www.testim.io

Comparing test automation frameworks

Also known as:

Puppeteer vs Selenium vs Cypress vs Playwright

Oren Rubin

SQGNE SEP 2020

ABOUT ME

Testim.io
CEO

Applitools
Director of R&D

Wix
Web Architect

Cadence
Compiler Engineer

IBM
Cloud Engineer

Mentor
Google Launchpad

Developer Expert
Google

External Lecturer
Technion University

Meetup Co-Organizer
Selenium IL, GDG, Ember.js

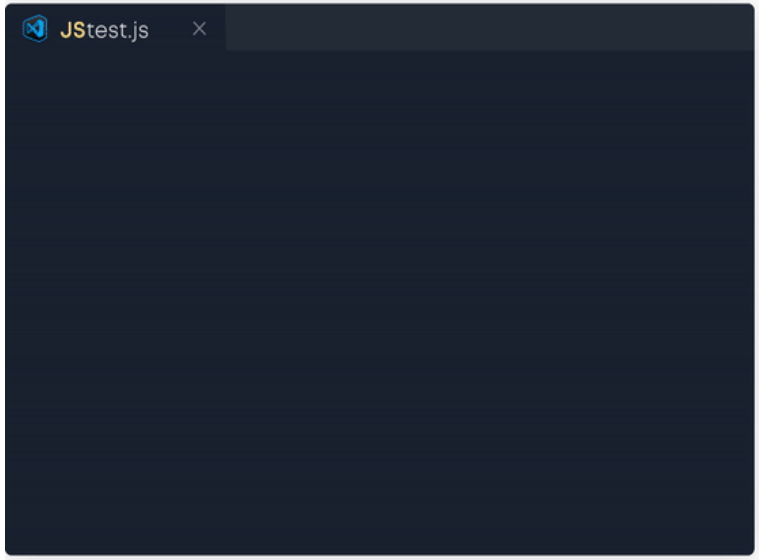
Testim

Tests that deliver

Testim is the FIRST AI-based test automation platform (2014).

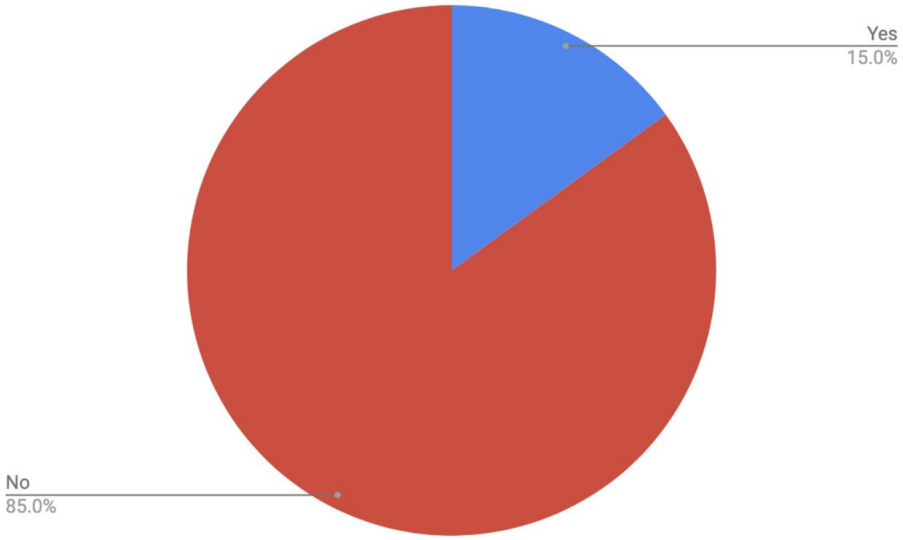
We use, integrate, and develop on top of many test infrastructure. We know them intimately.

Trusted by thousands of devs to author tests super fast, and auto-healing them.



So many test frameworks! why?

