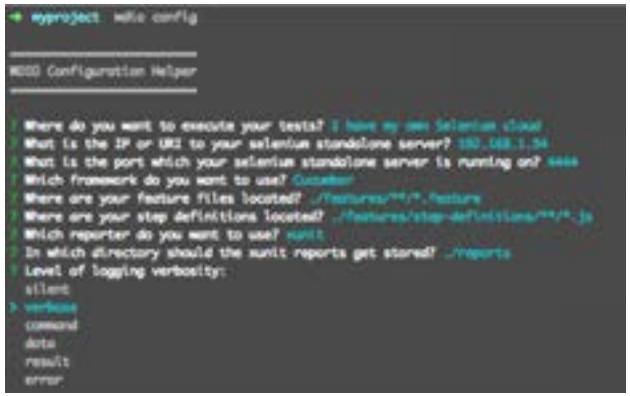


Fig 5: WebDriver.IO built-in command-line configuration wizard

Pro's

- **Automation Coverage:** Good for E2E functional testing
- **Fast Feedback and Reporting:** Good reporting plugins like Allure, Junit, Perfecto DigitalZoom™
- **Automation at Scale and Cloud:** Easily integrated into cloud testing solutions (e.g. Perfecto) for parallel testing
- **Community and Support:** Strong community backing the technology — integrations, plugins, support, documentation
- **Automation Robustness and Maintainability:** Sync-based testing supported
- **SDLC Process Fit:** BDD-friendly through Jasmine, Mocha, Cucumber and others — more standard FW to choose vs. proprietary
- **SDLC Process Fit:** Adopted by new emerging tools such as Chimp.JS
- **Automation Robustness and Maintenance:** Config file generation wizard supported, speeds up the testing setup (see image below)

Con's

- **SDLC Process Fit:** Less fit for angular-specific web sites; angular-specific object locating can be challenging
- **SDLC Process Fit:** No TypeScript support compared to Protractor, if this is a relevant requirement — worth mentioning

Automation Coverage	
Visual navigation testing	Using external tools
Take screenshots	Yes
Network monitoring, Har File	Yes (through Google Chrome Driver)
Memory and Performance Profiling	Yes (through Google Chrome Driver)
Code Coverage Analysis	Jacoco
Testing Types	E2E/Acceptance
SDLC Process Fit	
Plugins/Integrations	Community Plugins available
BDD/ATDD Friendly	Yes (Jasmine, Mocha, Cucumber)
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	Yes
Page Object Model Creation	Yes
Execution Speed	Slower than headless
Sync	Yes
Feedback Loop and Reporting	
Reporters	Allure, Junit, HTML, XML, Perfecto
Community Strength	
Documentation	WebDriver.IO
Contributors	250 contributors
Cloud and Automation at scale	
Cloud Support	Yes (web and mobile)

Protractor is an end-to-end test framework for **Angular and AngularJS applications**. Protractor runs tests against your application running in a real browser, interacting with it as a user would. This framework is highly supported by contributors from Google, the main leader behind AngularJS. This framework that is **built on top of Selenium** and is the most widely adopted framework, especially when testing Angular/AngularJS websites. In addition to **many extensions and plugins** that Protractor has for things such as visual testing and more, **Protractor has a unique synchronization mechanism** that can automatically execute the next step in your test the moment the webpage finishes pending tasks so you don't have to worry about waiting for your test and web page to sync.

Another great thing about Protractor is its ability to support BDD and Cucumber scripting. If you're leveraging BDD as part of your Angular/AngularJS website SDLC, then this framework should be a great fit for you.

Pro's

- **Automation Coverage:** Good for E2E functional testing of Angular-based websites
- **Fast Feedback and Reporting:** Good reporting plugins such as Allure, Junit, Perfecto DigitalZoom™
- **Automation at Scale and Cloud:** Easily integrated with cloud testing solutions (e.g. Perfecto) for parallel testing

- **Community and Support:** Strong community backing the technology — integrations, plugins, support, documentation
- **Automation Robustness and Maintainability:** Sync-based testing supported
- **SDLC Process Fit:** BDD-friendly through Jasmine, Mocha, Cucumber and others — more standard FW to choose vs. proprietary
- **SDLC Process Fit:** Supports TypeScript development

Con's

- **Automation Robustness and Maintainability:** If there is an issue with WebdriverJs layer (between Selenium and Protractor), the Protractor team should wait for the WebDriverJs team to fix that issue.