E2E tests as part as dev life cycle

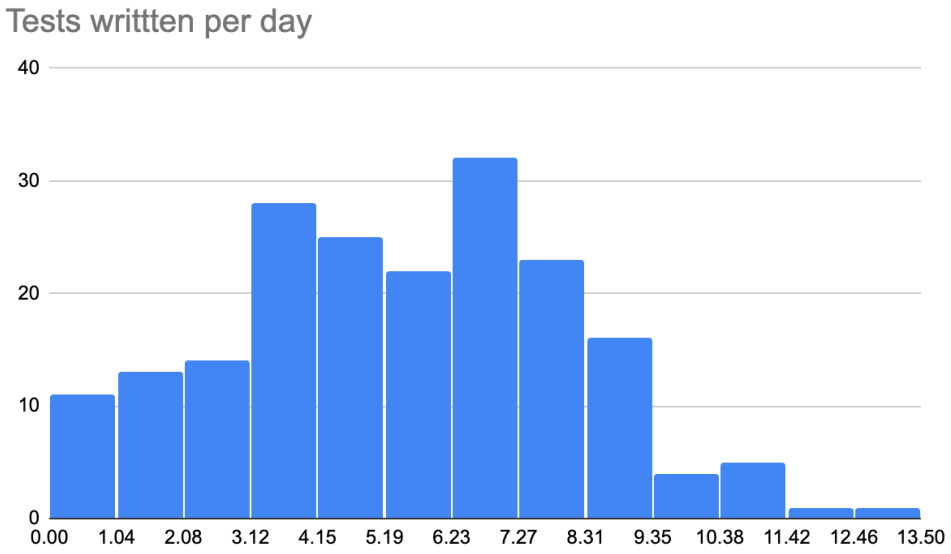


Measured across 284 companies.

Tests means automated E2E tests run as part of the dev cycle.

Tests Authoring Speed

Relatively stable and low amount of tests written. This is data of senior automation engineers:



Agenda

Part 1: Automation Frameworks Infrastructure

Part 2: Key Differences

Part 3: Takeaways

Part 4: Testim Playground & Root Cause

PART 01

Automation frameworks

Automation Frameworks

An automation framework **automates your browser**.

It allows simulating user actions in browsers like **clicks**.

Automation Frameworks

Before discussing frameworks. There are two fundamental ways to execute actions in browsers:

Through the **debugger** and through **executing JavaScript in the page**.

How Events Work

CAPTURE & BUBBLE

```
button.addEventListener("click",
    () => console.log("Ahoy!"),
    true
);
```

HTML

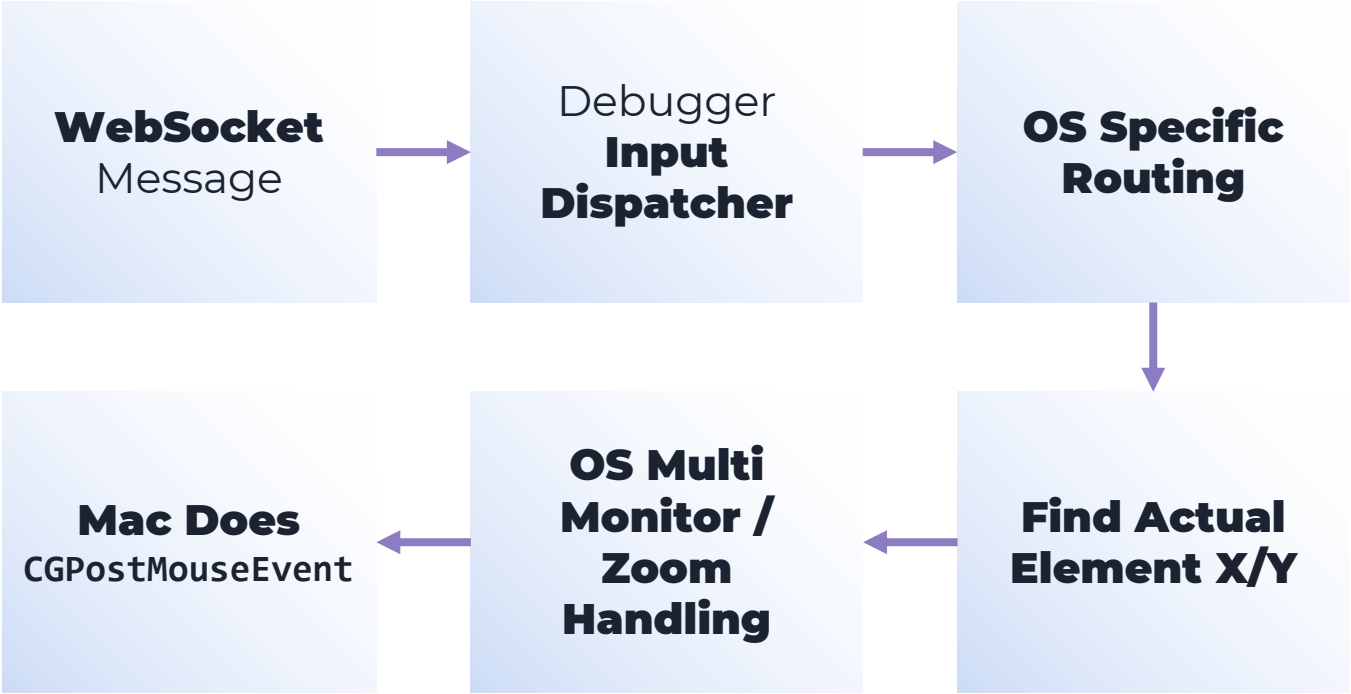
BODY

HEADER

BUTTON

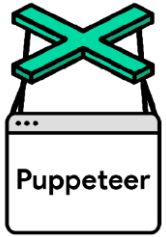
```
<html>
  <body>
    <header>
      <button>
        Hello
      </button>
    </header>
  </body>
</html>
```

Automation fires `Input.dispatchMouseEvent`



Automation Frameworks

Most popular automation frameworks



Selenium

A family of tools

- Selenium Webdriver : Automation Frameworks Infrastructure
- Selenium Grid: Execution Environment
- Selenium IDE: Basic Record/Playback

Testing Framework - Selenium

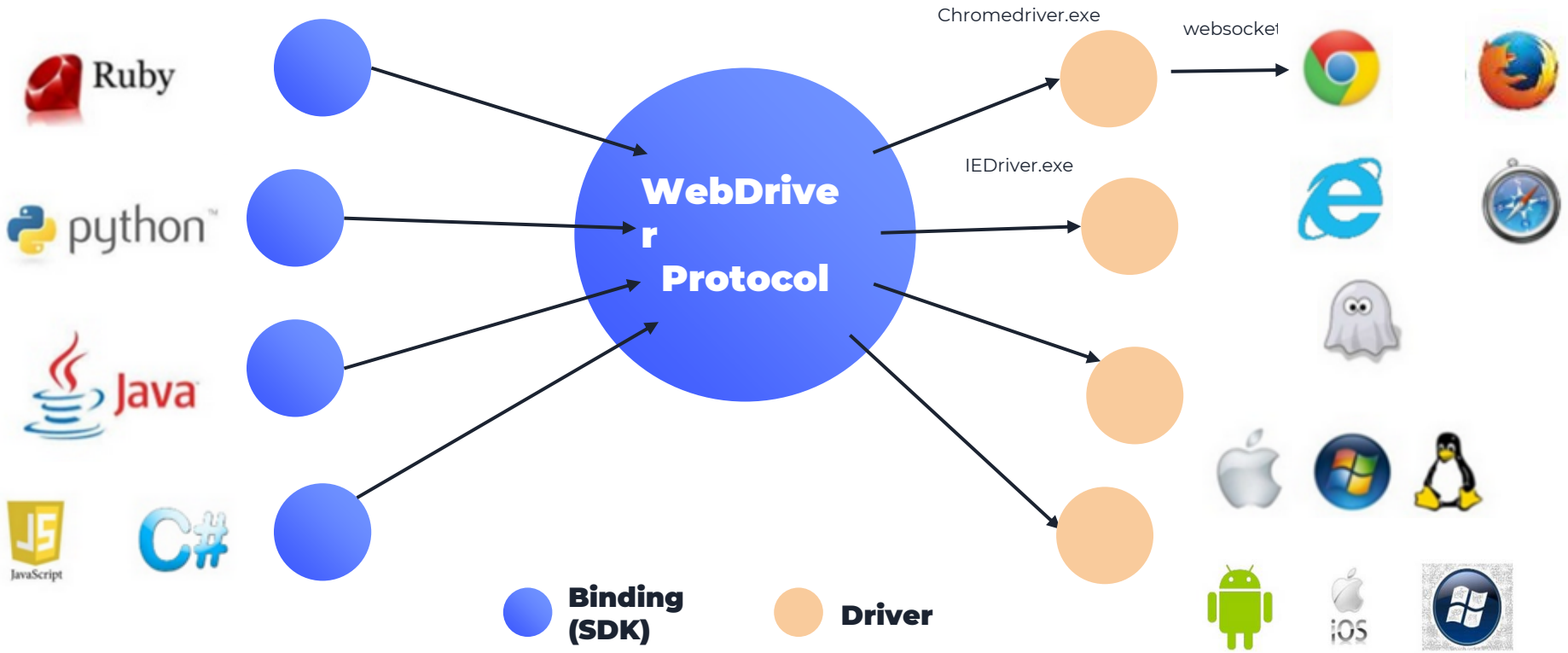
- By far the most popular framework for software testing.
- Open standard & open source
- Uses an **HTTP REST JSON** protocol for sending commands called the "Webdriver Protocol"
- https://www.w3.org/TR/webdriver/