Automation Coverage	
Visual navigation testing	Using external tools
Take screenshots	Yes
Network monitoring, Har File	Yes (through Google Chrome Driver)
Memory and Performance Profiling	Yes (through Google Chrome Driver)
Code Coverage Analysis	Jacoco
Testing Types	E2E/Acceptance
SDLC Process Fit	
Plugins/Integrations	Various community plugins, Jenkins CI, Jasmine, Mocha, Cucumber, Visual Studio, WebStorm, Grunt
BDD/ATDD Friendly	Yes (Support Jasmine, Mocha, Cucumber)
Dev Language Support	JavaScript, TypeScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	Yes
Execution Speed	Slower than headless
Sync	Yes
Feedback Loop and Reporting	
Reporters	Jasmine2HTML, JUnit, Allure
Community Strength	
Documentation	Protractor
Contributors	234 contributors
Cloud and Automation at scale	
Cloud Support	Yes (web and mobile)

Codecept.IO is a JavaScript acceptance testing solution for Node.JS. In this solution, testing can be authored from an end-user perspective. Every command is described as an action of a user visiting a site.

In addition, Codecept supports various helpers like WebDriverIO, Protractor, Nightmare and Selenium WebDriver, enabling teams to extend their testing use cases.

Pro's

- **Automation Coverage:** Suits acceptance testing in a DSL/BDD language
- **Automation Robustness and Maintainability:** Synchronous test API's for more stable and linear tests
- **SLDC Process Fit:** Easy to write and understand each test due to the unique syntax the tests are written in
- **SDLC Process Fit:** Backend agnostic to multiple WebDrivers that are used
- **Automation Robustness and Maintainability:** Built-in dependency enables creation of a POM
- **Automation Robustness and Maintainability/SDLC Process Fit:** Uses JavaScript DSL for the BDD-based test authoring with common predefined functions
- **Automation at Scale and Cloud:** Among the large set of helpers that Codecept uses, there is also Appium to enable mobile testing in addition to Web.

Con's

- **SDLC Process Fit:** Not fully designed for mobile app testing even though Appium helper is available.
- **SDLC Process Fit:** Requires command line and/or wrappers to execute scripts in opposed to seamless IDE plugin
- **Automation Robustness and Maintainability:** Async testing solution
- **Community and Support:** Relatively small number of contributors
- **Automation Coverage:** Cannot perform visual navigation testing (OCR based testing)

Automation Coverage	
Visual navigation testing	No
Take screenshots	Yes
Network monitoring, Har File	Yes (through Google Chrome Driver)
Memory and Performance Profiling	Yes (through Google Chrome Driver),
Code Coverage Analysis	Jacoco
Testing Types	E2E/Acceptance
SDLC Process Fit	
Plugins/Integrations	Jasmine, Qunit, Travis CI, Jenkins, TeamCity, Buster.JS is built-in. This FW uses QtWebKit
BDD/ATDD Friendly	Mocha JavaScript DSL language
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	Yes
Execution Speed	Slower than headless
Sync	Yes
Feedback Loop and Reporting	
Reporters	CLI, XML, HTML
Community Strength	
Documentation	Codecept.JS
Contributors	81 contributors
Cloud and Automation at scale	
Cloud Support	Yes (web and mobile)

PhantomJS is a headless WebKit scriptable with a JavaScript API. It has fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG. Among its key benefits are screen capturing, network monitoring that captures an HAR file, as well as friendly script execution capabilities through Jasmine, Mocha and other tools. It's important to understand that PhantomJS itself is not a test framework, it is only used to launch the tests via a suitable test runner (e.g. Buster.JS has Phantom.JS as a supported tool in its solution).

Since this is a headless (and local) testing solution, it is compatible with various web development frameworks like jQuery, Bootstrap, CodeMirror, and others.

This tool is mostly beneficial for fast unit testing driven through CI or command line post code commits for fast feedback, and less for larger UI functional web testing. Google recently launched a Chrome headless browser that may be a suitable replacement for Phantom and which uses the Blink rendering engine, compared to the WebKit one used by Phantom.

Pro's

- **Automation Coverage:** Suits unit testing, mostly
- **Automation Coverage:** Additional automation artifacts such as HAR file and more
- **Automation Coverage:** Screen capture capabilities
- **Community and Support:** Great documentation and community support
- **Automation Coverage:** Strong APIs for various testing capabilities (filesystem, cookies, page size etc.)