Selenium Code

```
npm install selenium-webdriver
```

```
// await let driver = new Chrome(); // Open Google Chrome on THIS machine

await driver.get('http://demo.testim.io');
await driver.findElement({ css: 'button'}).click();
await driver.findElement({ css: '#login'}).sendKeys('Testim FTW!');
await driver.findElement({ css: '[type=pass]'}).sendKeys('12345');
await driver.findElement({ css: '[form=login]'}).click();
```

HTTP - The Universal Protocol



Selenium - Internals

ChromeDriver is an

HTTP Server

POST /session/

```
{
        "capabilities": {
                "browserName" : "Chrome",
                "browserValue": "Chrome"
        }
}
```

Selenium - Internals

POST /session/<session-id>

/element/<element-id>

/click

ExecuteElementClick

```
Status ExecuteClickElement(...) {

  Status status = GetElementTagName(...);

  events.push_back(MouseEvent(kMovedMouse, kNoneMouseButton));

  events.push_back(MouseEvent(kPressedMouse, kLeftMouseButton);

  events.push_back(MouseEvent(kReleasedMouse, kLeftMouseButton);

  status = web_view->DispatchMouseEvents(events)   session->GetCurrentFrameId());

  return status;

}
```

Selenium - Internals

ChromeDriver is an **HTTP Server**


```
CommandMapping(kPost,
"session/:sessionId/element/:id/click",

WrapToCommand("ClickElement"
base::BindRepeating(&ExecuteClickElement)
)),
```

DispatchMouseEvents

```
Status WebViewImpl::DispatchMouseEvents(events) {
 for (auto it = events.begin(); it != events.end(); ++it) {
   params.SetString("type", GetAsString(it->type));
   // …
   status = client->SendCommand("Input.dispatchMouseEvent", params);
 }
 return Status(kOk);
}
```

Selenium

Pros:

1. Runs on all browsers.

2. Many drivers and clients (language).

3. Dispatches clicks with debugger.

4. Lots of grid options.

Cons:

1. Not Bi-Directional* yet because it's an http server (Working on it now)

2. Harder to set up yourself than alternatives.

* Allows mock network, console log gathering on the fly, wait for idle network,..

Cypress

Cypress is a e2e testing framework.

It focuses on trying to provide good developer experience and an integrated environment.

Testing Framework - Cypress

Clicking in **Cypress** works like **Selenium 1** and dispatches **DOM Events Directly**

```
return _.extend({},
  mouseDownPhase.events,
  mouseUpPhase.events,
  mouseClickEvents
)
```

This is **flaky** for cross browser and cross site tests. It's part of **why selenium has its reputation**.

Cypress

No multi tab/window support, **no hover,** and **no cross frames nor Shadow DOM.**
Not modern JavaScript (e.g. no loops).
Only chaining

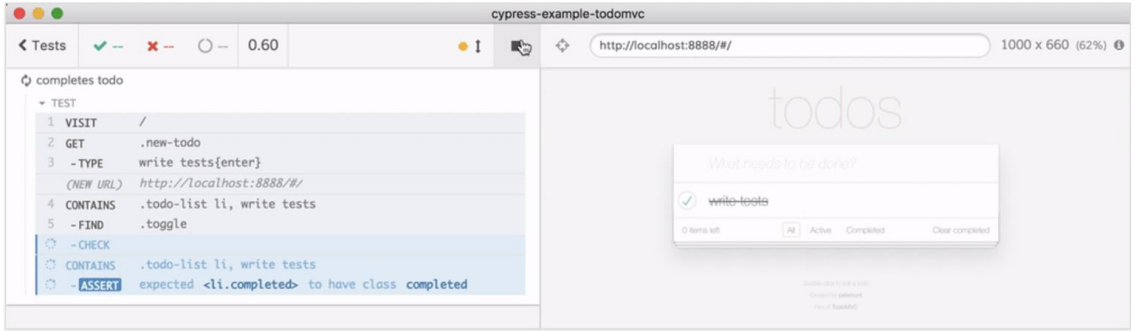
Debugging is less intuitive

```
it('completes todo', () => {
  // opens TodoMVC running at "baseUrl"
  cy.visit('/')
  cy.get('.new-todo').type('write tests{enter}')
  cy.contains('.todo-list li', 'write tests')
    .find('.toggle').check()

● cy.contains('.todo-list li', 'write tests')
    .should('have.class', 'completed')
})
```

Cypress

Expected

```
it('completes todo', () => {
  // opens TodoMVC running at "baseUrl"
  cy.visit('/')
  cy.get('.new-todo').type('write tests{enter}')
  cy.contains('.todo-list li', 'write tests')
    .find('.toggle').check()

  cy.contains('.todo-list li', 'write tests')
    .should('have.class', 'completed')
})
```



Cypress

Actual

```
it('completes todo', () => {
  // opens TodoMVC running at "baseUrl"
  cy.visit('/')
  cy.get('.new-todo').type('write tests{enter}')
  cy.contains('.todo-list li', 'write tests')
    .find('.toggle').check()

  cy.contains('.todo-list li', 'write tests')
    .should('have.class', 'completed')
})
```

Testing Framework -

Puppeteer

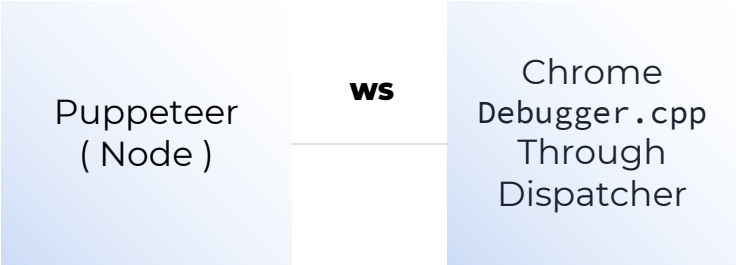
Puppeteer is a popular **test automation tool** maintained by **Google.**

It automates **Chrome** and **Firefox**. It is relatively **simple** and stable.

Fundamentally, puppeteer was intended to be an **automation tool.**

Puppeteer Architecture

Puppeteer is simple—it's just a **WebSocket client**



Puppeteer Architecture

Clicking in Puppeteer **does the same thing as ChromeDriver**

```
async click(x, y, options = {}) {
   const {delay = null} = options;
   this.move(x, y);
   this.down(options);
   if (delay !== null)
     await new Promise(f => setTimeout(f, delay));
   await this.up(options);
 }
```

Puppeteer Code

Puppeteer has a simple model for browser pages, which helps with stability.

```
await page.goto('http://demo.testim.io');
await page.click('button');
await page.type('#login', 'Testim FTW!');
await page.type('[type=password]', 'password');
await page.click('[form=login]');
```

Puppeteer Code

Feels like a thin wrapper around the CDP

Chrome DevTools Protocol Viewer

`Input.dispatchMouseEvent`

Dispatches a mouse event to the page.