Con's

- **Automation Coverage:** Cannot fit an E2E testing objective
- **Automation at Scale and Cloud:** Doesn't run on real browsers, automation at scale is an issue
- **SDLC Process Fit:** Doesn't support BDD as a built-in capability, tests can be triggered from BDD frameworks
- **SDLC Process Fit:** Outdated rendering engine compared to Google's headless solution

Automation Coverage	
Visual navigation testing	No
Take screenshots	Yes
Network monitoring, Har File	Yes
Memory and Performance Profiling	No
Code Coverage Analysis	No
Testing Types	Unit, Headless
SDLC Process Fit	
Plugins/Integrations	Mocha for test execution
BDD/ATDD Friendly	No, Phantom tests can be triggered from BDD frameworks
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	NA
Execution Speed	Fast
Sync	No
Feedback Loop and Reporting	
Reporters	Jasmine reporters, Karma reports
Community Strength	
Documentation	Phantom.JS
Contributors	147 contributors
Cloud and Automation at scale	
Cloud Support	No

JSDOM

JSDom is an in-JavaScript implementation of the DOM to be used with node.js. The DOM is the document object model, which is the tree of nodes that make up the UI for documents shown in web browsers. Among the leading contributors for this open-source you'll find developers from Google and whatwg.org. Because jsdom is implemented in JavaScript, we can have a DOM-like API to work with or without needing a browser. That means that we can run our tests in environments without browsers, such as in Node or in continuous integration environments.

Pro's

- **SDLC Process Fit:** Good for unit testing and for fast feedback on specified subset of your website implementation
- **Community and Support:** Large community of contributors
- **SDLC Process Fit:** Implements the standard WHATWG DOM
- **SDLC Process Fit:** Supports Mocha for scripting in an easy and readable syntax

Con's

- **Automation Coverage:** Cannot fit an E2E testing objective
- **Automation Coverage:** Does not target real browsers, doesn't test at scale or in a cloud based environment
- **Automation Robustness and Maintainability:** Asynchronous script loading. There is no way with JSDom to tell the user when it's a good time to run your code and inspect the resulting DOM structure since it has no real way to know when the entire page was loaded
- **Automation Coverage:** No screenshots or visual testing support
- **Fast Feedback Loop and Reporting:** Limited reporting capabilities, mostly leverages built-in console for reports

Automation Coverage	
Visual navigation testing	No
Take screenshots	No
Network monitoring, Har File	No
Memory and Performance Profiling	No
Code Coverage Analysis	No
Testing Types	Unit, Headless
SDLC Process Fit	
Plugins/Integrations	Karma
BDD/ATDD Friendly	No
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	Yes
Execution Speed	Fast
Sync	No
Feedback Loop and Reporting	
Reporters	Built in console for reporting
Community Strength	
Documentation	JSDom
Contributors	202 contributors
Cloud and Automation at scale	
Cloud Support	No

Chimp makes it super easy for developers to write automated tests by taking away all the pain associated with setting up tools and allowing developers to focus on building in quality. It does so by integrating and sprinkling magic over the following tools:

- Mocha, Jasmine *or* Cucumber.js
- Selenium *and* WebdriverIO
- Chai *or* Jasmine assertion libraries inside your steps
- Built in Node.js, works for any web application (*with special Meteor support*)

Pro's

- **Automation Coverage:** E2E/Acceptance solution, BDD friendly, Mix of supported tools
- **Automation Robustness and Maintainability:** Synchronized testing solution
- **SDLC Process Fit:** Large set of integrations to leading standard tools like Mocha, Jasmine, Cucumber, and most CI tools
- **SDLC Process Fit:** Innovative approach to agile/fast feedback loop through tools like Simian and Meteor
- **Automation at Scale and Cloud:** integrations to cloud supported for automation at scale
- **Automation Coverage:** Support for taking web screenshots
- **Fast Feedback Loop and Reporting:** Good assertions mechanism through Chai and others
- **Fast Feedback Loop and Reporting:** Supports debug mode for inspecting nodes through breakpoints and more.
- Automation at Scale: Mobile testing support through Appium

Con's

- **Automation Coverage:** Doesn't support visual navigation testing
- **Community and Support:** Community is ramping up, 40 contributors
- **Fast Feedback Loop and Reporting:** Basic reporting, doesn't provide a cross-platform reporting dashboard
- **SDLC Process Fit:** Setup and configuration seems complex

Automation Coverage	
Visual navigation testing	No
Take screenshots	Yes
Network monitoring, Har File	Yes (through Google Chrome Driver)
Memory and Performance Profiling	Yes (through Google Chrome Driver)
Code Coverage Analysis	Jacoco
Testing Types	E2E/Acceptance
SDLC Process Fit	
Plugins/Integrations	Mocha, Jasmine, Cucumber, Meteor, Most CI solutions, Simian, Chai, WebDriver.IO
BDD/ATDD Friendly	Yes
Dev Language Support	JavaScript
Automation Robustness and Maintainability	
Config File Generation	No
Page Object Model Creation	No
Execution Speed	Slower than headless
Sync	Yes (via wrappers)
Feedback Loop and Reporting	
Reporters	Relies on integrated FW reports like Mocha and others
Community Strength	
Documentation	Chimp
Contributors	40 Contributors
Cloud and Automation at scale	
Cloud Support	Yes

Summary

We have explored the web testing landscape with a special focus on the Selenium WebDriver API. Many open-source projects were reviewed, illustrating unique benefits, pros and cons. Choosing test frameworks should be determined by the combination of applying organizational and technical fit criteria.