PARAMETERS

`type`	**string** Type of the mouse event. ALLOWED VALUES: `mousePressed, mouseReleased, mouseMoved, mouseWheel`
`x`	**number** X coordinate of the event relative to the main frame's viewport in CSS pixels.
`y`	**number** Y coordinate of the event relative to the main frame's viewport in CSS pixels. 0 refers to the top of the viewport and Y increases as it proceeds towards the bottom of the viewport.
`modifiers` optional	**integer** Bit field representing pressed modifier keys. Alt=1, Ctrl=2, Meta/Command=4, Shift=8 (default: 0).
`timestamp` optional	TimeSinceEpoch Time at which the event occurred.
`button` optional	**string** Mouse button (default: "none"). ALLOWED VALUES: `none, left, middle, right, back, forward`
`buttons` optional	**integer** A number indicating which buttons are pressed on the mouse when a mouse event is triggered. Left=1, Right=2, Middle=4, Back=8, Forward=16, None=0.
`clickCount`	**integer**

Puppeteer

Pros:

1. Simple to set up, installs Chrome in a working version automatically.

2. Thin wrapper.

3. Bi-Directional (events).

4. Maintained by Google.

5. JS 1st

Cons:

1. Not cross-browser,

2. no easy grids.

3. Not cross-platform (userland projects exist).

Testing Framework - Playwright

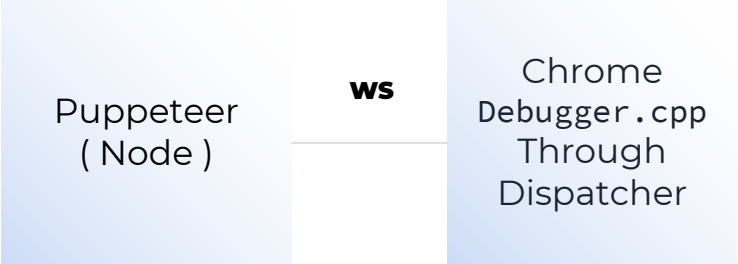
Playwright is a **new** popular **test automation tool** maintained by **Microsoft.**

It automates **Chrome, Safari** (WebKit) and **Firefox**.

It is written by people who previously worked on Puppeteer at Google.

Playwright Architecture

Playwright does the same thing as Puppeteer and is a **WebSocket client**



Playwright Code

Playwright uses syntax similar to Puppeteer with minor differences in construction.

```
await page.goto('http://demo.testim.io');
await page.click('button');
await page.type('#login', 'benjamin');
await page.type('[type=password]', 'password');
await page.click('[form=login]');
```

Playwright Differences

Playwright offers new features that are **test framework** rather than **automation framework** specific:

1. Automatically wait for elements to be available.

2. Built-in support for selecting elements by text.

3. Allows for isolated sessions on the same browser more easily.
   - Better iframe support using context

Playwright

Pros:

1. Simple to set up
2. Cross browser
3. test automation tooling
4. Improved test stability
5. well thought out API.

* easy to migration from/to puppeteer.

Cons:

1. No IE11,
2. no easy grids,
3. No plugin system (compared to Selenium)

PART 02

Key Differences

Disclaimer

Testim isn’t discussed below - but it supports all these features because it is an AI-based solution built on top of test automation frameworks which we consider infrastructure.

Feature: Cross Browser

✅✅✅✅ (Everything)

❌ (Only Chrome/Firefox)

❌ (Only Chrome/Firefox)

✅✅ (Chrome/Safari/Firefox)

Feature: Multiple Tabs

✅✅ (Simple switch API)

❌ No support.

✅✅✅✅ (Intuitive API)

✅✅✅✅ (Intuitive API)

Feature: Recording Tests

✅ Yes
(with Testim Playground / Selenium IDE)

❌ No support.
(in the future maybe)

✅ Yes
(with Testim Playground)

✅ Yes
(with Testim Playground & Now with Playground CLI)

Feature: Trusted Actions (e.g. hover)

☑ Yes

✕ No support, can use puppeteer plugin.

☑ Yes

☑ Yes

Feature: Parallelism Grids and Infrastructure

✅
Basic: Selenium Grid (OS Project)
Advanced: many grid providers

🧑‍🤝‍🧑 Only in their closed source paid cloud or build your own (github.com/agoldis/sorry-cypress).

❌ Usually: build your own infra.
Grid providers support coming soon!

❌ Usually: build your own infra.
Grid providers support coming soon!

Feature: Performance

- ☑ Fast enough, really.
- ☑ Faster in some cases
- ☑ Super fast!
- ☑ Super fast!

Key Findings

Tools	Total(sec)	Performance
Selenium - 4.0.0-alpha.1 (chromedriver - 74.0.0)	13.687	Average
WebdriverIO - 5.9.6 (chromedriver - 74.0.0)	5.447	Good
Testcafe - 1.2.0	20.370*	Basic
Cypress - 3.2.0	15.734	Average
Puppeteer	2.625	Excellent
Taiko - 0.8.0	6.556	Good

gauge.org/2019/08/21/how-taiko-compares-to-other-browser-automation-tools

Feature: Stability

✕ ☑ Complex Automatic Wait For mechanism.

✕ ☑ Complex mechanism that doesn't work with frames.

✕ ☑ Wait fors for certain things, but have to waitFor manually for others.

✕ ☑ ☑ Better wait fors for certain things, but have to waitFor manually for others.

Feature: Smart Locators

❌ ❌ ❌ No support for selecting elements in multiple ways

❌ ❌ ❌ No support for selecting elements in multiple ways

❌ ❌ ❌ No support for selecting elements in multiple ways

❌ ❌ ✅ A start of supporting custom selector engines.

Feature: Debugging

✕ ☑ A bit hard to figure out all the terminology. Debugging remote grids relies on the grid provider.

✕ ☑ You're not even writing modern JavaScript you're chaining promises.
- Makes up with DOMs.

☑ Writing and debugging JavaScript from your IDE

☑ Writing and debugging JavaScript from your IDE

Feature: Self-Healing Tests

✕ No.

✕ No.

✕ No.

✕ No.

Feature: Docs + Resources

✅✅✅ Very large community.
Many testers, tutorials

✅✅ Small community but super buzz
- and very nice documentation.

✅ Small community but lots of
tutorials at this point

✅ ❌ Some docs and tutorials out of date
due to changing API.
- Most accurate guides at playwright.tech

Feature: Autonomous Testing

✕ No.

✕ No.

✕ No.

✕ No.

PART 03

Summary

Automation has **a lot of tradeoffs**.

Test automation tools are **different from each other** with each containing **pros and cons**.

It makes sense to **mix and match** and use tools together.

Chrome File Edit View History Bookmarks People Tab Window Help

Puppeteer, Selenium, Playwrig

testim.io/blog/puppeteer-selenium-playwright-cypress-how-to-choose/

testim PRODUCT DEVELOPERS COMPANY RESOURCES PRI

2. When you are done adjusting the weights, press "calculate score" to see the weighted average scores.

CATEGORIES	IMPORTANCE	PUPPETEER	PLAYWRIGHT	SELENIUM	CYPRESS
Cross Browser	Medium (3)	2	3	5	2
Multi-tab & frames	Medium (3)	5	5	3	0
Authoring speed	Medium (3)	4	4	4	3
User simulation fidelity	Medium (3)	4	4	4	3
Parallelism, Grids, & Infra.	Medium (3)	0	0	4	2
Execution speed	Medium (3)	5	5	3	4
Stability	Medium (3)	3	4	3	3

testim.io/blog/puppeteer-selenium-playwright-cypress-how-to-choose/

PART 03

Testim Playground & Root Cause

testim.io/playground

&

testim.io/root-cause/
(github.com/testimio/root-cause)

IT'S A WRAP!

We're Hiring

testim.io/careers

or

oren@testim.io