While we mostly covered tools that offer E2E functional testing and unit testing, it is important to mention that there obviously are also tools for API testing, such as SoapUI and others, that were not in the scope of this research.

With the key considerations of technical fit and organizational fit in mind, Perfecto's Continuous Quality Lab in the cloud is architected to provide support for teams regardless of their open-source framework of choice or their development practice method. Perfecto's lab offers a large set of desktop VMs as well as smartphones and tablets in the cloud that can be targeted from your Nightwatch, WebDriverIO, Selenium, Protractor, Robot or other framework that you choose, in order to achieve automation at scale with maximum availability. At the end of execution, quality analysis is fast, with digital reporting that offers side-by-side execution dashboard with screenshots, testing artifacts such as HAR files, performance measurements, logs, and more. All tests can be orchestrated via CI engine such as Jenkins, CircleCI, TeamCity, Bamboo and others.

You should continuously evaluate the market and adjust your tool stack according to both market evolution and your own product's growth and its required capabilities; also, make sure you have the correct match for your web developers and testers.

Appendix

Recommended resources

The Top 8 Essential JavaScript Automation Frameworks — Joe Colantonio

Differences between nightwatch.js and webdriver.io — StackOverflow

Test frameworks comparison — Slant

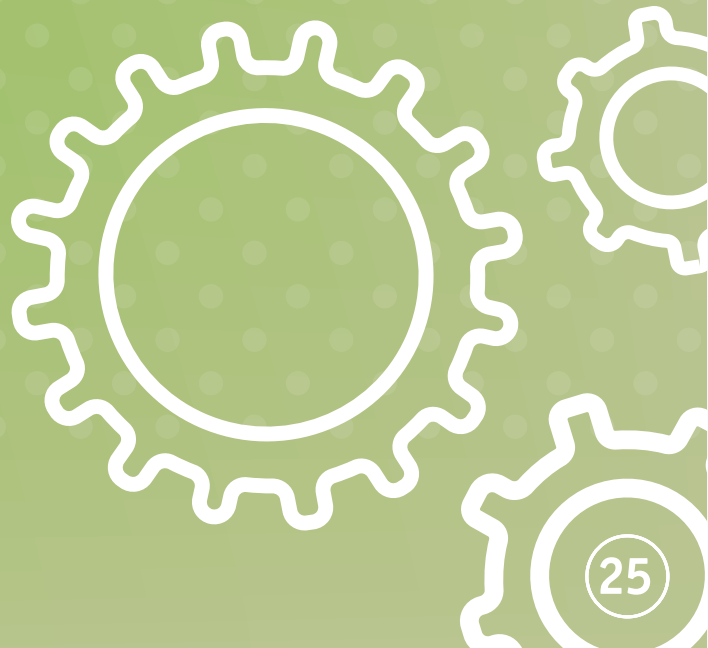
Protractor vs. Webdriver.io vs. Nightwatch.JS — WebDriverJS.com

Top 5 most rated node.js frameworks for E2E web testing — Adrian Lewis on Medium

An overview of JavaScript testing in 2017 — Vitalik Zaidman on Medium

How to automate web testing using open-source frameworks — Perfecto's slide share

Recommended JavaScript unit testing tools — Slant



Eran Kinsbruner

Director, Mobile Evangelist at Perfecto Mobile

About Perfecto

Perfecto enables exceptional digital experiences. We help you transform your business and strengthen every digital interaction with a quality—first approach to creating web and native apps, through a cloud—based test environment called the Continuous Quality Lab™. The CQ Lab is comprised of real devices and real end—user conditions, giving you the truest test environment available.

More than 1,500 customers, including 50% of the Fortune 500 across the banking, insurance, retail, telecommunications and media industries rely on Perfecto to deliver optimal mobile app functionality and end user experiences, ensuring their brand's reputation, establishing loyal customers, and continually attracting new users. For more information about Perfecto, visit www.perfectomobile.com, join our community follow us on Twitter at @PerfectoMobile.

Get content just like this delivered to your inbox!

Perfecto

Seek Perfection